Docs

Date Project

Derafu Date

Anonymous · 05/11/2026 · #php

Table of Contents

Date Project 3
- ***Introduction*** 4
- ***Months*** 6
- ***Periods*** 8
- ***Date Units*** 11
- ***Parsing*** 13
- ***Calculations*** 15
- ***Working Days*** 16
- ***Formatting*** 18
- ***Clocks*** 20
- ***Errors and Translations*** 21

Date Project

Derafu Date

Anonymous · 05/11/2026 · #php

Derafu Date

Last updated on 05/11/2026

Introduction

Months, periods, working days, date parsing and translatable date messages in PHP, with immutable objects and no hidden clock

Anonymous · 05/11/2026

Derafu: Date



Calendar months, periods of any length, working days, strict date parsing and the words that describe a date, built on `DateTimeImmutable`.

Nothing in it changes a date that is given and nothing reads the current time by itself: what depends on “now” takes a PSR-20 clock. The words that describe a date (`Monday`, `3 days ago`) are not written in a language: they are translatable messages, so a language is added with a catalogue and without code.

Install

```
composer require derafu/date
```

It requires PHP 8.5 or later, `derafu/translation` (the messages) and `psr/clock`. It has no other dependency.

The classes

Class	What it is for	Page
`Month`	A month of the calendar (`2025-11`), also as the number `202511`.	Months
`Period`	A range of days: a month, a quarter, a year, or from the 15th to the 14th.	Periods
`DateUnit` and `Weekday`	The units of time, to move a date, and the days of the week.	Date Units
`DateParser`	Read and validate dates, and the periods written as numbers.	Parsing

Class	What it is for	Page
`DateCalculator`	Age, days and months between dates, weeks, lists of years.	Calculations
`WorkCalendar`	Working days, with the holidays that are given to it.	Working Days
`DateFormatter`	Describe dates, months and elapsed time as translatable messages.	Formatting
`SystemClock` and `FixedClock`	The current moment, and a moment that does not change.	Clocks

The errors are in Errors and Translations.

Example

```
use Derafu\Date\Enum\DateUnit;
use Derafu\Date\Service\DateFormatter;
use Derafu\Date\ValueObject\Month;
use Derafu\Date\ValueObject\Period;
use Derafu\Date\ValueObject\WorkCalendar;

// A billing cycle that goes from the 15th to the 14th.
$cycle = Period::starting('2025-11-15', DateUnit::MONTH);   // 2025-11-15/2025-12-14
$next = $cycle->next();                                     // 2025-12-15/2026-01-14

// The tenth working day of a month, with the holidays of the country.
$calendar = new WorkCalendar(['2025-11-01', '2025-12-08']);
$day = $calendar->getNthWorkingDay(Month::of(2025, 12), 10);

// A message that is translated where the language is known.
$message = (new DateFormatter())->formatDate($day);
$message->trans($translator, 'es');   // Lunes, 15 de Diciembre del 2025
```

Last updated on 05/11/2026

Months

Month, an immutable calendar month that is written as AAAA-MM or as the number AAAAMM

Anonymous · 05/11/2026

Months

```
use Derafu\Date\ValueObject\Month;
```

A `Month` is a year and a month of that year. It does not change: moving it gives another month. It is written as text (`2025-11`) and, for what keeps months as numbers, as an integer (`202511`).

Create

Method	What it does
`Month::of($year, $month)`	From a year (1 to 9999) and a number (1 to 12).
`Month::fromString('2025-11')`	From a text written as `AAAA-MM`.
`Month::fromInt(202511)`	From a number written as `AAAAMM`.
`Month::tryFromInt(202511)`	The same, and `null` if it is not a month.
`Month::fromDate($date)`	The month of a date.
`Month::current($clock = null)`	The current month, from a clock.

Anything that is not a month is an `InvalidMonthException`: month 13, a year out of 1 to 9999, a text with a day (`2025-11-01`), a number that is not six digits.

Ask

Method	What it gives
`getYear()`, `getMonth()`	The parts.
`getFirstDay()`, `getLastDay()`	The first and the last day, at the start of the day (`DateTimeImmutable`).
`getDays()`	The number of days of the month (28 to 31).
`contains($date)`	Whether a date is in the month.
`toString()`, `toInt()`	`2025-11` and `202511`. A month is also `Stringable` and serializes to JSON as its text.

