Docs

Data Processor Project

Derafu Data Processor

Anonymous · 09/11/2026 · #php

Table of Contents

Data Processor Project 3
- ***Introduction*** 4
- ***Rules Reference*** 7
- ***Custom Registries*** 16
- ***Custom Rules*** 17
- ***Custom Registrar*** 19
- ***Custom Parser*** 21
- ***Translations*** 23

Data Processor Project

Derafu Data Processor

Anonymous · 09/11/2026 · #php

Derafu Data Processor

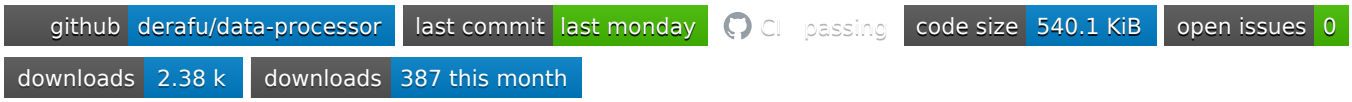
Last updated on 09/11/2026

Introduction

PHP library that processes data in four phases: casting, transformation, sanitization and validation, with custom rules

Anonymous · 09/11/2026

Four-Phase Data Processing Library



A PHP library designed to process data through four distinct phases: casting, transformation, sanitization and validation.

What Makes it Different?

Unlike traditional validation libraries that focus solely on validation, Derafu Data Processor offers:

- **Clear Phase Separation**: Each processing phase (cast → transform → sanitize → validate) has its own purpose and runs in a specific order.
- **Independent Transformations**: A dedicated transformation layer for complex data conversions, separate from basic type casting.
- **Type-Safe Operations**: Strong typing and clear error handling in each phase.
- **Extensible Rule System**: Each type of rule (caster, transformer, sanitizer, validator) follows its own interface.
- **Minimal Dependencies**: `intl` extension for specific format rules, `Carbon` for dates, and `derafu/translation` for translatable error messages (see Translations).

Features

- **Type-Safe Casting**: Convert between data types safely.
    - Basic types (string, int, float, bool), strict: a value that is not clearly a boolean or an integer is an error, it is not guessed.
    - Date and time handling, with absolute ISO 8601 dates (a relative text like `tomorrow` is an error).
- **Data Transformations**: Complex data conversions.
    - Base64 encoding/decoding.

    - JSON processing.
    - Slug generation.
    - Character transliteration.

- **Rich Sanitization Rules**: Clean and normalize input data.

    - String sanitization (trim, strip_tags, substring, etc.).
    - Regex-based cleaning (regex_remove, regex_keep).
    - HTML entity handling (htmlspecialchars, htmlentities, etc.).
    - Character set manipulation (remove_non_printable, remove_chars, etc.).

- **Comprehensive Validation**: Various validation rules.

    - String validations (required, min_length, max_length, etc.).
    - Number validations (range, gte, lte, etc.).
    - Date validations (after, before, between, etc.).
    - Format validations (email, URL, UUID, IBAN, hostname, host, etc.).
    - Values of HTML inputs (color, month, week, time, datetime_local).
    - File validations (file, image, mimetype).
    - Array validations (min_items, max_items, contains, allowed_keys, dependent_required, unique, unique_loose, etc.).

- **Optional Fields**: A `null`, `''` or `[]` value is `null` without running the rules, unless the field is `required`.

Installation

```
composer require derafu/data-processor
```

Basic Usage

```
use Derafu\DataProcessor\Processor;
use Derafu\DataProcessor\Registrar\DefaultRuleRegistrar;
use Derafu\DataProcessor\RuleParser;
use Derafu\DataProcessor\RuleRegistry;
use Derafu\DataProcessor\RuleResolver;

$registry = new RuleRegistry();
(new DefaultRuleRegistrar())->register($registry);
$processor = new Processor(new RuleResolver($registry), new RuleParser());

// Simple validation.
$result = $processor->process('test@example.com', [
    'validate' => ['email'],
```

```
]);

// Multi-phase processing.
$result = $processor->process(' TEST@EXAMPLE.COM ', [
    'transform' => ['lowercase'],
    'sanitize' => ['trim'],
    'validate' => ['email', 'max_length:255'],
]);

// Array validation.
$result = $processor->process([1, 2, 2, 3], [
    'validate' => ['unique', 'max_items:5'],
]);

// File validation.
$result = $processor->process($_FILES['upload'], [
    'validate' => ['file:2M', 'mimetype:application/pdf,image/*']
]);
```

