Docs

Data Processor Project

Derafu Data Processor

Anonymous · 24/08/2026 · #php

Table of Contents

Data Processor Project

Derafu Data Processor

Anonymous · 24/08/2026 · #php

Derafu Data Processor

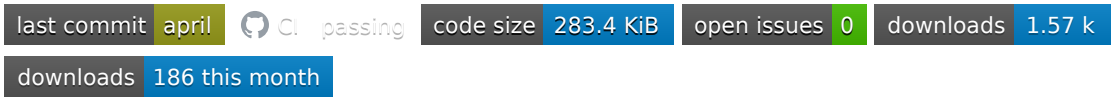
Last updated on 24/08/2026

Introduction

Four-Phase Data Processing Library

Anonymous · 24/08/2026

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.
- **Almost Zero Dependencies**: just `intl` extension for specific format rules and `Carbon` for dates.

Features

- **Type-Safe Casting**: Convert between data types safely.
    - Basic types (string, int, float, bool).
    - Date and time handling.
- **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, etc.).
    - File validations (file, image, mime_types).
    - Array validations (min_items, max_items, not_empty, etc.).

Installation

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

Basic Usage

```
use Derafu\DataProcessor\ProcessorFactory;

$processor = ProcessorFactory::create();

// 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', 'mime_types:application/pdf,image/jpeg']
```

```
]);
```

Last updated on 24/08/2026

Rules Reference

Rules Reference

Anonymous · 24/08/2026

Rules Reference

Cast Rules

Rule	Description	Example
`boolean`, `bool`	Convert value to boolean	`'cast' => 'boolean'`
`date`	Convert value to Carbon date	`'cast' => 'date'`
`datetime`	Convert value to Carbon datetime	`'cast' => 'datetime'`
`float`, `decimal`	Convert value to float	`'cast' => 'float'`
`integer`, `int`	Convert value to integer	`'cast' => 'integer'`
`string`, `str`	Convert value to string	`'cast' => 'string'`
`timestamp`	Convert value to Unix timestamp	`'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']`
`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']`

Rule	Description	Example
`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 non-printable characters	`'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']`
`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 whitespace from ends	`'sanitize' => ['trim']`

Validate Rules

Array Validation

Rule	Description	Example
`in`, `choices`	Check if value is in list	`'validate' => ['in:a,b,c']`
`max_items`	Check array maximum length	`'validate' => ['max_items:5']`
`min_items`	Check array minimum length	`'validate' => ['min_items:1']`
`notempty`	Check if array is not empty	`'validate' => ['notempty']`
`unique`	Check if array values are unique	`'validate' => ['unique']`

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']`
`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 limit	`'validate' => ['file:2M']`
`image`	Validate image file with size limit	`'validate' => ['image:2M']`
`mimetype`	Validate file mime type	`'validate' => ['mimetype:application/pdf,image/jpeg']`

Financial Validation

Rule	Description	Example
`bic`	Validate BIC/SWIFT code	`'validate' => ['bic']`
`card_number`	Validate credit card number	`'validate' => ['card_number']`
`iban`	Validate IBAN	`'validate' => ['iban']`

I18n Validation

Rule	Description	Example
`country`	Validate country code	`'validate' => ['country']`
`currency`	Validate currency code	`'validate' => ['currency']`
`language`	Validate language code	`'validate' => ['language']`
`locale`	Validate locale code	`'validate' => ['locale']`
`timezone`	Validate timezone identifier	`'validate' => ['timezone']`

ID Validation

Rule	Description	Example
`uuid`	Validate UUID string	`'validate' => ['uuid']`

Internet Validation

Rule	Description	Example
`email`	Validate email address	`'validate' => ['email']`
`hostname`	Validate hostname	`'validate' => ['hostname']`
`ip`	Validate IP address	`'validate' => ['ip']`, `'validate' => ['ip:v4']`, `'validate' => ['ip:v6']`
`url`	Validate URL	`'validate' => ['url']`

Numeric Validation

Rule	Description	Example
`decimal`, `float`, `real`	Validate decimal number	`'validate' => ['decimal:2']`
`digits_range`	Validate number of digits range	`'validate' => ['digits_range:3,5']`
`digits`	Validate exact number of digits	`'validate' => ['digits:4']`
`gt`	Greater than	`'validate' => ['gt:10']`