Method	What it gives
`toPeriod()`	The period of the month.

Move and compare

```
$month = Month::of(2025, 11);

$month->next();        // 2025-12
$month->next(3);       // 2026-02
$month->previous(11);  // 2024-12
$month->monthsUntil(Month::of(2026, 3));   // 4

$month->isBefore($other);
$month->isAfter($other);
$month->equals($other);
$month->compareTo($other);   // negative, zero or positive
```

`next()` and `previous()` take any number of steps (a negative number goes the other way). A result out of the years 1 to 9999 is an `InvalidMonthException`.

Last updated on 05/11/2026

Periods

Period, an immutable range of days that can be a month, a quarter, a semester, a year or any cycle

Anonymous · 05/11/2026

Periods

```
use Derafu\Date\ValueObject\Period;
```

A `Period` is a range of days, from a first day to a last day, **both included**. The time of the day is not part of it. It can be any range: a month, a quarter, a year, several months, or from the 15th of a month to the 14th of the next one.

It does not change: moving it gives another period. It is `Stringable` (`2025-11-15/2025-12-14`) and serializes to JSON as `{"start": "2025-11-15", "end": "2025-12-14"}`.

Create

Method	What it does
`Period::between($start, $end)`	From a first day to a last day (dates or texts).
`Period::month($year, $month)`	A month.
`Period::fromMonth($month)`	The period of a `Month`.
`Period::months($start, $end)`	From the first day of a month to the last day of another one.
`Period::quarter($year, $quarter)`	A quarter, from 1 to 4.
`Period::semester($year, $semester)`	A semester, 1 or 2.
`Period::year($year)`	A year.
`Period::starting($date, $unit, $count = 1)`	Some units from a date: it ends the day before the same date of the units that follow.

```
Period::starting('2025-11-15', DateUnit::MONTH);      // 2025-11-15/2025-12-14
Period::starting('2025-01-01', DateUnit::MONTH, 3);   // 2025-01-01/2025-03-31
Period::starting('2025-07-01', DateUnit::YEAR);       // 2025-07-01/2026-06-30
```

An end before the start, a quarter that is not 1 to 4, a semester that is not 1 or 2 and a number of units less than 1 are an `InvalidPeriodException`.

Ask

Method	What it gives
`getStart()`, `getEnd()`	The first and the last day, at the start of the day.
`getStartMonth()`, `getEndMonth()`	The months of both.
`getDays()`	The number of days.
`getMonths()`, `countMonths()`	The months that have days of the period, and how many.
`contains($dateOrPeriod)`	Whether a date, or a whole period, is inside.
`overlaps($period)`, `equals($period)`	Whether they share days, or are the same.

Alignment

`isAlignedTo($unit)` tells if the period starts at the beginning of a unit and ends at its end. A period from January 1 to June 30 is aligned to months, quarters and semesters, and it is not aligned to years. Every period is aligned to days, and a week starts on Monday.

Move and split

```
$quarter = Period::quarter(2025, 4);
$quarter->next();       // 2026-01-01/2026-03-31
$quarter->previous(2);  // 2025-04-01/2025-06-30

Period::semester(2025, 2)->split(DateUnit::QUARTER);
// 2025-07-01/2025-09-30 and 2025-10-01/2025-12-31
```

`next()` and `previous()` give the period that follows or precedes with the same length. A period that is a whole number of months from its first day (a month, a quarter, or from March 15 to April 14) moves by that number of months; any other one moves by the same number of days.

```
Period::between('2025-03-15', '2025-04-14')->next();   // 2025-04-15/2025-05-14
Period::between('2025-01-01', '2025-01-10')->next();   // 2025-01-11/2025-01-20
```

`split($unit, $count = 1)` cuts the period in consecutive periods of that unit, counted from its first day. The last one ends where the period ends, so it can be shorter.

The year

Method	What it does
`isSingleYear()`	Whether it starts and ends in the same year.

Method	What it does
`getYear()`	The year, or an `InvalidPeriodException` if it has days of more than one year.
`requireSingleYear()`	The same period, or an `InvalidPeriodException`.
`startsOnYearStart()`	Whether it starts on January 1.
`isBeforeStartInSameYear($date)`	Whether a date is before the start, in the year of the start.