Dependency Injection

Applications using Symfony DependencyInjection can import the package configuration:

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

Inject `Derafu\DataProcessor\Contract\ProcessorInterface` into your application services. The configured processor uses the container's `RuleResolverInterface` and `RuleParserInterface`. The resolver uses the same `RuleRegistryInterface` registered in the container.

The registry is initialized by `RuleRegistrarInterface`, implemented by `CompositeRuleRegistrar`. It collects services tagged with `derafu_data_processor.rule_registrar`. `DefaultRuleRegistrar` has priority `1000`; additional registrars normally use priority `0` and run afterward, allowing them to add rules or replace existing names. Higher priorities run first; among equal priorities, registration order is preserved. See Custom Registrar and Custom Parser.

The YAML supplies the processor and its rules. Configuring an application translator is a separate choice; see Translations.

Last updated on 09/11/2026

Rules Reference

Rules Reference

Anonymous · 09/11/2026

Rules Reference

Cast Rules

Rule	Description	Example
`boolean`, `bool`	Convert value to boolean. Only clear values are accepted: `true`, `1`, `yes`, `on` and `false`, `0`, `no`, `off`, `''`, `null` (text without distinguishing case); any other value is an error	`'cast' => 'boolean'`
`date`	Convert an absolute ISO 8601 date to a Carbon date (see Absolute dates)	`'cast' => 'date'`
`datetime`	Convert an absolute ISO 8601 date and time to a Carbon datetime (see Absolute dates)	`'cast' => 'datetime'`
`float`, `decimal`	Convert value to float. A float does not keep trailing zeros (`'10.50'` is `10.5`), so validate its decimals with `decimal:0,N` (up to `N` decimals), not with `decimal:N`	`'cast' => 'float'`
`integer`, `int`	Convert value to integer. A number with a decimal part (`'1.9'`) or out of range is an error, it is not truncated; `'1.0'` and `'1e2'` are `1` and `100`	`'cast' => 'integer'`
`string`, `str`	Convert value to string	`'cast' => 'string'`
`timestamp`	Convert value to Unix timestamp. A number is used as it is; a text must be an absolute ISO 8601 date (see Absolute dates)	`'cast' => 'timestamp'`

Transform Rules

Rule	Description	Example
`base64_decode`	Decode base64 string	`'transform' => ['base64_decode']`
`base64_encode`	Encode value to base64	`'transform' => ['base64_encode']`
`json_decode`	Decode JSON string	`'transform' => ['json_decode']`, `'transform' => ['json_decode:array']`
`json_encode`	Encode value to JSON	`'transform' => ['json_encode']`, `'transform' => ['json_encode:pretty']`

Rule	Description	Example
`json_to_array`	Convert JSON string to array	`'transform' => ['json_to_array']`
`json_to_object`	Convert JSON string to object	`'transform' => ['json_to_object']`
`round`	Round numeric value	`'transform' => ['round:2']`
`lowercase`, `lower`	Convert string to lowercase	`'transform' => ['lowercase']`
`slug`	Convert string to URL friendly format	`'transform' => ['slug']`
`transliterate`	Convert characters to ASCII	`'transform' => ['transliterate']`
`uppercase`, `upper`	Convert string to uppercase	`'transform' => ['uppercase']`

