Exceptions

Working with Translatable Exceptions

Anonymous · 09/09/2026

Working with Translatable Exceptions

This guide covers the translatable exception hierarchy shipped by `derafu/translation`.

A Note on Translation Ids

Every example in this guide uses **the real English message text as the translation id** (e.g. `'The field {field} is required.'`), not an abstract key like `validation.required`. This is deliberate: Symfony's own translation documentation recommends real text for shared libraries specifically (as opposed to end-user applications, where abstract keys make more sense), so that code stays readable and still produces a sane message even when no translator — or no matching translation — is available. `TranslatableExceptionTrait` is built around this idea: the message you throw with is always what `getMessage()` returns when untranslated.

Available Exceptions

The library mirrors PHP's own SPL exception hierarchy with translatable equivalents.

Core Exceptions

- `Derafu\Translation\Exception\Core\TranslatableException` — extends `Exception`.
- `Derafu\Translation\Exception\Core\TranslatableLogicException` — extends `LogicException`.
- `Derafu\Translation\Exception\Core\TranslatableRuntimeException` — extends `RuntimeException`.

Logic Exceptions

- `Derafu\Translation\Exception\Logic\TranslatableDomainException` — extends `DomainException`.
- `Derafu\Translation\Exception\Logic\TranslatableInvalidArgumentException` — extends `InvalidArgumentException`.
- `Derafu\Translation\Exception\Logic\TranslatableLengthException` — extends `LengthException`.
- `Derafu\Translation\Exception\Logic\TranslatableOutOfRangeException` — extends

`OutOfRangeException`.

Runtime Exceptions

- `Derafu\Translation\Exception\Runtime\TranslatableOutOfBoundsException` — extends `OutOfBoundsException`.
- `Derafu\Translation\Exception\Runtime\TranslatableOverflowException` — extends `OverflowException`.
- `Derafu\Translation\Exception\Runtime\TranslatableRangeException` — extends `RangeException`.
- `Derafu\Translation\Exception\Runtime\TranslatableUnderflowException` — extends `UnderflowException`.
- `Derafu\Translation\Exception\Runtime\TranslatableUnexpectedValueException` — extends `UnexpectedValueException`.

Every class above implements `TranslatableInterface` and uses `TranslatableExceptionTrait` internally, so they all share the same constructor and `trans()` method described below.

Using the Trait Directly

TranslatableExceptionTrait

If you can't (or don't want to) extend one of the exceptions above — for example, because you already extend something else — use the trait instead:

```
use Derafu\Translation\Contract\TranslatableInterface;
use Derafu\Translation\Trait\TranslatableExceptionTrait;
use DomainException;
use Throwable;

class OrderException extends DomainException implements TranslatableInterface
{
    use TranslatableExceptionTrait;

    // $defaultDomain is a *typed* property on the trait, so PHP does not
    // allow overriding its default value by simply redeclaring it here
    // (that's a fatal "incompatible property definition" error). Set it in
    // the constructor instead, before calling normalizeMessage().
    public function __construct(
        string|array|TranslatableInterface $message,
        int $code = 0,
        ?Throwable $previous = null,
    ) {
        $this->defaultDomain = 'orders';
        parent::__construct($this->normalizeMessage($message), $code, $previous);
    }
```

```
    public static function insufficientStock(string $product): self
    {
        return new self(['Not enough stock for "{product}".', 'product' => $product]);
    }
}
```

When to Use the Trait vs. Extending a Base Exception

Use the trait when:

- You already extend another exception class.
- You need custom behavior beyond translation.

Use one of the base exceptions when:

- You don’t need custom behavior.
- You want the simplest possible implementation:

```
class ValidationException extends TranslatableDomainException
{
    // Inherits everything from the trait, with no extra code.
}
```

The Constructor

All translatable exceptions accept the same three argument shapes:

```
public function __construct(
    string|array|TranslatableInterface $message,
    int $code = 0,
    ?Throwable $previous = null,
)
```

- **`string`**: used as both the exception message and the translation id.
- **`array`**: the first element is the message/id, the remaining elements are named ICU parameters (`['The field {field} is required.', 'field' => 'email']`).
- **`TranslatableInterface`**: a `TranslatableMessage` (or any other implementation) built directly — useful when you want to reuse the same translatable message in more than one place.

Default Domain and Locale

`TranslatableExceptionTrait` defines:

```
protected string $defaultDomain = 'errors';
protected string $defaultLocale = 'en';
```

Override `$defaultDomain` in your own exception class to group its messages under a different translation domain. This works fine through normal class inheritance, when you extend one of the built-in exceptions (they already apply the trait for you):

```
class OrderException extends TranslatableDomainException
{
    protected string $defaultDomain = 'orders';
}
```

If you `use TranslatableExceptionTrait` directly in your own class instead (see above), redeclaring `$defaultDomain` in that *same* class fails with a fatal "incompatible property definition" error — PHP does not allow a class to override the default value of a trait's own *typed* property by redeclaring it in the class that uses the trait. Set it in the constructor instead in that case, as shown in the `OrderException` example above.

Best Practices

1. **Choose the most specific exception type available.** It carries semantic meaning for callers doing `catch (SomeSplException $e)`, even before translation comes into play.

2. **Use the real message text as the id**, per the note above — don't introduce abstract keys unless you have a specific reason to (e.g. a dedicated translation-management workflow that needs stable ids independent of wording).

3. **Provide every ICU parameter the message references.** A missing parameter is left unexpanded in the output rather than throwing.

4. **Group related exceptions under a domain-specific base class:**

   ```
   abstract class OrderException extends TranslatableDomainException
   {
       protected string $defaultDomain = 'orders';
   }
   ```

```
class OrderNotFoundException extends OrderException {}
class OrderValidationException extends OrderException {}
```

Last updated on 09/09/2026