Last updated on 05/11/2026

Date Units

DateUnit, to move a date by days, weeks, months, quarters, semesters and years, and Weekday

Anonymous · 05/11/2026

Date Units

```
use Derafu\Date\Enum\DateUnit;
use Derafu\Date\Enum\Weekday;
```

DateUnit

`DateUnit` is an enum of the units of time. The value of each case is its one-letter code.

Case	Code	Months
`DAY`	`D`	
`WEEK`	`W`	
`MONTH`	`M`	1
`QUARTER`	`Q`	3
`SEMESTER`	`S`	6
`YEAR`	`Y`	12

`DateUnit::fromCode('m')` gives the unit of a code in upper or lower case, and an `InvalidDateException` if there is none. `getMonths()` gives the months of the unit, or `null` for days and weeks.

Move a date

```
DateUnit::MONTH->add($date, 1);        // a month later
DateUnit::QUARTER->subtract($date, 2); // two quarters earlier
DateUnit::DAY->add($date, -3);         // a negative number goes the other way
```

`add($date, $steps = 1)` and `subtract($date, $steps = 1)` give a new `DateTimeImmutable`; the date that is given does not change and its time is kept.

Months, quarters, semesters and years never overflow: the day that the month of the result does not have is the last day of that month.

Date	Move	Result
`2024-01-31`	+ 1 month	`2024-02-29`
`2023-01-31`	+ 1 month	`2023-02-28`
`2024-08-31`	+ 1 semester	`2025-02-28`
`2024-02-29`	+ 1 year	`2025-02-28`

Weekday

`Weekday` numbers the days of the week like ISO 8601: Monday is `1` and Sunday is `7`. `Weekday::fromDate($date)` gives the day of the week of a date. A work calendar uses it to say which days are the weekend.

Last updated on 05/11/2026

Parsing

Read and validate dates, strictly or like PHP does, and validate the periods written as numbers

Anonymous · 05/11/2026

Parsing

```
use Derafu\Date\Support\DateParser;
```

Method	What it does
`DateParser::parse($text)`	Reads a date like PHP does.
`DateParser::parseStrict($text, $format = 'Y-m-d')`	Reads a date written exactly in a format.
`DateParser::ensure($dateOrText)`	A `DateTimeImmutable` from a date or a text.
`DateParser::isValid($text, $format = 'Y-m-d')`	Whether a text is a date written exactly in the format.
`DateParser::reformat($text, $format = 'd/m/Y')`	Writes a `Y-m-d` date in another format.
`DateParser::fromSpreadsheetSerial($number)`	The date of a spreadsheet serial number.

Lenient and strict

`parse()` is as lenient as `DateTimeImmutable`: it accepts relative texts (`tomorrow`), and a day that the month does not have is moved to the next month.

```
DateParser::parse('2025-02-31');   // 2025-03-03
```

`parseStrict()` and `isValid()` read a date in a format and refuse what is not a date: it must exist and have every field of the format. They do not accept relative texts, a day that the month does not have or extra text.

```
DateParser::parseStrict('15/01/2024', 'd/m/Y');   // 2024-01-15 00:00:00
DateParser::isValid('2025-02-31');                // false
DateParser::isValid('tomorrow');                  // false
```

They read the fields as PHP does, so the digits can be fewer than the format writes: `2025-1-5` is the date `2025-01-05`, and a year written `24` is the year 24. They are for reading; `->format('Y-m-d')` gives the date in its canonical form, and `reformat()` refuses what is not written with all its digits.

`parseStrict()` gives the start of the day. `parse()` and `parseStrict()` throw an `InvalidDateException` when the text is not a date.

Reformat

`reformat()` is for a date that must be exactly `Y-m-d`. It gives `null` if it is not one, and that includes a date written without the zeros (`2024-1-5`).

```
DateParser::reformat('2023-03-12');            // 12/03/2023
DateParser::reformat('2023-03-12', 'Y.m.d');   // 2023.03.12
DateParser::reformat('12/03/2023');            // null
```

Spreadsheet serial numbers

Spreadsheets number the days since 1900: `1` is January 1, 1900. They also count a day that never existed, February 29, 1900, which is the number `60` and gives February 28.

```
DateParser::fromSpreadsheetSerial(45292);   // 2024-01-01
```

Periods written as numbers

A period written as a number is a year (`2025`) or a year and a month (`202511`).