Sanitize Rules

Rule	Description	Example
`addslashes`	Add slashes before quotes	`'sanitize' => ['addslashes']`
`htmlentities`	Convert characters to HTML entities	`'sanitize' => ['htmlentities']`
`htmlspecialchars`	Convert special characters to HTML entities	`'sanitize' => ['htmlspecialchars']`
`regex_keep`	Keep only characters matching pattern	`'sanitize' => ['regex_keep:/[0-9]/']`
`regex_remove`	Remove characters matching pattern	`'sanitize' => ['regex_remove:/[^a-z]/']`
`remove_chars`	Remove specific characters	`'sanitize' => ['remove_chars:@#$']`
`remove_non_printable`	Remove ASCII control characters (except tab and line feed), C1 control characters and invisible format characters (`U+200B`, `U+200E`, `U+202A` to `U+202E`, `U+2060`, `U+FEFF`, `U+00AD`). `U+200C` and `U+200D` are kept, because some scripts and emoji sequences need them	`'sanitize' => ['remove_non_printable']`
`remove_prefix`	Remove string prefix	`'sanitize' => ['remove_prefix:http']`
`remove_suffix`	Remove string suffix	`'sanitize' => ['remove_suffix:.txt']`
`spaces`	Normalize multiple spaces to single	`'sanitize' => ['spaces']`

Rule	Description	Example
`strip_tags`	Remove HTML and PHP tags	`'sanitize' => ['strip_tags'],` `'sanitize' => ['strip_tags:<p> ']`
`substring`	Extract part of string	`'sanitize' => ['substring:5'],` `'sanitize' => ['substring:0,10']`
`trim`	Remove ASCII whitespace from ends (not the no-break space `U+00A0`)	`'sanitize' => ['trim']`

Validate Rules

Array Validation

Rule	Description	Example
`allowed_keys`	Check that an array has only the listed keys (`additionalProperties: false` of a JSON Schema object)	`'validate' => ['allowed_keys:name,email']`
`contains`	Check that the array has at least one of the listed values	`'validate' => ['contains:terms'],` `'validate' => ['contains:a,b']`
`dependent_required`	If the first key has a value, the other keys must have a value too	`'validate' => ['dependent_required:card,address,zip']`
`in`, `choices`	Check if value is in list. Value and list are compared as text (`2`, `2.0` and `'2'` are the same choice for `in:1,2,3`); an array is valid when each element is in the list; booleans, `null` and objects are never in the list	`'validate' => ['in:a,b,c']`, `'validate' => ['in:1,2,3']`
`max_contains`	Check that the array has at most N elements that are one of the listed values	`'validate' => ['max_contains:1,a,b']`
`max_items`	Check array maximum length. In an associative array, it counts the keys	`'validate' => ['max_items:5']`
`min_contains`	Check that the array has at least N elements that are one of the listed values	`'validate' => ['min_contains:2,a,b']`

Rule	Description	Example
`min_items`	Check array minimum length. In an associative array, it counts the keys	`'validate' => ['min_items:1']`
`notempty`	Check if array is not empty	`'validate' => ['notempty']`
`unique`	Check if array values are unique. The comparison is strict (same type and value): `['1e1', '10']` and `['1', 1]` are unique	`'validate' => ['unique']`
`unique_loose`	Check if array values are unique with a loose comparison (==): `['1e1', '10']` and `['1', 1]` are duplicates	`'validate' => ['unique_loose']`

Date Validation

Rule	Description	Example
`after`	Check if date is after another	`'validate' => ['after:2024-01-01']`
`after_or_equal`	Check if date is after or equal	`'validate' => ['after_or_equal:2024-01-01']`
`before`	Check if date is before another	`'validate' => ['before:2024-12-31']`
`before_or_equal`	Check if date is before or equal	`'validate' => ['before_or_equal:2024-12-31']`
`between`	Check if date is between two dates	`'validate' => ['between:2024-01-01,2024-12-31']`
`date_equals`	Check if date equals another	`'validate' => ['date_equals:2024-01-01']`
`date_format`	Check if date matches format	`'validate' => ['date_format:Y-m-d']`
`datetime_local`	Check a date and time as `<input type="datetime-local">` sends it: `YYYY-MM-DDTHH:MM` or `YYYY-MM-DDTHH:MM:SS`, with an existing date	`'validate' => ['datetime_local']`
`month`	Check a month as `<input type="month">` sends it: `YYYY-MM`	`'validate' => ['month']`
`time`	Check a time as `<input type="time">` sends it: `HH:MM` or `HH:MM:SS`	`'validate' => ['time']`

