

Translations

Translations

Anonymous · 21/10/2026

Translations

`Derafu\Twig\Extension\TranslationExtension` brings message translation to Twig templates, powered by [derafu/translation](#). It mirrors the structure of Symfony's own Twig translation integration (a `trans` filter, a `t()` function, a `{% trans %}` block tag, and a `{% trans_default_domain %}` tag), but without depending on `symfony/twig-bridge` — that package's only hard requirements are `symfony/translation-contracts` and `twig/twig`, but it still ships 55 files (~500KB) of `Csrf`, `Form`, `HttpKernel`, `Security`, `Serializer`, and `Workflow` extensions this library has no use for. `TranslationExtension` depends only on `derafu/translation` instead.

Like the rest of this ecosystem, it does nothing unless you explicitly wire a translator — without one, everything still renders, using ICU formatting directly instead of a translation catalogue.

Registering It

`TwigCreator` (and so `TwigService`) always registers a `TranslationExtension`, so the templates can use `trans` without any setup. That one has no translator: everything renders in English (see [Fallback Without a Translator](#)). To translate, give your own instance, with the translator, in the `extensions` option, the same way you would add any other Twig extension. The one you give is the one that is used, and the default one is not registered:

```
use Derafu\Twig\Extension\TranslationExtension;
use Derafu\Twig\Service\TwigService;
use Derafu\Translation\TranslatorFactory;

$translator = TranslatorFactory::create(
    defaultLocale: 'es',
    fallbackLocales: ['es', 'en'],
);

$twigService = new TwigService([
    'extra' => false, // Keeps this example scoped to just this extension.
    'extensions' => [
        new TranslationExtension($translator, 'app+intl-icu'),
    ],
]);
```

The constructor takes three optional arguments:

```
final class TranslationExtension extends AbstractExtension
{
    public function __construct(
        ?TranslatorInterface $translator = null,
        ?string $domain = null,
        ?string $locale = null,
    ) {
    }
}
```

`$domain/$locale` are fallbacks used only when a call site doesn't specify its own — they are never assumed or required. Prefer `{% trans_default_domain %}` (below) to set a domain per template instead of relying on this constructor argument, since a single `TranslationExtension` instance is shared by every template rendered through that Twig Environment.

The `trans` Filter

```
{{ 'Hello {name}!'|trans({'name': user.name}) }}
{{ 'Hello {name}!'|trans({'name': user.name}, 'app+intl-icu') }}
{{ 'Hello {name}!'|trans({'name': user.name}, 'app+intl-icu', 'fr') }}
```

Placeholders use ICU syntax (`{name}`), the same convention used everywhere else in this ecosystem — never `%name%`. The domain should generally carry the `+intl-icu` suffix (see [the ICU MessageFormat guide](#)); without it, the underlying translator falls back to plain `strtr()`, which does not understand `{name}`-style placeholders.

A translatable value as the message

The filter also takes a value that is already translatable: a `TranslatableMessage`, what `t()` makes, or a translatable exception. It brings its own id, parameters and domain, so it is translated as it is:

```
{{ message|trans }}           {# A TranslatableMessage, for example a flash
message. #}
{{ t('Close', {}, 'app+intl-icu')|trans }}
{{ error|trans }}             {# A translatable exception. #}
{{ message|trans({}, null, 'fr') }} {# The locale can be given. #}
```

- Without a translator the value is its text. For an exception, it is its message, never the dump

that PHP makes of it (class, file, line and trace).

- **Parameters can not be given** for a value that is already translatable: they would not be used, so the filter throws a `TwigException`, like Symfony's does. The domain is not checked, because the `{% trans_default_domain %}` tag puts one in every call; the value keeps its own.
- A translatable value is also accepted as a **parameter** of a text, and it is translated too: `{{ 'Failed: {reason}'|trans({'reason': error}) }}`.
- `null` and an empty text give an empty text. Any other `Stringable` is translated as a text.

The `t()` Function

```
{% set message = t('Hello {name}!', {'name': user.name}, 'app+intl-icu') %}
```

Builds a `Derafu\Translation\TranslatableMessage` without translating it — useful to hand a translatable value to PHP code that will translate it later, possibly with a different locale.

To translate it right there, apply the filter: `{{ t('Close')|trans }}`.

Printing it directly does not use the configured translator. `{{ t(...) }}` casts the object to a string via `__toString()`, which always applies ICU formatting only — the same fallback used when no translator is configured at all. To translate a message immediately within a template, use the `trans` filter or the `{% trans %}` tag below, both of which call the translator directly; reach for `t()` only when the translation needs to happen somewhere else, later.

