

Casting

Read scalar values and objects strictly or convert scalars explicitly, with nullable and required read methods

Anonymous · 26/10/2026

ValueCaster

`ValueCaster` exposes strict reads and explicit conversions. Every method accepts `mixed`; invalid input throws instead of silently becoming a default value.

Strict Reads

`getXXX()` accepts null and otherwise requires the supported input type. `requireXXX()` rejects null as well as incorrect types. Reads preserve existing types, with `Stringable` accepted specifically by the string methods.

Nullable read	Required read	Accepted non-null input
<code>getString(\$value): ?string</code>	<code>requireString(\$value): string</code>	string or <code>Stringable</code>
<code>getInt(\$value): ?int</code>	<code>requireInt(\$value): int</code>	int only
<code>getFloat(\$value): ?float</code>	<code>requireFloat(\$value): float</code>	float only
<code>getNumber(\$value): int float null</code>	<code>requireNumber(\$value): int float</code>	int or float, preserving its type
<code>getBool(\$value): ?bool</code>	<code>requireBool(\$value): bool</code>	bool only
<code>getObject(\$value, \$class): ?T</code>	<code>requireObject(\$value, \$class): T</code>	instance of the class or interface, preserving the object

```
use Derafu\Value\Exception\TypeMismatchException;
use Derafu\Value\ValueCaster;

ValueCaster::getString(null);      // null
ValueCaster::requireString(' text '); // ' text '
ValueCaster::requireInt(12);       // 12
ValueCaster::requireNumber(12.0);  // 12.0, still a float
ValueCaster::getBool(false);       // false
ValueCaster::requireObject($pool, CacheItemPoolInterface::class); // $pool
```

```
try {
    ValueCaster::getString(5);
} catch (TypeMismatchException $exception) {
    // Numeric input is not accepted by strict string reads.
}
```

String reads convert Stringable objects through the existing `toString()` operation, without trimming. Numbers and booleans are rejected by those reads. `ValueType::string()` remains a strict check of a PHP string and does not accept Stringable.

```
use Derafu\Value\ValueCaster;

$text = new class () implements Stringable {
    public function __toString(): string
    {
        return ' #ABCDEF ';
    }
};

ValueCaster::getString($text);    // ' #ABCDEF '
ValueCaster::requireString($text); // ' #ABCDEF '
ValueCaster::toString(5);        // '5': explicit conversion accepts numeric input
```

Strict read failures raise the translatable `TypeMismatchException`. A failing Stringable conversion preserves its `ValueConversionException` and the original throwable in the cause chain. These methods operate on one supplied value; they do not read keys or assign defaults.

Explicit Conversions

The `toXXX()` methods preserve null and convert their supported inputs. Unsupported conversions raise `ValueConversionException`.

Method	Return type
<code>toInt(\$value)</code>	<code>?int</code>
<code>toFloat(\$value)</code>	<code>?float</code>
<code>toNumber(\$value)</code>	<code>int float null</code>
<code>toBool(\$value)</code>	<code>?bool</code>
<code>toString(\$value)</code>	<code>?string</code>

Integers

`toInt()` accepts integers, integral finite floats within the platform integer range, and numeric strings representing an exact integer in that range. It rejects boolean inputs and never truncates a fraction.

```
use Derafu\Value\ValueCaster;

ValueCaster::toInt('0012'); // 12
ValueCaster::toInt('12.0'); // 12
ValueCaster::toInt('1e3'); // 1000
ValueCaster::toInt(12.0); // 12
ValueCaster::toInt(null); // null
```

Numeric strings are interpreted without a float intermediate. For example, `'1.00000000000000000001'` is rejected as fractional even though converting it to float could hide its fraction. Accepted integers must fit between `PHP_INT_MIN` and `PHP_INT_MAX`.

Floats

`toFloat()` accepts integers, floats and complete numeric strings according to PHP's `is_numeric()` rules. Numeric text may contain signs, decimal points, exponent notation and exterior numeric whitespace. Text such as `'12abc'`, `'1,25'`, `'1_000'`, `'INF'` and `'NAN'` is rejected. Booleans are also rejected.

```
use Derafu\Value\ValueCaster;

ValueCaster::toFloat('0012'); // 12.0
ValueCaster::toFloat('1e3'); // 1000.0
ValueCaster::toFloat('1e400'); // INF
ValueCaster::toFloat('1e-400'); // 0.0
ValueCaster::toFloat(INF); // INF
is_nan(ValueCaster::toFloat(NAN)); // true
```

The conversion uses PHP's float representation, including rounding, infinity and underflow to zero. It does not preserve the spelling or all decimal digits of the source text. The optional policies in [ValueParser](#) can retain numeric text when interpretation would alter its format or significant decimal precision.

Numbers

`toNumber()` calls `toInt()` first. If that conversion fails, it calls `toFloat()`. Failure of the second conversion is propagated as `ValueConversionException`.

```
use Derafu\Value\ValueCaster;

ValueCaster::toNumber('12.0'); // 12
ValueCaster::toNumber(12.0); // 12
ValueCaster::toNumber('12.5'); // 12.5
ValueCaster::toNumber(null); // null
```

Use this method when either numeric PHP type is acceptable. It inherits both conversion contracts, including float overflow and underflow. Request `toFloat()` when the result must specifically be a float.

Booleans

Accepted input	Result
<code>true</code> , <code>1</code> , <code>1.0</code> , <code>'1'</code> , <code>'true'</code> , <code>'yes'</code> , <code>'on'</code>	<code>true</code>
<code>false</code> , <code>0</code> , <code>0.0</code> , <code>'0'</code> , <code>'false'</code> , <code>'no'</code> , <code>'off'</code> , <code>''</code>	<code>false</code>
<code>null</code>	<code>null</code>

String tokens use PHP's boolean validation filter, which trims exterior whitespace and compares case-insensitively. Whitespace-only text therefore becomes `false`. Other values, such as `2`, `0.5` or `'unknown'`, throw. This is not PHP's direct boolean cast: the text `'false'` becomes `false`.

Strings

`toString()` accepts strings, integers, floats, booleans and `Stringable` objects. Existing text is preserved without trimming. Booleans become `'1'` and `'0'`; numbers use PHP's string representation, with native NAN represented as `'NAN'`.

```
use Derafu\Value\ValueCaster;

ValueCaster::toString(false);    // '0'
ValueCaster::toString(true);     // '1'
ValueCaster::toString(12.5);     // '12.5'
ValueCaster::toString(' text '); // ' text '
ValueCaster::toString(null);     // null
```

Arrays, resources and objects without `Stringable` are rejected. Numeric and boolean conversions do not accept `Stringable` objects; use the parser when their text should be interpreted automatically. A failing `__toString()` is wrapped in `ValueConversionException`, retaining the original throwable as its cause.

Last updated on 26/10/2026