Rule	Description	Example
`week`	Check a week as `<input type="week">` sends it: `YYYY-Www`, with the week 53 only in the years that have it	`'validate' => ['week']`
`weekday`	Check if date is weekday	`'validate' => ['weekday']`
`weekend`	Check if date is weekend	`'validate' => ['weekend']`

File Validation

Rule	Description	Example
`file`	Validate file with size limits (`max_size`, and `min_size`, 1 byte by default)	`'validate' => ['file:2M']`, `'validate' => ['file:max_size=2M,min_size=0']`
`image`	Validate image file (JPEG, PNG, GIF or WebP) with size limits (`max_size`, and `min_size`, 1 byte by default)	`'validate' => ['image:2M']`, `'validate' => ['image:max_size=2M,min_size=0']`
`mimetype`	Validate file mime type, detected from its content. A whole type allows any of its subtypes with `type/*`; any other `*` is text and never matches	`'validate' => ['mimetype:application/pdf,image/jpeg']`, `'validate' => ['mimetype:image/*']`, `'validate' => ['mimetype:application/pdf,application/x-empty']`

Financial Validation

Rule	Description	Example
`bic`	Validate BIC/SWIFT code (it identifies a bank, not an account)	`'validate' => ['bic']`
`card_number`	Validate credit card number	`'validate' => ['card_number']`
`iban`	Validate IBAN: country, length of that country, and check digits. Spaces and lowercase are accepted. Only the countries of the SWIFT IBAN Registry have IBAN (in Latin America: Brazil, Costa Rica, Dominican Republic, Guatemala, Honduras, Nicaragua and El Salvador), any other country is an error	`'validate' => ['iban']`

I18n Validation

Rule	Description	Example
`country`	Validate country code	`'validate' => ['country']`
`currency`	Validate ISO 4217 currency code, in uppercase or lowercase. It has the codes in force, with the funds codes (`CLF`), and not the precious metals and special codes (`XAU`, `XAG`, `XDR`, `XXX`, etc.)	`'validate' => ['currency']`
`language`	Validate language code (2 or 3 letters), with an optional country: `es`, `fil`, `es_CL` or `es-CL`	`'validate' => ['language']`
`locale`	Validate locale code, with `_` or `-` as separator: `es_CL` or `es-CL`. The list is the one of ICU, so it depends on the version of ICU	`'validate' => ['locale']`
`timezone`	Validate timezone identifier	`'validate' => ['timezone']`

ID Validation

Rule	Description	Example
`uuid`	Validate UUID string (RFC 9562): versions 1 to 8, and the nil and max UUIDs. With a version, only that version is valid	`'validate' => ['uuid']`, `'validate' => ['uuid:4']`

Internet Validation

Rule	Description	Example
`email`	Validate email address. The domain can have international characters (`jose@exämple.com`); the part before `@` must be ASCII	`'validate' => ['email']`
`host`	Validate a hostname or an IP address. `host:v4` and `host:v6` limit the version of the IP address; a hostname is valid anyway	`'validate' => ['host']`, `'validate' => ['host:v4']`
`hostname`	Validate hostname. The last label can not be only digits, so `1.2.3.4` and `example.123` are not hostnames (RFC 1123)	`'validate' => ['hostname']`
`ip`	Validate IP address	`'validate' => ['ip']`, `'validate' => ['ip:v4']`, `'validate' => ['ip:v6']`
`url`	Validate URL. With parameters, only those schemes are valid. A host with international characters is accepted	`'validate' => ['url']`, `'validate' => ['url:http,https']`