Rule	Description	Example
`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']`
`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']`
`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	`'validate' => ['required']`
`slug`	Validate URL friendly string	`'validate' => ['slug']`
`starts_with`	String starts with value	`'validate' => ['starts_with:Test']`

Last updated on 24/08/2026

Custom Registries

Creating Custom Registries

Anonymous · 24/08/2026

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
    ->addTransformRule('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 24/08/2026

Custom Rules

Creating Custom Rules

Anonymous · 24/08/2026

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 24/08/2026

Custom Registrar

Creating Custom Rule Registrar

Anonymous · 24/08/2026

Creating Custom Rule Registrar

For better organization, you can create a custom rule registrar:

```
use Derafu\DataProcessor\Contract\RuleRegistryInterface;

final class CustomRuleRegistrar implements RuleRegistrarInterface
{
    public function register(RuleRegistryInterface $registry): void
    {
        // Register caster rules.
        $registry
            ->addCasterRule('custom_type_1', CustomType1Rule::class)
            ->addCasterRule('custom_type_2', CustomType2Rule::class);

        // Register transform rules.
        $registry
            ->addTransformRule('custom_transform_1', CustomTransform1Rule::class)
            ->addTransformRule('custom_transform_2', CustomTransform2Rule::class);

        // Register sanitizer rules.
        $registry
            ->addSanitizerRule('custom_sanitize_1', CustomSanitize1Rule::class)
            ->addSanitizerRule('custom_sanitize_2', CustomSanitize2Rule::class);

        // Register validator rules.
        $registry
            ->addValidatorRule('custom_validate_1', CustomValidate1Rule::class)
            ->addValidatorRule('custom_validate_2', CustomValidate2Rule::class);
    }
}

// Usage.
$processor = ProcessorFactory::create(
    new CustomRuleRegistrar(),
    withDefaultRules: false // Create without default rules using the factory.
);
```

This allows you to:

- Create domain-specific rule sets.
- Override default rules with custom implementations.
- Group related rules together.
- Control which rules are available in your application.

Last updated on 24/08/2026

Custom Parser

Creating Custom Rule Parser

Anonymous · 24/08/2026

Creating Custom Rule Parser

You can create your own rule parser to customize how rules are parsed from strings or arrays:

```
use Derafu\DataProcessor\Contract\RuleParserInterface;

final class CustomRuleParser implements RuleParserInterface
{
    public function parse(string|array $rules): array
    {
        if (is_array($rules)) {
            return $this->parseArray($rules);
        }
        return $this->parseString($rules);
    }

    private function parseArray(array $rules): array
    {
        // Your array parsing logic here.
        $parsed = [];
        foreach ($rules as $type => $typeRules) {
            // Handle different formats for rules.
            $parsed[$type] = $this->parseTypeRules($typeRules);
        }
        return $parsed;
    }

    private function parseString(string $rules): array
    {
        // Your string parsing logic here.
        // Example: parse rules like "t(rule1|rule2) s(rule3) required|email"
        return [
            'transform' => ['rule1', 'rule2'],
            'sanitize' => ['rule3'],
            'validate' => ['required', 'email'],
        ];
    }

    private function parseTypeRules(string|array $rules): array|string
    {
        // Handle parsing of individual rule types.
```

```
        if (is_string($rules)) {
            return explode('|', $rules);
        }
        return $rules;
    }
}

// Usage with processor factory.
$processor = ProcessorFactory::create(
    parser: new CustomRuleParser()
);

// Or manual instantiation.
$registry = new RuleRegistry(); // Then register the rules you need.
$resolver = new RuleResolver($registry);
$parser = new CustomRuleParser();
$processor = new Processor($resolver, $parser);

// Using your custom parser.
$result = $processor->process('test@example.com', 't(lowercase) s(trim)
required|email');

// Or.
$result = $processor->process('test@example.com', [
    'transform' => 'lowercase',
    'sanitize' => 'trim',
    'validate' => 'required|email',
]);
```

This allows you to:

- Create custom rule parsing formats.
- Support different string formats for rules.
- Add custom shorthand notations.
- Implement domain-specific rule syntax.
- Handle complex rule configurations.

Last updated on 24/08/2026