Package 'messydates'

July 27, 2026

Title A Flexible Class for Messy Dates

Version 1.1.0

Description Contains a set of tools for constructing and coercing
into and from the ``mdate" class.
This date class implements ISO 8601-2:2019(E) and
allows regular dates and times to be annotated
to express unspecified date or time components,
approximate or uncertain components,
ranges, and sets of dates.
The package therefore retains, represents, and reasons about data and time imprecision,
resolving to a single data/time only on demand.
This is useful for describing and analysing temporal information,
whether historical or recent, where date or time precision may vary.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 4.1)

Imports methods, lubridate, stringi

Suggests testthat (>= 3.0.0), rmarkdown, anytime, clock

URL https://globalgov.github.io/messydates/

BugReports https://github.com/globalgov/messydates/issues

Config/Needs/build roxygen2, devtools

Config/Needs/check covr, lintr, spelling

Config/Needs/website pkgdown

Config/testthat/edition 3

Config/testthat/parallel true

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author James Hollway [cre, aut, ctb] (IHEID, ORCID:
<https://orcid.org/0000-0002-8361-9647>),
Henrique Sposito [ctb] (IHEID, ORCID:
<https://orcid.org/0000-0003-3420-6085>),
Jael Tan [ctb] (IHEID, ORCID: <https://orcid.org/0000-0002-6234-9764>),
Nathan Werth [ctb]

Maintainer James Hollway <james.hollway@graduateinstitute.ch>

Repository CRAN

Date/Publication 2026-07-27 09:50:02 UTC

Contents

`battles` *Dates of battles in 2001*

Description

A dataset containing the names and dates of battles in 2001, according to Wikipedia (https://en.wikipedia.org/wiki/List_of_ba

Usage

```
battles
```

Format

A data frame with 20 rows and 5 variables:

Battle name of the battle, character

Date date or date range, a mdate class vector

Parties parties to the conflict, character

US_party is the US a party to the battle, numeric

N_actors number of actors to conflict, numeric

Examples

```
battles$Date
as.Date(battles$Date, FUN = vmin)
```

`class_mdate` *A flexible date class for messy dates*

Description

Recent extensions to standardised date notation in ISO 8601-2_2019(E) enable the recording of dates as unspecified, uncertain, approximate, and as a set or a range. These functions create and validate a new date class for R that can contain and parse these annotations. Whereas `new_messydate()` creates a new `mdate` object, and `make_messydate()` creates a new `mdate` object from one, two, or three date variables, `validate_messydate()` checks that the object is valid.

Note that the functions documented here are typically used internally, not by users; users are recommended to use `as_messydate()` to coerce dates and character strings, including historical prose, into `mdate` objects.

Usage

```
new_messydate(x = character())

validate_messydate(x)

make_messydate(..., resequence = FALSE)

## S3 method for class 'mdate'
print(x, ...)

## S3 method for class 'mdate'
format(x, ...)
```

Arguments

x	A character scalar or vector in the expected "yyyy-mm-dd" format annotated, as necessary, according to ISO 8601-2_2019(E).
...	One (yyyy-mm-dd), two (yyyy-mm-dd, yyyy-mm-dd), or three (yyyy, mm, dd) variables.
resequence	Users have the option to choose the order for ambiguous dates with or without separators (e.g. "11-01-12" or "20112112"). NULL by default. Other options include: 'dmy', 'ymd', 'mdy', 'ym', 'my' and 'interactive' If 'dmy', dates are converted from DDMMYY format for 6 digit dates, or DDMMYYYY format for 8 digit dates. If 'ymd', dates are converted from YYMMDD format for 6 digit dates, or YYYYMMDD format for 8 digit dates. If 'mdy', dates are converted from MMDDYY format for 6 digit dates or MMDDYYYY format for 8 digit dates. For these three options, ambiguous dates are converted to YY-MM-DD format for 6 digit dates, or YYYY-MM-DD format for 8 digit dates. If 'my', ambiguous 6 digit dates are converted from MM-YYYY format to YYYY-MM. If 'ym', ambiguous 6 digit dates are converted to YYYY-MM format. If 'interactive', it prompts users to select the existing component order of ambiguous dates, based on which the date is reordered into YYYY-MM-DD format and further completed to YYYY-MM-DD format if they choose to do so.