Numeric Validation

Rule	Description	Example
`decimal`, `float`, `real`	Validate decimal number with exactly `N` decimals (`decimal:2`), or between `min` and `max` decimals (`decimal:0,2`)	`'validate' => ['decimal:2'],` `'validate' => ['decimal:0,2']`
`digits_range`	Validate that the value is only digits, with a number of digits between `min` and `max`	`'validate' => ['digits_range:3,5']`
`digits`	Validate that the value is only digits, with exactly that number of digits	`'validate' => ['digits:4']`
`gt`	Greater than	`'validate' => ['gt:10']`
`gte`, `min`	Greater than or equal	`'validate' => ['gte:10']`
`integer`, `int`	Validate integer	`'validate' => ['integer']`
`lt`	Less than	`'validate' => ['lt:10']`
`lte`, `max`	Less than or equal	`'validate' => ['lte:10']`
`multiple_of`	Check if number is multiple	`'validate' => ['multiple_of:5']`
`numeric`	Validate numeric value	`'validate' => ['numeric']`
`range`	Validate number in range	`'validate' => ['range:1,100']`

String Validation

Rule	Description	Example
`alpha`	Only alphabetic characters	`'validate' => ['alpha']`
`alpha_dash`	Alphanumeric with dashes/underscores	`'validate' => ['alpha_dash']`
`alpha_num`	Only alphanumeric characters	`'validate' => ['alpha_num']`
`base64`	Validate base64 string	`'validate' => ['base64']`
`color`	Check a color as `<input type="color">` sends it: `#rrggbb`, in uppercase or lowercase (`#fff` and `red` are not valid)	`'validate' => ['color']`
`ends_with`	String ends with value	`'validate' => ['ends_with:Test']`
`json`	Validate JSON string	`'validate' => ['json']`
`length`	Exact string length	`'validate' => ['length:10']`
`max_length`	Maximum string length	`'validate' => ['max_length:255']`

Rule	Description	Example
`min_length`	Minimum string length	`'validate' => ['min_length:3']`
`not_regex`	Not match regular expression	`'validate' => ['not_regex:/[0-9]/']`
`no_whitespace`	No whitespace in string	`'validate' => ['no_whitespace']`
`regex`	Match regular expression	`'validate' => ['regex:/^[A-Z]+$/']`
`required`	Value is required (`null`, `''` and `[]` fail)	`'validate' => ['required']`
`slug`	Validate URL friendly string	`'validate' => ['slug']`
`starts_with`	String starts with value	`'validate' => ['starts_with:Test']`

Notes

Rule parameters

The parameters of a rule go after `:`, separated by commas. A comma that is part of a parameter is written as `\,`: `'in:a\,b,c'` has two choices, `a,b` and `c`. A parameter that is not valid is an error, it is not replaced by another value (`'uuid:9'` fails with `Invalid parameter: 9`).

The rules that take a literal (`regex`, `not_regex`, `starts_with`, `ends_with`, `date_format`, `after`, `before`, `trim`, `remove_chars`, `strip_tags`, `transliterate`, etc.) receive all the text after `:` as one parameter, and `\,` has no special meaning for them.

Empty values

A value that is `null`, `''` or `[]` has nothing to cast, transform or sanitize. If the rules do not have `required`, the result is `null` and the validators do not run, so an optional field sent empty is valid. If the rules have `required`, the validators run and the value fails with `Field is required.`

Absolute dates

The date validators (`after`, `before`, `between`, `date_equals`, `weekday`, `weekend`, etc.) and the casters `date`, `datetime` and `timestamp` accept as text only an absolute date in ISO 8601: `YYYY-MM-DD`, or with time, `YYYY-MM-DDTHH:MM[:SS[.fraction]]` (a space can replace the `T`), with `Z` or an offset (`+03:00`) at the end. The date and the time must exist. Date objects are accepted too.

A relative text (`tomorrow`, `+1 week`, `now`) or another format (`01/02/2024`, which can be read as January 2 or February 1) is an error, because its value depends on the clock of the server or on how it is read. `date_format` validates a text in another format; to convert it, read it with your own code before processing. The parameter that a rule compares with (`after:tomorrow`) can still be relative.

Arrays and objects

The array validators also work with an associative array (an object): `min_items` and `max_items` count its keys.

