Translations

Translations

Anonymous · 09/09/2026

Translations

`derafu/form` has two independent, optional translation surfaces. Both are powered by `derafu/translation`, and both do nothing unless you explicitly wire a translator — without one, everything renders in English exactly as before.

1. Validation Error Messages

`FormDataProcessor` runs field values through `derafu/data-processor` rules. When a rule fails, it throws a `Derafu\DataProcessor\Exception\ValidationException` (or one of the other translatable exceptions from that package — see its Translations guide). `FormDataProcessor` accepts an optional translator and locale; when given, it translates each error before adding it to the result:

```
final class FormDataProcessor implements FormDataProcessorInterface
{
    public function __construct(
        FormRulesResolverInterface $resolver,
        ProcessorInterface $processor,
        UiSchemaRuleEvaluatorInterface $evaluator = new UiSchemaRuleEvaluator(),
        ?TranslatorInterface $translator = null,
        ?string $locale = null,
    ) {
    }
}
```

Without a translator, `getErrors()` returns the untranslated (English) messages — the exact same behavior as before this parameter existed.

2. Input Action Labels

Controls rendered with the `Control` UI schema type can declare `options.actions` — input-group buttons like “show/hide password”, “generate a random password”, or “copy the field’s value”. Their labels (and the default copy confirmation message) are resolved by `InputActionResolver`, which also accepts an optional translator:

```
final class InputActionResolver
{
    public function __construct(
        ?TranslatorInterface $translator = null,
        ?string $locale = null,
    ) {
    }
}
```

An explicit `label` or `message` given directly in `options.actions` is always used as-is and never translated — only the three built-in defaults are:

```
[
    'type' => 'Control',
    'scope' => '#/properties/password',
    'options' => [
        'actions' => ['toggle-password'],
    ],
]
```

`InputActionResolver` isn't wired into the rendering pipeline directly — `ElementRendererProvider` (used internally by `FormRendererFactory`) forwards it to `ControlRenderer`. To use a translator-aware resolver, pass your own `ElementRendererProvider` via the `element_renderers` factory option (see Activating It below).

What Ships

`derafu/form` includes:

- `resources/translations/form+intl-icu.es.php` — a Spanish translation for the four strings `InputActionResolver` can produce (validation error messages come from `derafu/data-processor`'s own translations instead — `derafu/form` doesn't duplicate them).
- `Derafu\Form\Translation\FormTranslationResourceProvider` — a `TranslationResourceProviderInterface` implementation pointing at that directory.

Neither does anything on its own — `derafu/form` is a library, it doesn't build or own a `Translator`.

Activating It (Plain PHP)

```
use Derafu\DataProcessor\ProcessorFactory;
use Derafu\DataProcessor\Translation\DataProcessorTranslationResourceProvider;
use Derafu\Form\Processor\FormDataProcessor;
use Derafu\Form\Processor\FormRulesResolver;
use Derafu\Form\Translation\FormTranslationResourceProvider;
use Derafu\Translation\TranslatorFactory;
```

```
$translator = TranslatorFactory::create(
    defaultLocale: 'es',
    fallbackLocales: ['es', 'en'],
    resourceProviders: [
        new DataProcessorTranslationResourceProvider(), // Validation error messages.
        new FormTranslationResourceProvider(),           // Input action labels.
    ],
);

$processor = new FormDataProcessor(
    new FormRulesResolver(),
    ProcessorFactory::create(),
    translator: $translator,
    locale: 'es',
);
```

For the action labels, build the renderer with a translator-aware `ElementRendererProvider`:

```
use Derafu\Form\Factory\FormRendererFactory;
use Derafu\Form\Renderer\ElementRendererProvider;
use Derafu\Form\Renderer\Support\InputActionResolver;

$formRenderer = FormRendererFactory::create([
    'element_renderers' => new ElementRendererProvider(
        actionResolver: new InputActionResolver($translator, 'es'),
    ),
]);
```

Activating It (Dependency Injection)

If your application uses `symfony/dependency-injection`, import all three recipe files. `form-services.yaml` already registers `InputActionResolver` and `ElementRendererProvider` as autowired services, and wires `FormRendererInterface`'s `element_renderers` option to that `ElementRendererProvider` — nothing extra to configure beyond registering your own `Translator`:

```
imports:
    - { resource: '../vendor/derafu/translation/resources/config/translation-
services.yaml' }
    - { resource: '../vendor/derafu/data-processor/resources/config/data-processor-
services.yaml' }
    - { resource: '../vendor/derafu/form/resources/config/form-services.yaml' }

services:
    Symfony\Component\Translation\Translator:
        factory: ['Derafu\Translation\TranslatorFactory', 'create']
        arguments:
            $defaultLocale: 'es'
```

```
            $fallbackLocales: ['es', 'en']
            $resourceProviders: !tagged_iterator derafu_translation.resource_provider
```

Both `DataProcessorTranslationResourceProvider` and `FormTranslationResourceProvider` are already tagged `derafu_translation.resource_provider` in their respective recipe files, so they're picked up automatically. `FormDataProcessor` and `InputActionResolver` both autowire `?TranslatorInterface`, so they resolve to the `Translator` above via the alias `translation-services.yaml` provides — no explicit argument wiring needed for either of them.

Last updated on 09/09/2026