Details

If three date variables are passed to make_messydate(), function will create a single date (yyyy-mm-dd) from it. If two date variables are passed to make_messydate(), function will create a range of dates from it (yyyy-mm-dd..yyyy-mm-dd). If one date variable is passed to make_messydate(), function defaults to as_messydate().

format() returns the underlying ISO 8601-2 strings, so that mdate vectors render legibly when held in a data.frame or tibble column (which format their columns rather than printing them).

Value

Object of class mdate

Dates and times

Since v1.0.0, messydates operates with and on both dates and times. Times of day may be appended to a date using a space, e.g. 2019-03-01 14:30:00. ISO 8601-1 sec. 4.3.2 and RFC 3339 both permit a space as an alternative to the T separator more commonly seen in machine-generated timestamps; messydates uses a space for readability, though T continues to be accepted on input. Hours, minutes, and seconds are accepted (with optional fractional seconds), as are 12-hour am/pm times. Coordinated Universal Time is written with the Z designator, and other zones as a numeric offset, e.g. +02:00. Time components accept the same annotations as dates: approximate (~), uncertain (?), both (%), and unspecified (X), e.g. 2019-03-01 ~14:30 or 2019-03-01 14:XX. Because : is also used as a range separator, times are detected and protected before ranges are parsed, so 2009-01-01:2019-01-01 remains a range while 2019-03-01 14:30:00 is read as a time. A time of day may also be given on its own, with no date part, e.g. 14:30 or 2:30pm. This requires a clear time signal (a colon-separated clock or an am/pm suffix), so a bare number such as 2019 is still read as a year rather than an hour; a bare am/pm hour (2pm) is taken as an exact hour and filled to 14:00.

Imprecision annotations

Unspecified date components, such as when the day is unknown, can be represented by one or more `X`s in place of the digits. The modifier `*` is recommended to indicate that the entire time scale component value is unspecified, e.g. `X*-03-03`, however this is not implemented here. Please be explicit about the digits that are unspecified, e.g. `XXXX-03-03` expresses 3rd March in some unspecified year, whereas `2003-XX-03` expresses the 3rd of some month in 2003. If time components are not given, they are expanded to this.

Approximate date components, modified by `~`, represent an estimate whose value is asserted to be possibly correct. For example, `2003~-03-03` The degree of confidence in approximation depends on the application.

Uncertain date components, modified by `?`, represent a date component whose source is considered to be dubious and therefore not to be relied upon. An additional modifier, `%`, is used to indicate a value that is both uncertain and approximate.

Set annotations

These functions also introduce standard notation for ranges of dates. Rather than the typical R notation for ranges, `:`, ISO 8601-2_2019(E) recommends `...` This then can be applied between two time scale components to create a standard range between these dates (inclusive), e.g. `2009-01-01..2019-01-01`. But it can also be used as an affix, indicating "on or before" if used as a prefix, e.g. `..2019-01-01`, or indicating "on or after" if used as a suffix, e.g. `2009-01-01...`

And lastly, notation for sets of dates is also included. Here braces, `{}`, are used to mean "all members of the set", while brackets, `[]`, are used to mean "one member of the set".

See Also

`as_messydate()` for the full, user-facing coercion pipeline, including parsing of free text and historical prose (e.g. Roman numerals, "circa", "between ... and ...").

Examples

```
new_messydate("2012-03-03")
validate_messydate(new_messydate(c("2012-03-03", "2012-XX-03~")))
# invalid characters or missing digits raise an error
tryCatch(validate_messydate(new_messydate("2012-03-03g")),
         error = function(e) e$message)
make_messydate("2010", "10", "10")
```

`class_mduration` *A flexible duration class for messy durations*

Description

Most R packages handle duration and periods as exact time or date intervals. However, this is not possible for 'messy' dates where uncertainty or approximation might be present. The `mduration` class accounts for uncertainty and approximation in `mdate` objects to return their duration as a range of possible dates.

Non-range values (a single date, or a range collapsed to a single value) are returned unchanged. When both ends of the range carry a time of day, `approx_range` is still interpreted as a number of days, but the returned range keeps sub-day precision (e.g. `"2010-01-01 09:00..2010-01-01 17:00"`).