`contains`, `min_contains` and `max_contains` compare the elements with the listed values as text, like `in`: `2`, `2.0` and `'2'` are the same value. Booleans, `null` and arrays are never equal to a value. In `min_contains` and `max_contains` the first parameter is the number `N`, and the rest are the values: `'min_contains:2,a,b'` needs at least two elements that are `a` or `b`. `N` must be an integer of 0 or more.

`allowed_keys` compares the keys of the array as text. `'allowed_keys:a,b'` fails with `Properties not allowed: c.` for `['a' => 1, 'c' => 2]`.

In `dependent_required` the first parameter is the key that triggers the rule, and the rest are the keys that become required. A key has a value when it exists and is not `null`, `''` or `[]` (`false` and `0` are values). If the first key has no value, the rule passes. `'dependent_required:card,address'` fails with `Properties required when card is present: address.` for `['card' => '4111', 'address' => '']`.

A value that is not an array fails with `Value must be an array.`

Last updated on 09/11/2026

Custom Registries

Creating Custom Registries

Anonymous · 09/11/2026

Creating Custom Registries

You can create your own rule registry to customize which rules are available:

```
use Derafu\DataProcessor\RuleRegistry;
use Derafu\DataProcessor\Processor;
use Derafu\DataProcessor\RuleResolver;

// Create a custom registry.
$registry = new RuleRegistry();

// Register only the rules you need.
$registry
    ->addTransformerRule('base64_encode', Base64EncodeRule::class)
    ->addSanitizerRule('trim', TrimRule::class)
    ->addCasterRule('integer', IntegerRule::class)
    ->addValidatorRule('email', EmailRule::class);

// Create resolver, parser and processor with your registry.
$resolver = new RuleResolver($registry);
$parser = new RuleParser();
$processor = new Processor($resolver, $parser);
```

Last updated on 09/11/2026

Custom Rules

Creating Custom Rules

Anonymous · 09/11/2026

Creating Custom Rules

Custom Cast Rule

```
use Derafu\DataProcessor\Contract\CasterRuleInterface;

final class CustomCastRule implements CasterRuleInterface
{
    public function cast(mixed $value, array $parameters = []): mixed
    {
        // Your casting logic here.
    }
}

// Register the rule.
$registry->addCasterRule('custom_cast', CustomCastRule::class);
```

Custom Transform Rule

```
use Derafu\DataProcessor\Contract\TransformerRuleInterface;

final class CustomTransformRule implements TransformerRuleInterface
{
    public function transform(mixed $value, array $parameters = []): mixed
    {
        // Your transformation logic here.
    }
}

// Register the rule.
$registry->addTransformerRule('custom_transform', CustomTransformRule::class);
```

Custom Sanitizer Rule

```
use Derafu\DataProcessor\Contract\SanitizerRuleInterface;
```

```
final class CustomSanitizerRule implements SanitizerRuleInterface
{
    public function sanitize(mixed $value, array $parameters = []): mixed
    {
        // Your sanitization logic here.
    }
}

// Register the rule.
$registry->addSanitizerRule('custom_sanitize', CustomSanitizerRule::class);
```

Custom Validator Rule

```
use Derafu\DataProcessor\Contract\ValidatorRuleInterface;

final class CustomValidatorRule implements ValidatorRuleInterface
{
    public function validate(mixed $value, array $parameters = []): void
    {
        // Your validation logic here.
    }
}

// Register the rule.
$registry->addValidatorRule('custom_validate', CustomValidatorRule::class);
```

Using Custom Rules

Once registered, you can use your custom rules just like built-in ones:

```
$processor->process('field', $value, [
    'cast' => 'custom_cast',
    'transform' => ['custom_transform'],
    'sanitize' => ['custom_sanitize'],
    'validate' => ['custom_validate'],
]);
```

