

Types

Compose strict validation, scalar conversion, collections, enums, dates and custom types

Anonymous · 26/10/2026

ValueType

`ValueType<T>` validates or explicitly converts a value and returns it as `T`. PHPStan follows the generic return type through collection and union compositions.

```
use Derafu\Value\ValueType;

ValueType::int()->parse(12);          // 12
ValueType::toInt()->parse('12.0');   // 12
ValueType::number()->parse(12.0);    // 12.0, unchanged
ValueType::toNumber()->parse(12.0);  // 12
```

Available Factories

Factory	Contract
<code>mixed()</code>	Any value, including null.
<code>string()</code> , <code>int()</code> , <code>float()</code> , <code>bool()</code>	Exactly the indicated PHP type, without coercion.
<code>number()</code>	An existing integer or float, preserved unchanged.
<code>scalar()</code>	An existing boolean, integer, float or string.
<code>nonEmptyString()</code>	A string whose content is nonempty before and after PHP <code>trim()</code> .
<code>toInt()</code> , <code>toFloat()</code> , <code>toNumber()</code> , <code>toBool()</code> , <code>toString()</code>	Delegate to the corresponding <code>ValueCaster</code> method, excluding null.
<code>array()</code>	Any PHP array.
<code>map()</code>	An array with string keys only.
<code>list()</code>	An array with consecutive integer keys starting at zero.
<code>arrayOf(\$type)</code>	Any array whose values match the supplied type.
<code>mapOf(\$type)</code>	A string-keyed array whose values match the supplied type.
<code>listOf(\$type)</code>	A list whose values match the supplied type.

Factory	Contract
<code>literal(\$first, ...\$others)</code>	One of the supplied boolean, integer or string values, compared strictly.
<code>instanceOf(\$class)</code>	An object of the given class or interface, returned unchanged.
<code>backedEnum(\$class)</code>	An existing case or an exact int/string backing value of the requested backed enum.
<code>dateTime()</code>	A <code>DateTimeInterface</code> , integer timestamp or non-blank date string interpreted by PHP.
<code>oneOf(\$first, ...\$others)</code>	The first type that accepts the value.

The primitive `float()` rejects integers even though `ValueCaster::toFloat()` can convert them. Use an explicit conversion type when conversion is intended.

Nullability

Apply `nullable()` to accept null in addition to a type's ordinary values:

```
use Derafu\Value\ValueType;

ValueType::int()->nullable()->parse(null);    // null
ValueType::toInt()->nullable()->parse(null);  // null
```

`ValueCaster` preserves null directly, but conversion types reject it by default.

`ValueType::toInt()->parse(null)` throws `TypeMismatchException`. This keeps nullability explicit in a composed type.

Collections

Empty collections are accepted. Lists require consecutive integer keys; maps require string keys; arbitrary arrays accept both. PHP converts some numeric string keys to integers, so `['123' => 'value']` is not a map, while `['01' => 'value']` is.

```
use Derafu\Value\ValueType;

$type = ValueType::arrayOf(ValueType::mapOf(ValueType::toInt()));
$type->parse([3 => ['quantity' => '12.0'], 8 => ['quantity' => '2']]);
// [3 => ['quantity' => 12], 8 => ['quantity' => 2]]
```

Collections preserve their keys and report nested errors with paths such as `rows.8.quantity`. These factories describe element types; `mapOf()` does not define required or optional named fields.

Unions and Literal Values

```
use Derafu\Value\ValueType;

$type = ValueType::oneOf(ValueType::toInt(), ValueType::string());
$type->parse('12.0'); // 12
$type->parse('text'); // 'text'

ValueType::literal('asc', 'desc')->parse('asc'); // 'asc'
ValueType::literal(false)->parse(false);          // false; 0 would be rejected
```

Union order matters because conversion types can interpret the input. `oneOf()` tries the next candidate when the current candidate throws a translatable invalid-argument exception. It does not catch arbitrary runtime failures.

Backed Enums

```
use Derafu\Value\ValueType;

enum Status: string
{
    case Pending = 'pending';
    case Approved = 'approved';
}

$type = ValueType::backedEnum(Status::class);
$type->parse('approved');          // Status::Approved
$type->parse(Status::Pending);     // the same enum case
```

Int-backed enums accept integer backing values; string-backed enums accept strings. Numeric strings are not coerced to integers, and another enum's case is not accepted just because it has the same backing value. Unknown values throw `TypeMismatchException`. Passing a class that is not a backed enum to the factory throws `ValueException`.

Dates

`dateTime()` preserves an existing `DateTimeInterface`. An integer timestamp becomes a `DateTimeImmutable` using `@timestamp`; date text is parsed by `DateTimeImmutable`. Relative dates, timezones and date normalization follow PHP's parser. Empty or invalid input throws `TypeMismatchException`.

```
use Derafu\Value\ValueType;

$date = ValueType::dateTime()->parse('2026-01-15');
```

```
$date->format('Y-m-d'); // '2026-01-15'
```

Custom Types and Error Context

The constructor takes a diagnostic name and a closure receiving the value, path and optional source. Return the typed result or throw `ValueType::mismatch()`:

```
use Derafu\Value\ValueType;

$even = new ValueType(
    'even',
    static fn (mixed $value, string $path, ?string $source): int
        => is_int($value) && $value % 2 === 0
        ? $value
        : throw ValueType::mismatch($path, 'even', $value, $source),
);

$even->parse(4, 'workers', 'settings.php'); // 4
```

`parse($value, $path = '', $source = null)` attaches a diagnostic location and origin to failures. An empty root path is displayed as `$`. `ValueType::path($parent, $key)` joins a diagnostic path with a string or integer key; it does not navigate data. See [Exceptions and Translations](#).

Last updated on 26/10/2026