Usage

```
new_messyduration(x = character())

validate_messyduration(x, approx_range = 0)

make_messyduration(x, approx_range = 0)

## S3 method for class 'character'
make_messyduration(x, approx_range = 0)

## S3 method for class 'mdate'
make_messyduration(x, approx_range = 0)

## S3 method for class 'mduration'
print(x, ...)
```

Arguments

`x`	An `mdate` variable with ranges.
`approx_range`	Range to expand approximate dates, in days. If 3, for example, widens the range by 3 days on both sides, moving the start 3 days earlier and the end 3 days later; if -3, narrows the range by 3 days from both sides.
`...`	Additional arguments passed to `str()`.

Value

Object of class `mduration`

Examples

```
make_messyduration(as_messydate(c("2010-01-01..2010-12-31", "2010-01..2010-12")))
# widen (or narrow) the range at both ends
make_messyduration(as_messydate("2010-06-01..2010-06-10"), approx_range = 3)
# ranges that carry a time of day keep sub-day precision
make_messyduration(as_messydate("2010-01-01 09:00..2010-01-01 17:00"))
```

class_methods *Basic vector methods for* mdate *objects*

Description

These methods let mdate vectors behave like ordinary character vectors for subsetting, replacement, concatenation, repetition, and deduplication, while ensuring the result remains a validated mdate object.

Usage

```
## S3 method for class 'mdate'
x[..., drop = TRUE]

## S3 replacement method for class 'mdate'
x[i, ...] <- value

## S3 method for class 'mdate'
x[[...]]

## S3 replacement method for class 'mdate'
x[[i, ...]] <- value

## S3 method for class 'mdate'
c(...)

## S3 method for class 'mdate'
rep(x, ...)

## S3 method for class 'mdate'
unique(x, incomparables = FALSE, ...)

## S3 method for class 'mdate'
duplicated(x, incomparables = FALSE, ...)
```

Arguments

x	An mdate object.
drop	Included for consistency with the default [method; has no effect since mdate objects are always vectors.
i, ...	Index or indices, as for the default methods; for c(), one or more objects to concatenate (coerced to mdate first).
value	A replacement value, coerced to mdate and validated before assignment.
incomparables	Values that cannot be compared, as for the default methods; FALSE by default.

Details

unique() and duplicated() compare the annotated strings, not the sets of dates they expand to, so two values that could refer to the same date but are written differently (e.g. "2012-01" and "2012-01-01..2012-01-31") are treated as distinct.

Value

An mdate object, except for duplicated(), which returns a logical vector, and c(), which returns the (unclassed) result when called on a single object.

Examples

```
d <- as_messydate(c("2012-01-01", "2012-02-01", "2012-03-01"))
d[2]
d[2] <- "2012-02-02"
c(d, as_messydate("2012-04-01"))
rep(d, 2)
unique(as_messydate(c("2012-01-01", "2012-01-01", "2012-02-01")))
```

---

coerce_from *Coercion from* mdate *to common date classes*

Description

These functions coerce objects of mdate class to common date classes such as Date, POSIXct, and POSIXlt. Since mdate objects can hold multiple individual dates, however, an additional function must be passed as an argument so that these functions know how to resolve multiple dates into a single date.

For example, one might wish to use the earliest possible date in any ranges of dates (min), the latest possible date (max), some notion of a central tendency (mean, median, or modal), or even a random selection from among the candidate dates.

These functions then, building on expand() and the resolve functions, are particularly useful in converting back out of the mdate class for use with existing methods and models, especially for checking the robustness of results.

Usage

```
## S3 method for class 'mdate'
as.Date(x, FUN = vmin, ...)

## S3 method for class 'mdate'
as.POSIXct(x, tz = "UTC", FUN = vmin, ...)

## S3 method for class 'mdate'
as.POSIXlt(x, tz = "UTC", FUN = vmin, ...)

## S3 method for class 'mdate'
```

```
as.data.frame(x, ...)

## S3 method for class 'mdate'
as.list(x, ...)

## S3 method for class 'mdate'
as.double(x, ...)

## S4 method for signature 'mdate'
as_datetime(x, ...)
```

Arguments

x	A mdate object
FUN	A function that can be used to resolve expanded messy dates into a single date. For example, min(), max(), mean(), median(), modal(), and random(). vmin(), vmax(), vmean(), vmedian(), vmodal(), and vrandom() are the vectorised equivalents, resolving each element separately rather than summarising the whole vector.
...	Arguments passed on to the S3 generics.
tz	Character string specifying the time zone for the conversion, if required. By default "UTC" (Universal Time Coordinated), equivalent to GMT. If "" then the current time zone is used.

