

Docs » Views and Templates Category » Twig Project » Lint

Lint

Lint

Anonymous · 08/10/2026

Lint

Derafu\Twig\Lint has tools that read templates to find problems. They are for tools and tests, not for the code that renders (see [Code Quality](#)).

There are tools for [route references](#) and for [translations](#).

Route references

A template that links to a route that does not exist only fails when that link is rendered, which can be never for a long time (for example, a link inside a loop that never runs). `RouteReferenceScanner` finds every route a template refers to, without rendering anything, so a test can check them.

It finds the calls to `path()`, `url()` and `is_active_path()`:

```
use Derafu\Twig\Lint\RouteReferenceScanner;

$scanner = new RouteReferenceScanner($twig); // A Twig\Environment.

$references = $scanner->scanDirectory(__DIR__ . '/../templates');

foreach ($references as $reference) {
    $reference->function; // 'path', 'url' or 'is_active_path'.
    $reference->name;      // 'docs_doc', or null when it is not a literal.
    $reference->template;  // 'docs/show.html.twig'.
    $reference->line;      // 12.
    $reference->isDynamic();
}
```

`scanTemplate('docs/show.html.twig')` does the same for one template, and `scanDirectory()` takes every `.twig` file of a directory, in order of template and then of line. The directory must be one of the paths of the loader of the environment.

Checking that the routes exist

The scanner only finds references. Checking them is up to you, with the router of the templates you scanned:

```
$undefined = array_filter(
    $references,
    fn ($reference) => !$reference->isDynamic() && !$router->has($reference->name)
);

$this->assertSame([], $undefined);
```

Use the router that serves **those** templates. A template of a package that is only rendered when you import one of its optional route files (a login template that needs the routes of the database provider, for example) is not an error in an application that never renders it. That is why the scanner does not decide.

How it works

- It reads what Twig parses, with the environment the templates are written for, so comments, texts and strings that only look like a call are ignored, and calls inside macros, filters or other calls are found.
- It recognizes the functions by what they are (they belong to `RoutingExtension`), not by their name. A `path()` from another extension is not a route.
- It uses the environment you give it. Use the real one, the one with the functions, tags and components of your templates, for example the one of your renderer (`$renderer->getEngine('twig')->getTwig()`). A template it can not parse raises Twig's `SyntaxError`, with the name of the template.
- If the environment has no `RoutingExtension`, it raises a `LogicException`: finding nothing would look like a clean result.
- The `line` of a reference is the line of the file (see [The lines of a template](#)).

Dynamic names

When the name is not a literal (`path(item.route.name)`), it can only be known when the template is rendered. The reference has `name` as `null` and `isDynamic()` returns `true`. They are reported so they are visible, and they can not be checked by reading.

Translations

Everything a package shows to people must be translatable (see the [Lint of derafu/translation](#) for the same check on code). In templates that means two things: the messages that go through the translation must have their entry in the catalogue, and the texts that are written in the template must go through the translation. Three tools read it, and `TwigTranslationAudit` puts them together with the audit of the code.

If the conventions of this section are not followed, the templates can not be validated. The tools do not guess: what they can not read is reported, and the test of the

package fails until the template is written in a way that can be read. Nothing is skipped silently.

The audit

`TwigTranslationAudit` audits a package that has templates in one call. It uses `Derafu\Translation\Lint\TranslationAudit` for the code, and adds the templates to the same report: the messages of the templates are checked against the same catalogues, and an entry that only a template uses is not reported as left over. A package without templates does not need it: it uses `TranslationAudit`.

```
use Derafu\Twig\Lint\TwigTranslationAudit;

$report = (new TwigTranslationAudit())->audit(
    __DIR__ . '/../src',
    __DIR__ . '/../resources/templates',
    new MyPackageTranslationResourceProvider(),
    $twig, // A Twig\Environment that can parse the templates.
);

$this->assertFalse($report->nothingFound);
$this->assertSame([], $report->describe($report->dynamicMessages));
$this->assertSame([], $report->describe($report->missingTranslations));
$this->assertSame([], $report->describe($report->notUsedBySources));
$this->assertSame([], $report->describe($report->notTranslatable));
$this->assertSame([], $report->describe($report->untranslatedTexts));
```

Argument	What it is
<code>\$directory</code>	The code to read.
<code>\$templates</code>	The templates to read. It must be one of the paths of the loader of the environment.
<code>\$providers</code>	The catalogues of that package : one <code>TranslationResourceProviderInterface</code> or an iterable of them.
<code>\$twig</code>	The environment the templates are written for, with its extensions and components. It must have the <code>TranslationExtension</code> .
<code>\$locale</code>	The locale whose catalogue is checked. <code>es</code> by default.
<code>\$allowedThrowables</code> , <code>\$messageMethods</code>	The same as in <code>TranslationAudit</code> .
<code>\$defaultDomain</code>	The domain of the messages that do not have one: the one of the <code>TranslationExtension</code> of the environment (messages if it has none).
<code>\$allowedTexts</code>	Texts that a template writes and are allowed to not be translated, by the whole text (a brand name, for example). It is an explicit decision of the package.