Last updated on 09/11/2026

Custom Registrar

Creating a Custom Rule Registrar

Anonymous · 09/11/2026

Creating a Custom Rule Registrar

A registrar groups rule registrations. It can add caster, transformer, sanitizer and validator rules, including application-specific names for existing rule classes. For example:

```
namespace App\DataProcessor;

use Derafu\DataProcessor\Contract\RuleRegistrarInterface;
use Derafu\DataProcessor\Contract\RuleRegistryInterface;
use Derafu\DataProcessor\Rule\Transformer\String\UppercaseRule;
use Derafu\DataProcessor\Rule\Validator\String\MinLengthRule;

final class CustomRuleRegistrar implements RuleRegistrarInterface
{
    public function register(RuleRegistryInterface $registry): void
    {
        $registry->addTransformerRule('capital_letters', UppercaseRule::class);
        $registry->addValidatorRule('display_name', MinLengthRule::class);
    }
}
```

Dependency Injection

Import the package configuration and tag the registrar. Each member of the collection has its own service ID; the composite remains registered under `RuleRegistrarInterface`.

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

services:
    App\DataProcessor\CustomRuleRegistrar:
        tags: ['derafu_data_processor.rule_registrar']
```

The processor injected through `ProcessorInterface` can use both default and additional rules:

```
$result = $processor->process('alice', [
```

```
    'transform' => ['capital_letters'],
    'validate' => ['required', 'display_name:3'],
]);
// 'ALICE'
```

`DefaultRuleRegistrar` has tag priority `1000`; additional registrars use priority `0` unless configured otherwise. Higher priorities run first. A later registration of the same name in the same category replaces the earlier rule. This lets a tagged registrar override a default without discarding unrelated defaults.

Explicit Construction

Without a container, initialize a registry with the registrars you choose:

```
use App\DataProcessor\CustomRuleRegistrar;
use Derafu\DataProcessor\Processor;
use Derafu\DataProcessor\Registrar\CompositeRuleRegistrar;
use Derafu\DataProcessor\Registrar\DefaultRuleRegistrar;
use Derafu\DataProcessor\RuleParser;
use Derafu\DataProcessor\RuleRegistry;
use Derafu\DataProcessor\RuleResolver;

$registry = new RuleRegistry();
(new CompositeRuleRegistrar([
    new DefaultRuleRegistrar(),
    new CustomRuleRegistrar(),
]))->register($registry);

$processor = new Processor(new RuleResolver($registry), new RuleParser());
```

To use only custom rules, omit `DefaultRuleRegistrar` from that explicit list. The package YAML includes defaults automatically.

Last updated on 09/11/2026

Custom Parser

Creating a Custom Rule Parser

Anonymous · 09/11/2026

Creating a Custom Rule Parser

Implement `RuleParserInterface` to support application-specific rule syntax. This parser adds a `contact_email` shorthand and delegates other strings and arrays to the built-in parser:

```
namespace App\DataProcessor;

use Derafu\DataProcessor\Contract\RuleParserInterface;
use Derafu\DataProcessor\RuleParser;

final class ContactRuleParser implements RuleParserInterface
{
    public function parse(string|array $rules): array
    {
        return (new RuleParser())->parse(
            $rules === 'contact_email'
                ? 't(lowercase) s(trim) required|email'
                : $rules
        );
    }
}
```

Dependency Injection

Import the package YAML, then replace the parser under its interface ID:

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

services:
    Derafu\DataProcessor\Contract\RuleParserInterface:
        class: App\DataProcessor\ContactRuleParser
```

The configured `ProcessorInterface` receives this parser. In a service that receives that processor, the shorthand works directly:

```
$result = $processor->process(' TEST@EXAMPLE.COM ', 'contact_email');
// 'test@example.com'
```

The parser returns categories (`cast`, `transform`, `sanitize`, `validate`) with the same structure expected by the processor. It changes how rules are expressed, not the order of the processing phases.

Explicit Construction

Without a container, supply the parser through the processor constructor:

```
use App\DataProcessor\ContactRuleParser;
use Derafu\DataProcessor\Processor;
use Derafu\DataProcessor\Registrar\DefaultRuleRegistrar;
use Derafu\DataProcessor\RuleRegistry;
use Derafu\DataProcessor\RuleResolver;

$registry = new RuleRegistry();
(new DefaultRuleRegistrar())->register($registry);
$processor = new Processor(new RuleResolver($registry), new ContactRuleParser());

$result = $processor->process(' TEST@EXAMPLE.COM ', 'contact_email');
// 'test@example.com'
```

Last updated on 09/11/2026

Translations

Translations

Anonymous · 09/11/2026

Translations

`derafu/data-processor`'s exceptions can be translated. This page explains what's translatable, what ships out of the box, and how to activate it in your application.