Details

as.Date() always drops any time of day carried by x (a calendar date has no time component); use as.POSIXct() or as.POSIXlt() to keep the time.

as.POSIXct() and as.POSIXlt() keep the time of day (defaulting to midnight if x is date-only), and honour a UTC offset if x carries one. They do not support dates before the common era; use as.Date() for those.

as.data.frame() places the (unresolved) mdate vector in a single-column data frame, as for any other vector.

as.list() splits x into a list of length-one mdate objects, one per element, without resolving any of them.

as.double() converts x to the number of days since 1970-01-01 (as for as.double(as.Date(x))), without resolving ranges, sets, or unspecified components first; it is mostly useful for already-precise dates.

{lubridate}'s as_date() and as_datetime() also accept an mdate (delegating to as.Date()/as.POSIXct() above, so the FUN resolver still applies).

Value

A date object of Date, POSIXct, or POSIXlt class

See Also

resolve_extrema(), resolve_tendency()

Other coerce: coerce_to

Examples

```
as.Date(as_messydate("2012-01"), FUN = vmin)
as.Date(as_messydate("2012-01-01"), FUN = vmean)
as.Date(as_messydate("2012-01"), FUN = vmax)
as.Date(as_messydate("2012-01"), FUN = vmedian)
as.Date(as_messydate("2012-01"), FUN = vmodal)
as.Date(as_messydate("2012-01"), FUN = vrandom)
# "1000 BC" is the astronomical year -0999 (year zero exists), whereas the
# signed ISO "-1000" below is astronomical year -1000
as.Date(as_messydate("1000 BC"), FUN = vmax)
as.Date(as_messydate("1000 BC"), FUN = vmedian)
as.Date(as_messydate(c("-1000", "2020")), FUN = vmin)
# the time of day, if any, is dropped
as.Date(as_messydate("2012-01-01 14:30"), FUN = vmin)
as.POSIXct(as_messydate("2012-01-01 14:30:00"), FUN = vmin)
as.POSIXct(as_messydate("2012-01-01 14:30:00+02:00"), FUN = vmin)
as.POSIXlt(as_messydate("2012-01-01 14:30:00"), FUN = vmin)
as.data.frame(as_messydate(c("2012-01-01", "2012-02")))
as.list(as_messydate(c("2012-01-01", "2012-02")))
as.double(as_messydate("2012-01-01"))
```

---

coerce_to *Coercion from common date classes to* mdate

Description

These methods coerce various date classes into the mdate class. They represent the main user-facing class-creating functions in the package. In addition to the typical date classes in R (Date, POSIXct, and POSIXlt), there is also a direct method for converting text or character strings to mdate. The function can also extract dates and times from text, including some historical prose conventions, though this is a work-in-progress and currently only works in English.

Usage

```
as_messydate(x, resequence = FALSE)

## S3 method for class 'Date'
as_messydate(x, resequence = FALSE)

## S3 method for class 'POSIXct'
as_messydate(x, resequence = FALSE)

## S3 method for class 'POSIXlt'
```

```
as_messydate(x, resequence = FALSE)

## S3 method for class 'character'
as_messydate(x, resequence = NULL)

## S3 method for class 'factor'
as_messydate(x, resequence = NULL)

## S3 method for class 'numeric'
as_messydate(x, resequence = NULL)

## S3 method for class 'list'
as_messydate(x, resequence = FALSE)

mdate(x, resequence = FALSE)
```

