Introduction

Translation Library with Exception Support

Anonymous · 09/09/2026

Translation Library with Exception Support

A small library that adds two things on top of `symfony/translation`:

1. **Translatable exceptions**: exceptions that carry a message, ICU parameters and a translation domain, and can render themselves in English (or any locale) with zero setup, or be translated later by handing them a real translator.
2. **A resource discovery layer**: `symfony/translation` doesn't provide a way to discover translation files in a directory on its own (that's normally wired by a full framework). This library adds that missing piece as a small, standalone component.

Everything else — ICU message formatting, locale fallback, file loaders (YAML, JSON, PHP, XLIFF, PO, MO, CSV, INI) — comes directly from `symfony/translation` itself. This library does not reimplement a translation engine.

Features

- **Translatable Exceptions**: A full hierarchy of exceptions (mirroring PHP's SPL exceptions) that support translation with minimal setup.
- **ICU Support**: Powered by PHP's `intl` extension via `symfony/translation`'s own ICU formatter. Works even without a translator configured.
- **Multi-Directory Resource Discovery**: Register N directories (or a tagged collection of providers, if you use a DI container) and have every translation file inside them loaded automatically.
- **Locale Fallback**: Configurable fallback locale chain, courtesy of `symfony/translation`.
- **Lightweight**: The only hard dependencies are `symfony/translation`, `symfony/translation-contracts`, `symfony/yaml`, and the `intl` PHP extension. No framework, no `symfony/http-kernel`.

Installation

```
composer require derafu/translation
```

Basic Usage

Making Exceptions Translatable

```
// Before: your existing exception.
class ValidationException extends Exception
{
}

// After: extend one of the translatable base exceptions instead.
use Derafu\Translation\Exception\Core\TranslatableException;

class ValidationException extends TranslatableException
{
    // No further changes needed.
}
```

Using Translatable Exceptions

```
// 1. A plain string: used as both the message and the translation id.
throw new ValidationException('Email is invalid.');

// 2. An array: first element is the message/id, the rest are ICU
//    parameters keyed by name.
throw new ValidationException([
    'The value {value} is not a valid email.',
    'value' => 'test@example',
]);

// 3. A TranslatableInterface instance, built directly.
use Derafu\Translation\TranslatableMessage;

throw new ValidationException(new TranslatableMessage(
    'The value {value} is not a valid email.',
    ['value' => 'test@example'],
));
```

Without ever touching a translator, `getMessage()` already returns the fully ICU-formatted English text — `TranslatableExceptionTrait` formats it eagerly at construction time. Handing the exception a real translator later (via `trans()`) is entirely optional, and is what actually produces a translated string in another locale.

```
try {
    // ...
} catch (ValidationException $e) {
```

```
    // Untranslated (eager ICU formatting only).
    echo $e->getMessage();

    // Translated, given a real translator.
    echo $e->trans($translator, 'es');
}
```

Translation Files

Files follow `symfony/translation`'s own naming convention, flat inside a directory: `{domain}(+intl-icu)?.{locale}.{format}`.

```
// translations/errors+intl-icu.en.php
return [
    'The value {value} is not a valid email.' => 'The value {value} is not a valid email.',
];
```

```
# translations/errors+intl-icu.es.yaml
'The value {value} is not a valid email.': 'El valor {value} no es un correo electrónico válido.'
```

The `+intl-icu` suffix in the domain activates ICU MessageFormat parsing for that file (`{value}`-style placeholders, plural/select patterns). See ICU Formatting for details.

Building a Translator and Registering Resources

```
use Derafu\Translation\TranslationResourceRegistrar;
use Derafu\Translation\TranslatorFactory;

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

// Discover and register every translation file in a directory.
$registrar = new TranslationResourceRegistrar($translator);
$registrar->registerDirectory(__DIR__ . '/translations');

echo $translator->trans('The value {value} is not a valid email.', [
    'value' => 'test@example',
], 'errors+intl-icu', 'es');
```

Key Benefits

1. **Zero Compromise**: Existing exception-handling code (`getMessage()`, `catch` blocks) keeps working unchanged.
2. **ICU Power**: Full ICU message formatting, with or without a translator.
3. **Built on `symfony/translation`**: No custom translation engine to maintain or learn — the real Symfony component, with all its loaders.
4. **Multi-Package Friendly**: Register directories from N sources (each package can ship its own translations) with a single registrar.

When to Use This Library

- You want translatable exceptions without rewriting how you throw or catch them.
- You want ICU message formatting without depending on a full framework.
- You want a small, standalone way to discover translation files across multiple directories.

Last updated on 09/09/2026