Method	What it checks
`DateParser::isValidPeriod($period, $yearFrom = 2000, $yearTo = 2100, $length = null)`	A year or a month, within the years. With `$length` (4 or 6) it must have those digits.
`DateParser::isValidYearPeriod($period, ...)`	A year.
`DateParser::isValidMonthPeriod($period, ...)`	A year and a month.

To work with the months, use `Month::fromInt()`.

Last updated on 05/11/2026

Calculations

Age, days and months between dates, matching days, weeks and lists of years

Anonymous · 05/11/2026

Calculations

```
use Derafu\Date\Support\DateCalculator;
```

What depends on the current date takes a clock as its last argument, and uses the clock of the system if it is not given one. The dates are `DateTimeInterface` objects or texts.

Method	What it gives
`DateCalculator::age($birth, $clock = null)`	The age in years.
`DateCalculator::daysBetween($from, $to = null, $clock = null)`	The complete days from a date to another one (negative if the second is before). Without `$to`, until the clock.
`DateCalculator::monthsBetween($from, $to = null, $clock = null)`	The months between the months of two dates, without looking at the days. A period written as `202511` is accepted instead of a date.
`DateCalculator::countMatchingDays($from, $to, $days, $excludeWeekend = false)`	How many days of a range are in a list of `Y-m-d` dates.
`DateCalculator::firstDayOfWeek($date = null, $clock = null)`	Monday of the week, at the start of the day.
`DateCalculator::lastDayOfWeek($date = null, $clock = null)`	Sunday of the week, at the end of the day.
`DateCalculator::years($total, $from = null, $clock = null)`	A list of years, from the most recent to the oldest.

```
DateCalculator::monthsBetween(202301, '2024-01-15');   // 12
DateCalculator::monthsBetween('2024-01-15', '2024-02-14');   // 1
DateCalculator::years(3, 2024);   // [2024, 2023, 2022]
```

Last updated on 05/11/2026

Working Days

WorkCalendar, an immutable calendar of working days with the weekend and the holidays that are given to it

Anonymous · 05/11/2026

Working Days

```
use Derafu\Date\ValueObject\WorkCalendar;
```

A `WorkCalendar` knows which days are not working days: the days of the week that are weekend and the holidays. It does not change: the `with*()` methods give another calendar.

The holidays depend on the country and the year, so the package does not bring them: whoever uses the calendar gives them.

```
use Derafu\Date\Enum\Weekday;

$calendar = new WorkCalendar(
    holidays: ['2025-01-01', '2025-05-01'],
    weekend: [Weekday::SATURDAY, Weekday::SUNDAY],   // the default
);
```

The holidays are dates or texts written as `Y-m-d` (another text is an `InvalidDateException`). They are kept sorted and without repeating.

Ask

Method	What it does
`isHoliday($date)`	Whether the date is a holiday.
`isWorkingDay($date)`	Whether it is not weekend and not a holiday.
`addWorkingDays($date, $days)`	The date some working days later (earlier, if negative). The time is kept.
`subtractWorkingDays($date, $days)`	The date some working days earlier.
`getWorkingDayNumber($date)`	The number of a working day in its month (the first is 1), or `null` if the date is not a working day.
`getNthWorkingDay($month, $number)`	The date of the working day of a month with a number, or `null` if the month does not have so many. A number less than 1 is an `InvalidDateException`.

Method	What it does
`isLastWorkingDay($date)`	Whether it is the last working day of its month.
`countWorkingDays($from, $to)`	The working days of a range, both ends included.
`getHolidays()`, `getWeekend()`	What the calendar was given.

```
$calendar->addWorkingDays('2025-01-03', 1);               // 2025-01-06 (Monday)
$calendar->getNthWorkingDay(Month::of(2025, 3), 1);       // 2025-03-03 (March 1 is a
Saturday)
$calendar->getWorkingDayNumber('2025-03-03');             // 1
```

Other calendars

```
$calendar->withHolidays(['2026-01-01']);                    // other holidays, same
weekend
$calendar->withWeekend([Weekday::FRIDAY, Weekday::SATURDAY]); // other weekend, same
holidays
```

Last updated on 05/11/2026

Formatting

DateFormatter describes dates, months and elapsed time as translatable messages, so a language is added with a catalogue and no code

Anonymous · 05/11/2026

Formatting

```
use Derafu\Date\Service\DateFormatter;
```

`DateFormatter` describes a date, a month and the time that has passed. It does not give text: every method gives a `TranslatableMessage`, and the language is decided by whoever translates it.