Arguments

x	A scalar or vector of a class that can be coerced into mdate, such as Date, POSIXct, POSIXlt, or character.
resequence	Users have the option to choose the order for ambiguous dates with or without separators (e.g. "11-01-12" or "20112112"). NULL by default. Other options include: 'dmy', 'ymd', 'mdy', 'ym', 'my' and 'interactive' If 'dmy', dates are converted from DDMMYY format for 6 digit dates, or DDMMYYYY format for 8 digit dates. If 'ymd', dates are converted from YYMMDD format for 6 digit dates, or YYYYMMDD format for 8 digit dates. If 'mdy', dates are converted from MMDDYY format for 6 digit dates or MMDDYYYY format for 8 digit dates. For these three options, ambiguous dates are converted to YY-MM-DD format for 6 digit dates, or YYYY-MM-DD format for 8 digit dates. If 'my', ambiguous 6 digit dates are converted from MM-YYYY format to YYYY-MM. If 'ym', ambiguous 6 digit dates are converted to YYYY-MM format. If 'interactive', it prompts users to select the existing component order of ambiguous dates, based on which the date is reordered into YYYY-MM-DD format and further completed to YYYY-MM-DD format if they choose to do so.

Details

Coercion from POSIXct and POSIXlt preserves the time of day (and UTC offset) as an ISO 8601-2 date-time. Times of exactly midnight (00:00:00) are treated as date-only, so that timezone-naive dates round-trip unchanged.

Value

A mdate class object

Functions

- as_messydate(): Core mdate class coercion function
- as_messydate(Date): Coerce from Date to mdate class

- `as_messydate(POSIXct)`: Coerce from `POSIXct` to `mdate` class
- `as_messydate(POSIXlt)`: Coerce from `POSIXlt` to `mdate` class
- `as_messydate(character)`: Coerce character date objects to `mdate` class
- `as_messydate(factor)`: Coerce factors to `mdate` class, via their labels. This is the common case when a date column has been read in with `stringsAsFactors = TRUE`.
- `as_messydate(numeric)`: Coerce numeric objects to `mdate` class
- `as_messydate(list)`: Coerce list date objects to the most concise representation of `mdate` class

Parsing prose

Beyond plain and lightly-formatted dates, `as_messydate()` recognises several conventions common in e.g. historical texts and converts them to their ISO 8601-2 equivalent before the usual parsing takes place:

- Roman numerals for a bare year, e.g. `"MDCCLXXVI"` becomes `1776`.
- Roman calendar references, i.e. the Kalends, Nones, and Ides of a named month, e.g. `"the Ides of March, 44 BC"` becomes `"-0043-03-15"`. The Nones and Ides fall later (the 7th and 15th) in March, May, July, and October, and earlier (the 5th and 13th) in other months.
- Approximate qualifiers ("around", "circa", "about", "roughly", ...) add the `~` annotation, and uncertain qualifiers ("possibly", "perhaps", "reportedly", ...) add `?`; both together add `%`, e.g. `"possibly about 1910"` becomes `"%1910"`.
- Connectives joining two days of the same month: "between the 13th and 15th" or "from the 13th to the 15th" become a range (`..`); "the 13th or the 15th" becomes a set (`{}`); and a plain "the 13th and the 15th" becomes two separate dates.
- "before"/"prior to"/"no later than" and "after"/"since"/"no earlier than" become an open range, e.g. `"before 1910"` becomes `"..1910"`. The bound may itself be any precision the parser understands, including a decade or century.
- Decades ("the 1920s" becomes `"192X"`) and centuries ("the 19th century" becomes `"18XX"`).
- A comma-separated list of dates in prose, e.g. `"13th Feb, 1977, Feb 15 1977, 1910"`, is split into separate dates (here, three): a fragment that is only a year is treated as completing the date before it.

Eras and year numbering

`{messydates}` stores years in the ISO 8601-2 (proleptic Gregorian) astronomical numbering, in which a year zero exists and equals 1 BCE, `-0001` equals 2 BCE, and so on. Historical "BC"/"BCE" prose uses the older convention that has no year zero, so it is converted on input: a historical year `N BCE` becomes the astronomical year `-(N-1)`, e.g. `"1 BCE"` becomes `"0000"`, `"2 BCE"` becomes `"-0001"`, and `"44 BCE"` becomes `"-0043"`. A year written directly in signed ISO form (e.g. `"-0044"`) is already astronomical and is left unchanged, so `as_messydate("-0044")` (astronomical year -44, i.e. 45 BCE) and `as_messydate("44 BCE")` (`"-0043"`) intentionally differ. "AD"/"CE" prose is simply dropped, the year being unchanged.

