

Symfony Integration

Wire file-backed catalogues, custom factories and translated exceptions with Symfony DependencyInjection

Anonymous · 20/10/2026

Symfony Integration

The repository package can be used directly or wired through Symfony DependencyInjection. The examples assume the application installs and configures that component.

Configure Data Sources

```
services:
    Derafu\Repository\Contract\RepositoryManagerInterface:
        class: Derafu\Repository\Service\RepositoryManager
        arguments:
            $dataProvider: '@Derafu\Repository\Service\DataProvider'

    Derafu\Repository\Service\DataProvider:
        arguments:
            $sources:
                App\Entity\Country:
                    '%kernel.project_dir%/resources/data/countries.php'
                App\Contract\DocumentTypeInterface:
                    '%kernel.project_dir%/resources/data/document_types.php'
            $config:
                normalization:
                    idAttribute: codigo
                    nameAttribute: glosa
```

Country and DocumentType are application entity classes. Request the repositories using the same identifiers used as source keys. The DocumentTypeInterface identifier follows the entity convention described in [entities and mapping](#). `%kernel.project_dir%` is supplied by the application environment.

An entity's `repositoryClass` attribute selects its specialized in-memory repository. No separate repository service is required for that construction path.

Add a Factory

```
services:
```

```
App\Repository\ProductRepositoryFactory:
  arguments:
    $connection: '@app.database_connection'

Derafu\Repository\Contract\RepositoryManagerInterface:
  class: Derafu\Repository\Service\RepositoryManager
  arguments:
    $dataProvider: '@Derafu\Repository\Service\DataProvider'
    $factories:
      App\Entity\Product: '@App\Repository\ProductRepositoryFactory'
```

This manager definition combines explicit factories with the data provider from the preceding example. `app.database_connection` is an application-defined service, and `ProductRepositoryFactory` implements the package's factory contract. The repository itself receives the dependencies its factory supplies.

For an application with only explicit factories, omit `$dataProvider`.

Translated Exception Messages

The optional service resource registers `RepositoryTranslationResourceProvider` with the `derafu_translation.resource_provider` tag:

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

This import registers translation resources; it does not configure the manager or the data sources. The application's translator must consume the tagged resource providers as described in the [translation documentation](#).

Without a service container, provide the resource provider directly:

```
use Derafu\Repository\Translation\RepositoryTranslationResourceProvider;
use Derafu\Translation\TranslatorFactory;

$translator = TranslatorFactory::create(
    'es',
    resourceProviders: [new RepositoryTranslationResourceProvider()],
);
```

Package exceptions expose `trans($translator)` for translated output. Their original messages are in English, and the package includes a Spanish catalogue.