Argument	What it is
<code>\$textFormats</code>	The formats of the templates whose texts are read: <code>html</code> and <code>pdf</code> by default. The templates of other formats (<code>*.md.twig</code> , <code>*.xml.twig</code>) are not HTML, so their texts are not read. The messages are read in all of them.

The report has the facts of `TranslationAuditReport` (`dynamicMessages`, `missingTranslations`, `notUsedBySources`, `notTranslatable`, `nothingFound`), now for the code **and** the templates, and one more: `untranslatedTexts`, the texts that the templates write and that do not go through the translation. `describe()` turns a list of findings into lines for a failed test, with the path relative to the templates (`components/block-alert.html.twig:12 "Close" [aria-label]`).

Conventions for the templates

1. **Every text that a person reads goes through the translation**, with the text in English as its id (the `title`, `alt` and `aria-label` attributes too).
2. **The id is a literal text**. The tool reads the text, so the text has to be there. `text|trans` or `('Hello ' ~ name)|trans` can not be checked: they are reported as dynamic messages.
3. **A value goes in a parameter, not in the id**. The text around a value written in the template is a text of the template, so it is reported.
4. **The domain is a literal**, set for the whole template with `{% trans_default_domain 'twig+intl-icu' %}` or by the call `('Close'|trans({}, 'twig+intl-icu'))`. A domain that is not a literal is reported as dynamic. The domain of a package is its name (`twig`, `content`, `form`), and its catalogue is `resources/translations/<domain>+intl-icu.es.php`.
5. **t() does not use the default domain of the template**. It does not when the template is rendered either, so write its domain: `t('Close', {}, 'twig+intl-icu')`.
6. **The texts that a template gets from code** (a component that builds a text in PHP) are messages of the code: they follow the conventions of [derafu/translation](https://www.derafu.dev/translation).

For example:

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

<button aria-label="{{ 'Close'|trans }}"></button>
<span>{{ 'Previous'|trans }}</span>
<img alt="{{ 'Slide {number}'|trans({'number': loop.index}) }}">
```

What not to do

Do not	Why	Do
<code><button aria-label="Close"></code>	A text that is written in the template, and a person reads it.	<code>aria-label="{{ 'Close' trans }}"</code>

Do not	Why	Do
<code>Anterior</code> (or any language but English)	The id is English: it is what shows when there is no translation.	English in the template, Spanish in the catalogue.
<code>alt="Slide {{ n }}"</code>	The text around a value is a text.	<code>alt="{{ 'Slide {number}' trans({'number': n}) }}"</code>
<code>{{ title trans }}</code>	The id is only known when the template is rendered.	Translate the literal where it is written, or translate in the code that gives it.
<code>{% trans_default_domain domain %}</code>	The domain is only known when the template is rendered.	A literal domain.
An entry in the catalogue that no template and no code uses	It is reported as left over.	Remove it.

The pieces

`TwigTranslationAudit` is made of two scanners that can be used on their own. Both follow the conventions of `RouteReferenceScanner`: `scanTemplate()` and `scanDirectory()`, the environment given to the constructor, and errors that are not hidden.

TemplateMessageScanner

Finds the messages that the templates translate: the `trans` filter, the `t()` function and the `{% trans %}` tag. It gives `Derafu\Translation\Lint\MessageReference`, the same reference that `derafu/translation` finds in code, so it is checked in the same way.