Era prose is a matter of input only: it is always removed on parsing, the era of a date in an `mdate` being carried by the sign of its year alone. Where it appears in the input, it is resolved for each year separately. A marker written before a date governs that date (`"{BC1044-03-15,BC1033}"`);

otherwise a year takes the era of the first marker written after it, so a marker given once at the end still applies to every bound of a range or set: `"200..100 BC"`, `"..200 BC"` and `"200 BC.."` are all wholly BCE, whereas `"200 BC..100 AD"` spans the two eras and gives `"-0199..0100"`.

See Also

Other coerce: `coerce_from`

Examples

```
as_messydate("2021")
as_messydate("2021-02")
as_messydate("2021-02-01")
as_messydate("01-02-2021")
as_messydate("1 February 2021")
as_messydate("First of February, two thousand and twenty-one")
as_messydate("2021-02-01?")
as_messydate("2021-02-01~")
as_messydate("2021-02-01%")
as_messydate("2021-02-01..2021-02-28")
as_messydate("{2021-02-01,2021-02-28}")
as_messydate(c("-2021", "2021 BC", "-2021-02-01"))
as_messydate(c("210201", "20210201"), resequence = "ymd")
as_messydate(c("010221", "01022021"), resequence = "dmy")
# as_messydate(c("01-02-21", "01-02-2021", "01-02-91", "01-02-1991"),
# resequence = "interactive")
# ISO 8601-2 times, with the same annotations available on time components
as_messydate("2019-03-01 14:30:00Z")
as_messydate("2019-03-01 2:30pm")
as_messydate("2019-03-01 ~14:30")
# a time of day may also be given on its own, with no date part
as_messydate("2:30pm")
as_messydate("around 2pm")
# historical prose (see the "Parsing historical prose" section below)
as_messydate("MDCCLXXVI")
as_messydate("the Ides of March, 44 BC")
as_messydate("possibly about 1910")
as_messydate("the 1920s")
as_messydate("the 19th century")
as_messydate("before 1910")
as_messydate("between the 13th and 15th of Feb, 1977")
as_messydate(list(c("2012-06-01", "2012-06-02", "2012-06-03")))
as_messydate(list(c("2012-06-01", "2012-06-02", "2012-06-03",
"{2012-06-01, 2012-06-02, 2012-06-03}", "2012-06-01", "2012-06-03")))
```

`component_annotate` *Annotates dates as censored, uncertain, or approximate*

Description

Some datasets have for example an arbitrary cut off point for start and end points, but these are often coded as precise dates when they are not necessarily the real start or end dates. This collection of functions helps annotate uncertainty and approximation to dates according to ISO2019E standards. Inaccurate start or end dates can be represented by an affix indicating "on or before", if used as a prefix (e.g. `..1816-01-01`), or indicating "on or after", if used as a suffix (e.g. `2016-12-31..`). Approximate dates are indicated by adding a tilde to year, month, or day components, as well as groups of components or whole dates to estimate values that are possibly correct (e.g. `2003-03-03~`). Day, month, or year, uncertainty can be indicated by adding a question mark to a possibly dubious date (e.g. `1916-10-10?`) or date component (e.g. `1916-?10-10`).

Usage

```
on_or_before(x)

on_or_after(x)

approximate(x, component = NULL)

uncertain(x, component = NULL)
```

Arguments

`x`	A date vector
`component`	Annotation can be added on specific date components ("year", "month" or "day"), or to groups of date components (month and day ("md"), or year and month ("ym")). This must be specified. If unspecified, annotation will be added after the date (e.g. `1916-10-10?`), indicating the whole date is uncertain or approximate. For specific date components, uncertainty or approximation is annotated to the left of the date component. E.g. for "day": `1916-10-?10` or `1916-10-~10`. For groups of date components, uncertainty or approximation is annotated to the right of the group ("ym") or to both components ("md"). E.g. for "ym": `1916-10~-10`; for "md": `1916-?10-?10`.

Details

For date-times, `component` may also be "hour", "minute", or "second" (the marker is placed to the left of that time component), or "time" (the whole time of day is marked).

