

Parsing

Interpret scalar text and preserve numeric format or significant decimal precision when requested

Anonymous · 26/10/2026

ValueParser

```
ValueParser::parse(  
    mixed $value,  
    bool $preserveNumericFormat = false,  
    bool $preserveNumericPrecision = false,  
): mixed
```

The parser interprets one value. `Stringable` objects are converted to text through `ValueCaster::toString()`. Strings are interpreted using the sequence below. Every other value is returned unchanged, including native integers, floats, booleans, nulls, arrays, resources and other objects.

Interpretation Order

1. Recognize the special literal representations.
2. Try `ValueCaster::toNumber()`, which tries int before float.
3. If numeric conversion fails, try `ValueCaster::toBool()`.
4. If boolean conversion also fails, return the original text.

The parser delegates conversion rules to `ValueCaster`. It does not define a separate numeric grammar. An unsuccessful interpretation of plain text is not an exception; a failing `Stringable` conversion is propagated as `ValueConversionException`.

Literal text	Result
'null', '(null)'	null
'empty', '(empty)'	''
'true', '(true)'	true
'false', '(false)'	false

These tokens are recognized case-insensitively after trimming exterior whitespace. The word `'nil'` is not recognized.

```
use Derafu\Value\ValueParser;
```

```

ValueParser::parse(' (NULL) '); // null
ValueParser::parse('empty');    // ''
ValueParser::parse('0012');     // 12
ValueParser::parse('12.0');     // 12
ValueParser::parse('12.5');     // 12.5
ValueParser::parse('1e3');      // 1000
ValueParser::parse('yes');      // true
ValueParser::parse('0');        // 0, not false
ValueParser::parse('1');        // 1, not true
ValueParser::parse(' ');        // false, through toBool()
ValueParser::parse('');         // false, unlike the literal 'empty'
ValueParser::parse(' text ');   // ' text '

```

Numeric Format Preservation

With `preserveNumericFormat: true`, a successful numeric conversion is returned only if converting the number back through `ValueCaster::toString()` reproduces the complete original text. Otherwise the original text is retained. Exterior whitespace is part of this comparison.

```

use Derafu\Value\ValueParser;

ValueParser::parse('0012', preserveNumericFormat: true); // '0012'
ValueParser::parse('19.90', preserveNumericFormat: true); // '19.90'
ValueParser::parse('12.0', preserveNumericFormat: true);  // '12.0'
ValueParser::parse('1e3', preserveNumericFormat: true);   // '1e3'
ValueParser::parse(' 12 ', preserveNumericFormat: true); // ' 12 '
ValueParser::parse('12', preserveNumericFormat: true);    // 12
ValueParser::parse('0.5', preserveNumericFormat: true);   // 0.5

```

This comparison follows `ValueCaster`'s PHP number-to-string representation. It is a numeric policy: boolean words and special literals are still interpreted normally. Preserved numeric text does not fall through to boolean conversion.

Numeric Precision Preservation

With `preserveNumericPrecision: true`, non-finite numeric results and floats that fail the significant-decimal-digit comparison retain their original text. Successfully converted integers are exact and need no float comparison.

For floats, the parser compares significant mantissa digits with the shortest decimal representation found by its formatting round trip. Signs, the decimal point and insignificant leading/trailing zeros are excluded from the digit comparison. This is a decimal interpretation policy, not a requirement that fractions be exactly representable in binary.

```

use Derafu\Value\ValueParser;

```

```
ValueParser::parse('1.00000000000000000001', preserveNumericPrecision: true);
// '1.00000000000000000001'

ValueParser::parse('1e400', preserveNumericPrecision: true); // '1e400'
ValueParser::parse('1e-400', preserveNumericPrecision: true); // '1e-400'
ValueParser::parse('0.1', preserveNumericPrecision: true); // 0.1
ValueParser::parse('12.5000', preserveNumericPrecision: true); // 12.5
ValueParser::parse('0012', preserveNumericPrecision: true); // 12
```

Without this option, conversion follows ValueCaster, including float rounding, infinity and underflow. Precision preservation alone does not preserve spelling. Both options can be enabled together:

```
use Derafu\Value\ValueParser;

ValueParser::parse(
    '0012',
    preserveNumericFormat: true,
    preserveNumericPrecision: true,
); // '0012'
```

Existing Values and Stringable Objects

```
use Derafu\Value\ValueParser;

$text = new class () implements Stringable {
    public function __toString(): string
    {
        return '0012';
    }
};

ValueParser::parse($text); // 12
ValueParser::parse($text, preserveNumericFormat: true); // '0012'
ValueParser::parse(12.0); // 12.0; an existing float is returned unchanged
ValueParser::parse(['null', '2']); // ['null', '2']; arrays are not traversed
```

The numeric options apply to interpreted text, including Stringable text. They do not transform existing native numbers. Recursive array processing belongs to a caller that explicitly traverses the array.