```
use Derafu\Twig\Lint\TemplateMessageScanner;

$scanner = new TemplateMessageScanner($twig, defaultDomain: 'app');

foreach ($scanner->scanDirectory($templates) as $reference) {
    $reference->id;           // 'Close', or null when it is not a literal.
    $reference->domain;       // 'twig+intl-icu', or null when it is not a literal.
    $reference->file;         // The path of the template.
    $reference->line;
    $reference->isDynamic();
    $reference->identity();   // For a dynamic message: the template and the line of
    the call.
}
```

- It reads what Twig parses, so comments, texts and strings that only look like a call are ignored, and calls inside macros, filters, branches and loops are found.
- It recognizes the calls by what they are (they belong to `TranslationExtension`), not by their

name.

- `t('Close')` | `trans` is one message, the one of `t()`: the filter that translates what `t()` made is not another message, and it is not reported as dynamic. Only a `t()` call that is directly the input of the filter is recognized.
- The domain of a message is the one of the call, or the one set with `{% trans_default_domain %}`, or `$defaultDomain`, or `messages`. It is what the template renders with.
- If the environment has no `TranslationExtension` it raises a `LogicException`: finding nothing would look like a clean result.

TemplateTextScanner

Finds the texts that the templates write and that a person can read, without going through the translation: the text between the tags of the HTML and the value of the attributes that are shown to people.

```
use Derafu\Twig\Lint\TemplateTextScanner;

foreach ((new TemplateTextScanner($twig))->scanDirectory($templates) as $text) {
    $text->text;        // 'Close'.
    $text->kind;        // 'text', or the name of the attribute ('aria-label').
    $text->template;    // 'components/block-alert.html.twig'.
    $text->file;
    $text->line;
}
```

- It reads the templates that write HTML: the ones whose name says `html` or `pdf` before `.twig` (`page.html.twig`), or that do not say a format (`page.twig`). A template of another format (`list.md.twig`, `feed.xml.twig`) is skipped, because its text is not HTML. Another list can be given to the constructor (`formats:`).
- The attributes that it reads are `TemplateTextScanner::ATTRIBUTES` (`alt`, `title`, `aria-label`, `aria-description`, `aria-placeholder`, `placeholder`, `label`, `data-bs-title`, `data-bs-content`). Another list can be given to the constructor.
- A text is made of letters of any alphabet: punctuation, numbers and symbols are not found.
- What a template prints is not a text of the template, but it is known to be there: `Slide {{ loop.index }}` is the text `Slide`.
- The content of `<script>` and `<style>`, the comments of the HTML and the doctype (`<!DOCTYPE html>`) are not read: they are not shown.
- Whether a text has to be translated is for whoever uses it to decide: a brand name does not.

Known limits

These are things the tools can not see. They are not bugs: they are what reading templates, without rendering them, can know.

1. **The texts that the tool finds are found by rules.** Recognizing a message that goes through the translation is exact, because it reads what Twig parses. Recognizing a text that does not is

not: it reads the HTML that the template writes, with a list of attributes.

2. **Texts that do not come from the template are out of reach.** A value that a template prints (`{{ this.title }}`), a text given to a component as a property (`<!--TWIG_BLOCK_0-->`), the text of a variable: they are data. The code or the template that gives them is the one to translate them.
3. **Anything inside `<script>` and `<style>` is not read.** A text that JavaScript shows to people is not found.
4. **Attributes that are not in the list are not read**, and the list is of attributes whose name says that a person reads them.
5. **Templates are read one by one.** An included or embedded template is read as its own file, if it is in the directory that is scanned. One that is in another directory is only read if that directory is scanned too.
6. **`t()` and `trans` need a literal id.** A message that is not a literal is reported, and it can not have an entry in a catalogue.
7. **It checks that the entry exists, not that it is good.** A catalogue entry with the wrong translation, or a parameter missing, passes the audit. It is for a person to review.
8. **The environment has to be able to parse the templates.** A function, filter, tag or component that the environment does not have makes Twig raise its own `SyntaxError`, with the name of the template, and the audit stops.
9. **The audit reads one directory of templates**, and it must be one of the paths of the loader.
10. **The texts of the Markdown and XML templates are not checked.** The tool reads HTML. The messages that those templates translate (`trans`, `t()`) are read as in any other.
11. **The line inside a tag of several lines of a component is the line where the tag starts** (see below).

The lines of a template

The scanners say the line of the file, not the one that Twig counts. The environment writes again the code of a template before Twig parses it: the `<twig:...>` tags of the components become tags of Twig, and a tag of several lines becomes one line. Twig then counts the lines of that code, so every line after the first component of a template is off (upstream, Twig itself reports its errors in those lines too).

The scanners match the lines that Twig parsed with the lines of the file, so a finding has its real line. What is inside a tag of several lines (a message written in a property of the component) has the line where the tag starts, because the tag is one line for Twig and its inside can not be told apart. The `expression` of a dynamic message is the text of the line as Twig parsed it.

Twig also copies some nodes (for example the one of a filter that is used twice, as in `name|default('Hello'|trans)`). A message or a route that Twig copies is reported once.