Value

A `mdate` object with annotated date(s)

Functions

- `on_or_before()`: prefixes dates with ".." where start date is uncertain
- `on_or_after()`: suffixes dates with ".." where end date is uncertain
- `approximate()`: adds tildes to indicate approximate dates/date components
- `uncertain()`: adds question marks to indicate dubious dates/date components.

Examples

```
data <- data.frame(Beg = c("1816-01-01", "1916-01-01", "2016-01-01"),
  End = c("1816-12-31", "1916-12-31", "2016-12-31"))
transform(data, Beg = ifelse(Beg <= "1816-01-01",
  as.character(on_or_before(Beg)), Beg))
transform(data, End = ifelse(End >= "2016-01-01",
  as.character(on_or_after(End)), End))
transform(data, Beg = ifelse(Beg == "1916-01-01",
  as.character(approximate(Beg)), Beg))
transform(data, End = ifelse(End == "1916-12-31",
  as.character(uncertain(End)), End))
# time components can be annotated too
approximate("2019-03-01 14:30:00", "hour")
uncertain("2019-03-01 14:30:00", "second")
```

`component_extract` *Extracting components from messy dates*

Description

These functions allow the extraction of particular date components from messy dates, such as the `year()`, `month()`, `day()`, and, for date-times, `hour()`, `minute()`, `second()`, and the time zone (`tz()`).

These are methods for the same-named generics in `{lubridate}`, so they extend rather than mask them: calling e.g. `year()` on an `mdate` returns the messy-date-aware result (understanding partial precision such as `2012-06-XX`), while calling it on a `Date` or `POSIXct` still dispatches to `{lubridate}`'s own methods. This lets `{messydates}` and `{lubridate}` be loaded together, in either order, without one masking the other.

`precision()` allows for the identification of the greatest level of precision in (currently) the first element of each date.

Usage

```
## S3 method for class 'mdate'
year(x, ...)

## S3 method for class 'mdate'
month(x, ...)

## S3 method for class 'mdate'
mday(x, ...)

## S3 method for class 'mdate'
hour(x, ...)

## S3 method for class 'mdate'
minute(x, ...)
```

```
## S3 method for class 'mdate'
second(x, ...)

## S3 method for class 'mdate'
tz(x, ...)

precision(x)

## S3 method for class 'mdate'
precision(x)
```

Arguments

x	A mdate object
...	Additional arguments passed to or from other methods (accepted for compatibility with the {lubridate} generics; unused).

Details

Unlike {lubridate}'s tz(), which returns an Olson time zone name, tz.mdate() returns the ISO 8601 UTC offset *designator* carried by the date-time string ("Z" or e.g. "+02:00"), or NA when none is present.

Value

year(), month(), day(), hour(), minute(), and second() extraction return the integer for the requested component (NA where the component is absent or unspecified).

tz() returns the time zone designator or offset as a string.

precision() returns the level of greatest precision for each date.

Precision

Date precision is measured relative to the day in $1/days(x)$. That is, a date measured to the day will return a precision score of 1, a date measured to the month will return a precision score of between $1/28$ and $1/31$, and annual measures will have a precision of between $1/365$ and $1/366$. Times of day extend the same scale below the day: a date-time measured to the hour returns 24, to the minute 1440, and to the second 86400.

Examples

```
year(as_messydate(c("2012-02-03","2012","2012-02")))
month(as_messydate(c("2012-02-03","2012","2012-02")))
day(as_messydate(c("2012-02-03","2012","2012-02")))
hour(as_messydate(c("2012-02-03 14:30:00","2012-02-03")))
minute(as_messydate("2012-02-03 14:30:00"))
second(as_messydate("2012-02-03 14:30:05"))
tz(as_messydate("2012-02-03 14:30:00+02:00"))
precision(as_messydate(c("2012-02-03","2012","2012-02")))
precision(as_messydate("2012-02-03 14:30"))
```

convert_contract *Contract lists of dates into messy dates*

Description

This function operates as the opposite of `expand()`. It contracts a list of dates into the abbreviated annotation of messy dates.

Usage

```
contract(x, collapse = TRUE)
```

Arguments

`x`	A list of dates
`collapse`	Do you want ranges to be collapsed? TRUE by default. If FALSE ranges are returned in compact format.