What's Translatable

Four exception classes implement `Derafu\Translation\Contract\TranslatableInterface` (via `TranslatableExceptionTrait`, from `derafu/translation`):

- `Derafu\DataProcessor\Exception\ValidationException`
- `Derafu\DataProcessor\Exception\SanitizationException`
- `Derafu\DataProcessor\Exception\TransformationException`
- `Derafu\DataProcessor\Exception\CastingException`

`Derafu\DataProcessor\Exception\RuleNotFoundException` is **not** translatable on purpose — it signals a misconfigured rule name (a programming/configuration error), not something an end user should ever see.

All four use the `errors` domain (`TranslatableExceptionTrait`'s default). Every message thrown by a built-in rule is written in ICU MessageFormat syntax and uses the literal English text as its translation id — for example, `'Value must be greater than {value}.'` — following the same convention documented in derafu/translation's Exceptions guide.

Without any translator configured, `getMessage()` already returns the fully formatted English text — this library works exactly as before, whether or not you set up translation at all:

```
use Derafu\DataProcessor\Exception\ValidationException;

$e = new ValidationException(['Array must not contain more than {max} items.', 'max'
=> 5]);

echo $e->getMessage();
// "Array must not contain more than 5 items."
```

What Ships

`derafu/data-processor` includes:

- `resources/translations/errors+intl-icu.es.php` — a Spanish translation for every distinct message used by the four exception classes above (the `+intl-icu` domain suffix is required for `derafu/translation`'s ICU placeholders like `{max}` to be substituted — see ICU Formatting).
- `Derafu\DataProcessor\Translation\DataProcessorTranslationResourceProvider` — a `TranslationResourceProviderInterface` implementation pointing at that directory, ready to hand to a `TranslationResourceRegistrar` or a dependency-injection container.

Neither of these does anything by itself — `derafu/data-processor` is a library, it doesn't build or own a `Translator`. Activating translation is entirely up to the application that uses it.

Activating It (Plain PHP)

```
use Derafu\DataProcessor\Exception\ValidationException;
use Derafu\DataProcessor\Translation\DataProcessorTranslationResourceProvider;
use Derafu\Translation\TranslatorFactory;

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

$e = new ValidationException(['Array must not contain more than {max} items.', 'max'
=> 5]);

echo $e->trans($translator, 'es');
// "El arreglo no debe contener más de 5 elementos."
```

If you'd rather point at the directory directly (for example, to combine it with your own translation files) without going through the provider class, use `TranslationResourceRegistrar` instead:

```
use Derafu\Translation\TranslationResourceRegistrar;

$registrar = new TranslationResourceRegistrar($translator);
$registrar->registerDirectory(__DIR__ . '/vendor/derafu/data-
processor/resources/translations');
$registrar->registerDirectory(__DIR__ . '/translations'); // Your own, registered
last: it can override.
```

Activating It (Dependency Injection)

If your application uses `symfony/dependency-injection`, import both recipe files and feed the tagged providers into your own `Translator` registration:

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

`data-processor-services.yaml` already tags `DataProcessorTranslationResourceProvider` with `derafu_translation.resource_provider`, so it's picked up automatically — nothing else to configure. Anything in your own application that type-hints `Symfony\Contracts\Translation\TranslatorInterface` (as `derafu/translation`'s own exceptions do) resolves to this `Translator` via the alias `translation-services.yaml` provides.

A Note on Rule Names vs. Messages

Rule names (`'email'`, `'max_length'`, the strings you pass to `process()`) are never translated — they're an internal API, not user-facing text. Only the **exception messages** thrown when a rule fails go through translation.

Last updated on 09/11/2026