The `trans` Block Tag

```
{% trans %}Save{% endtrans %}
{% trans with {'name': user.name} %}Hello {name}!{% endtrans %}
{% trans with {'name': user.name} from 'app+intl-icu' %}Hello {name}!{% endtrans %}
{% trans with {'name': user.name} from 'app+intl-icu' into 'fr' %}Hello {name}!{% endtrans %}
```

Unlike Symfony's own `trans` tag, parameters are **never** auto-detected by scanning the message text for `%name%` occurrences — that convention predates ICU and is incompatible with it. Here, parameters always go through `with`, explicitly, and placeholders in the text are ICU (`{name}`), matching the `trans` filter.

The message must be plain text. Interpolating a Twig expression (`{% trans %}Hello {{ user.name }}!{% endtrans %}`) raises a `SyntaxError` at compile time: the translation id would depend on a runtime value and would never match an entry in the translation catalogue.

Never use the bare # ICU plural shorthand — in any of `trans`, `t()`, or `{% trans %}`. A literal `{#` in raw template text is always lexed by Twig itself as the start of a `{# comment #}`, before this extension's code ever runs, breaking the whole template outright (or, worse, silently consuming content up to the next stray `#`). That alone rules out the `{% trans %}` tag for any ICU plural/select message using `#` (`{count, plural, one {# item} other {# items}}`), since its body is raw template text.

A `{#` inside a Twig *string literal* — e.g. the message argument to the `trans` filter — is lexed literally and is safe **when only Twig's own extensions are registered**. That safety is not guaranteed once `symfony/ux-twig-component` is also registered (the runtime behind `<twig:...>` component tags): a real case surfaced where the exact same `{#`, inside a `trans` filter's string argument, in a template that both used a `<twig:...>` component and `{% extends %}`, broke the template with an unrelated-looking error (A template that extends another one cannot include content outside Twig blocks) — nothing in the message pointed at `{#` as the cause. The exact mechanism inside `ux-twig-component` was not identified (it is a third-party dependency), but bisection on the real template confirmed the `{#` substring as the trigger.

The rule that holds regardless of which extensions are registered: never write the bare `#` shorthand in an ICU plural/select message. This applies to `trans`, `t()`, and `{% trans %}` alike.

The fix is **not** to reference the same argument again as `{count}`: real ICU (PHP's `MessageFormatter`, used both by the no-translator fallback and, through Symfony, by a real translator) throws `U_ARGUMENT_TYPE_MISMATCH` when the same argument name is used both as the plural selector and as a literal reference inside a branch — `#` exists specifically to avoid that conflict, it is not interchangeable with a named reference to the same argument. The safe fix is to pass the same value under **two different argument names**: one used only for plural selection, one only for display:

```
{{ '{count, plural, one {{n}} item} other {{n}} items}}'|trans({'count': items|length,
'n': items|length}, 'app+intl-icu') }}
```

The `trans_default_domain` Tag

Sets a default domain for every `trans` filter call and `{% trans %}` block that follows it in the same template or block, and doesn't have its own domain:

```
{% trans_default_domain 'app+intl-icu' %}

{{ 'Save'|trans }}
{% trans %}Cancel{% endtrans %}
{{ 'Delete'|trans({}, 'admin+intl-icu') }} {# explicit domain still wins #}
```

The domain doesn't need to be a literal string — a dynamic expression works too, evaluated once regardless of how many `trans` calls use it:

```
{% trans_default_domain app.locale == 'es' ? 'es+intl-icu' : 'en+intl-icu' %}
```

The scope is the enclosing template or `{% block %}` — a `trans_default_domain` set inside a block does not leak to the rest of the template, and the template's own domain (if any) is restored once the block ends.

Fallback Without a Translator

`trans`, `t()`, and `{% trans %}` all work with no `Translator` configured at all — every one of them falls back to `TranslatableMessage::__toString()`, which still applies ICU formatting (interpolating `{name}`-style placeholders correctly) even though nothing gets translated:

```
{# No translator registered: renders "Hello Juan!" literally. #}  
{{ 'Hello {name}!'|trans({'name': user.name}) }}
```

This is unlike `symfony/twig-bridge`, whose own no-translator fallback uses a plain `strtr()`-based identity translator that does not understand ICU placeholders at all.

Last updated on 21/10/2026