

Translation Bundle

Register Derafu translation resource providers with Symfony's native translator, preserving application overrides, locales and caching

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

Translation Bundle

github	derafu/symfony-translation-bundle	last commit	today	CI	passing	code size	23 KiB
open issues	0	downloads	1	downloads	1 this month		

derafu/symfony-translation-bundle connects the resource providers from [derafu/translation](#) to Symfony's native translator. Services tagged `derafu_translation.resource_provider` supply directories containing translation catalogues. The bundle adds those catalogues to the translator configured by FrameworkBundle.

The application continues to use `Symfony\Contracts\Translation\TranslatorInterface`, the `translator` service and Symfony's normal locale, fallback, logging and cache configuration.

Requirements

- PHP 8.5 or higher
- Symfony 8.1 components compatible with the package's Composer constraints
- `derafu/translation`, installed as a dependency

Installation

```
composer require derafu/symfony-translation-bundle
```

The package currently requires `derafu/translation:dev-main`. If the application's stability settings reject that dependency, allow development packages with stable releases preferred:

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

Register the bundle alongside FrameworkBundle in `config/bundles.php`:

```
return [
    Symfony\Bundle\FrameworkBundle\FrameworkBundle::class => ['all' => true],
    Derafu\TranslationBundle\TranslationBundle::class => ['all' => true],
];
```

Configure translation through FrameworkBundle:

```
# config/packages/translation.yaml
framework:
    default_locale: es
    translator:
        default_path: '%kernel.project_dir%/translations'
        fallbacks: ['en']
```

The bundle has the configuration alias `derafu_translation` and requires no options under that alias. Its services are loaded automatically from `resources/config/symfony-translation-bundle-services.yaml`.

Register Package Resources

Import the service configuration supplied by each package whose translations the application needs. For example, an application using [derafu/array](#) can import its provider:

```
# config/services.yaml
imports:
    - { resource: '../vendor/derafu/array/resources/config/array-services.yaml' }
```

This path assumes Composer's default `vendor/` directory. The imported YAML registers the package's tagged provider; the bundle consumes that tag when constructing Symfony's translator. Installing a package alone does not register its provider as a service.

The bundle registers Derafu Translation's own resource provider, including its exception messages. Do not import `derafu/translation/resources/config/translation-services.yaml` for this integration: that configuration defines the standalone translator alias, while a Symfony application uses FrameworkBundle's existing alias.

Add a Custom Provider

Implement `TranslationResourceProviderInterface` to supply one or more directories. Providers can receive normal constructor dependencies from the container.

```
// src/Translation/ApplicationResourceProvider.php
namespace App\Translation;
```

```
use Derafu\Translation\Contract\TranslationResourceProviderInterface;

final class ApplicationResourceProvider implements
TranslationResourceProviderInterface
{
    /** @return iterable<string> */
    public function getDirectories(): iterable
    {
        return [__DIR__ . '/../../resources/translations'];
    }
}
```

Register its tag explicitly:

```
# config/services.yaml
services:
    App\Translation\ApplicationResourceProvider:
        tags: ['derafu_translation.resource_provider']
```

For a fixed list of directories, `Derafu\Translation\SimpleTranslationResourceProvider` provides the same interface without a custom class:

```
services:
    app.translation_resources:
        class: Derafu\Translation\SimpleTranslationResourceProvider
        arguments:
            $directories: ['%kernel.project_dir%/resources/translations']
        tags: ['derafu_translation.resource_provider']
```

These are alternative ways to register the same directories; choose one.

Catalogue Files and Overrides

Provider directories contain files directly inside them, using the naming convention `{domain}.{locale}.{format}` or `{domain}+intl-icu.{locale}.{format}`. For example:

```
resources/translations/
    messages.en.yaml
    messages.es.yaml
    errors+intl-icu.es.php
```

Discovery and filename validation use `Derafu\Translation\TranslationResourceRegistrar`. Missing directories, malformed names and unsupported extensions retain its translatable exceptions. See [Derafu Translation](#) for its supported formats and message conventions.

For a repeated message in the same locale and catalogue domain, precedence is:

Resources	Priority
Tagged providers	Registration order; the last provider wins.
Directories returned by one provider	Returned order; the last directory wins.
Resources already configured by Symfony	Loaded after provider resources, preserving Symfony's own ordering.

The application's `translations/` directory can therefore override package messages. Other package messages remain available. For an ICU catalogue such as `errors+intl-icu.es.php`, place the override in the corresponding ICU catalogue.

For example, if a provider defines `greeting` in `messages.es.yaml`, override it in the application:

```
# translations/messages.es.yaml
greeting: 'Hello from the application'
```

Request it through the normal translator:

```
use Symfony\Contracts\Translation\TranslatorInterface;

final class GreetingService
{
    public function __construct(private readonly TranslatorInterface $translator)
    {
    }

    public function greet(): string
    {
        return $this->translator->trans('greeting', domain: 'messages');
    }
}
```

Locales and Cache

Symfony's configured default locale and fallback locales apply to provider messages. Locales supplied only by a provider are included in native translation cache warming, subject to the application's `framework.enabled_locales` restriction.

Provider files and directories participate in Symfony's normal catalogue freshness checks. In debug mode, catalogue freshness checks detect modified, added or deleted provider files between application requests. Production deployments should clear and warm the application cache through the normal Symfony commands:

```
php bin/console cache:clear  
php bin/console cache:warmup
```

Symfony's logging decorator remains compatible with the provider resources. The bundle preserves the native translator service and its contract alias.

Disable Translation

```
# config/packages/translation.yaml  
framework:  
    translator: false
```

With translation disabled, the bridge stays inactive. It does not enable translation, create a replacement translator or resolve providers for translation registration.

Last updated on 26/10/2026