Method	Message
`formatDate($date, $includeWeekday = true)`	`Monday, January 15, 2024`
`formatMonth($month)`	`January 2024`. It takes a `Month` or a number written as `AAAAMM`.
`formatElapsed($date, $full = false)`	`3 days ago`, `1 year, 5 days ago`, `just now`.

```
$formatter = new DateFormatter();

$message = $formatter->formatDate('2024-01-15');

$message->trans($translator, 'es');   // Lunes, 15 de Enero del 2024
(string) $message;                    // Monday, January 15, 2024
```

Without a translator the message is written in English, which is the language of its id.

`formatElapsed()` describes the time between the date and the clock (the system clock, or the one that is given to the constructor). It gives the two largest units (`1 year, 5 days`), or all of them with `$full`: years, months, weeks, days, hours, minutes and seconds. A difference of nothing is `just now`.

The catalogues

The names and phrases are in the translation domain `date`. The English text of each message is its id, and the catalogue of a language translates it:

```
// resources/translations/date+intl-icu.es.php
return [
    'Monday' => 'Lunes',
```

```
    'January' => 'Enero',
    '{weekday}, {month} {day}, {year}' => '{weekday}, {day} de {month} del {year}',
    '{month} {year}' => '{month} de {year}',
    '{count, plural, one {# day} other {# days}}' => '{count, plural, one {# día}
other {# días}}',
    '{parts} ago' => 'hace {parts}',
    'just now' => 'recién',
];
```

The Spanish catalogue ships with the package. The order of the words, the plurals and the names are all in the catalogue, so another language is one more file (`date+intl-icu.fr.php`) and no code. The catalogue of a language must have every id of the domain `date`.

To have the catalogue in your translator, import the services of the package (see Errors and Translations).

Last updated on 05/11/2026

Clocks

SystemClock and FixedClock, the PSR-20 clocks of the package

Anonymous · 05/11/2026

Clocks

```
use Derafu\Date\Service\FixedClock;
use Derafu\Date\Service\SystemClock;
```

Nothing in the package reads the current time by itself. What depends on “now” (`Month::current()`, the age, the days until today, `formatElapsed()`) takes a `Psr\Clock\ClockInterface`, and uses `SystemClock` if it is not given one.

Class	What it gives
`SystemClock`	The current moment.
`FixedClock`	Always the same moment, given as a date or as a text.

```
$clock = new FixedClock('2024-01-15 12:00:00');

Month::current($clock);                              // 2024-01
DateCalculator::age('2004-01-16', $clock);           // 19
(new DateFormatter($clock))->formatElapsed('2024-01-15 11:45:00');   // 15 minutes ago
```

A `FixedClock` is for tests and for calculations that must be made as of a date. Any other PSR-20 clock works in their place.

Last updated on 05/11/2026

Errors and Translations

The exceptions of Derafu Date and how to translate their messages

Anonymous · 05/11/2026

Errors and Translations

Every error of the package is a `Derafu\Date\Exception\DateException`, which extends `TranslatableInvalidArgumentException` of `derafu/translation`. Catching `DateException`, or PHP's `InvalidArgumentException`, catches all of them.

Exception	When
`InvalidDateException`	A text is not a date, a date is not valid in its format, a time unit does not exist, or a number of the working day is less than 1.
`InvalidMonthException`	A month, a text `AAAA-MM` or a number `AAAAMM` is not a month, or a move leaves the years 1 to 9999.
`InvalidPeriodException`	The end of a period is before its start, a quarter or a semester does not exist, a number of units is less than 1, or a period must be in a single year and it is not.

The last two extend `DateException`, like the first one.

```
try {
    Month::fromInt(202513);
} catch (InvalidMonthException $e) {
    $e->getMessage();   // Invalid month period: 202513.
}
```

Translations

The messages of the exceptions (domain `errors`) and the names and phrases of the dates (domain `date`) are translatable, and the Spanish translations ship with the package. To have them in your translator, import its services (or add `DateTranslationResourceProvider` to the resource providers of the translator):

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

The same file registers `DateFormatterInterface` and a `SystemClock` for `Psr\Clock\ClockInterface`.

With the translator, the message of the example above is `El período 202513 no es un mes, debe escribirse AAAAMM.`

Last updated on 05/11/2026