Details

The ´contract()´ function first `expand()` 'mdate' objects to then display their most succinct representation.

Because `expand()` drops the time of day from ranges (see `?expand`), contracting a date-time range and then re-expanding it will not restore the original times; `contract()` is intended for date-level ranges, sets, and unspecified components.

Sets are returned in the notation they were given in: a `[]` ("one member of") set contracts back to `[]` rather than `{}`. This is only possible when `contract()` is passed an `mdate` (or something coercible to one), since `expand()` returns the member dates without recording which kind of set they came from. A list of dates therefore always contracts to `{}`. Note that a set whose members happen to be consecutive days contracts to a range, `..`, whichever notation it was given in.

Value

A `mdate` vector

Examples

```
d <- as_messydate(c("2001-01-01", "2001-01", "2001",
"2001-01-01..2001-02-02", "{2001-10-01,2001-10-04}",
"{2001-01,2001-02-02}", "28 BC", "-2000-01-01",
"{2001-01-01, 2001-01-02, 2001-01-03}"))
data.frame(d, contracted = contract(d))
# a full-month range collapses to a year-month by default...
contract(as_messydate("2012-06-01..2012-06-30"))
# ...unless collapse = FALSE keeps it as an explicit start..end range
contract(as_messydate("2012-06-01..2012-06-30"), collapse = FALSE)
# a '[]' set stays a '[]' set
contract(as_messydate("[2001-01-01,2001-02-02]"))
```

convert_expand *Expand messy dates to lists of dates*

Description

These functions expand on date ranges, sets of dates, and unspecified or approximate dates (annotated with '..', '', 'XX' or '~'). As these messydates may refer to several possible dates, the function "opens" these values to reveal a vector of all the possible dates implied. Imprecise dates (dates only containing information on year and/or month) are also expanded to include possible dates within that year and/or month. The function removes the annotation from dates with unreliable sources ('?'), before being expanded normally as though they were incomplete.

Usage

```
expand(x, approx_range = 0, by = "day")
```

Arguments

x	A mdate object. If not an 'mdate' object, conversion is handled first with ´as_messydate()´.
approx_range	Range to expand approximate dates, or date components, annotated with '~', by default 0. That is, removes signs for approximate dates and treats these dates as precise dates. If 3, for example, adds 3 days for day approximation, 3 months for month approximation, 3 years for year/whole date approximation, 3 years and 3 months for year-month approximation, and 3 months and 3 days for month-day approximation.
by	Granularity of enumeration, "day" by default. To avoid combinatorial explosion, ranges are enumerated at day granularity and any time-of-day components on ranges are dropped. Precise date-times (i.e. non-ranges) keep their time. Set by to a sub-day unit ("hour", "min", or "sec") to opt in to finer enumeration of precise date-time ranges.

Value

A list of dates, including all dates in each range or set.

Examples

```
d <- as_messydate(c("2008-03-25", "-2012-02-27", "2001-01?", "~2001",
"2001-01-01..2001-02-02", "{2001-01-01,2001-02-02}", "{2001-01,2001-02-02}",
"2008-XX-31", "..2002-02-03", "2001-01-03..", "28 BC"))
expand(d)
# widen an approximate day (the '~' before the day) by 3 days either side
expand(as_messydate("2001-01-~15"), approx_range = 3)
# a precise date-time is returned unchanged, keeping its time
expand(as_messydate("2012-01-01 14:30:00"))
# a date-time range drops its time by default (day granularity)...
```

```
expand(as_messydate("2019-03-01 09:00..2019-03-01 12:00"))
# ...unless a sub-day 'by' is requested
expand(as_messydate("2019-03-01 09:00..2019-03-01 12:00"), by = "hour")
```

`convert_sequence` *Sequence method for messydates*

Description

This function provides a sequence (`seq()`) method for messydates. This can be used with ranges or unspecified dates, and is particularly useful for defining a sequence of dates before the common era or between eras.

Usage

```
## S3 method for class 'mdate'
seq(from, to, by = "days", ...)
```

Arguments

`from`	A messydate or range. If 'from' is a range and 'to' is not specified, 'from' will be the minimum of the range and 'to' will be maximum.
`to`	A messydate.
`by`	Increment of the sequence. By default "days". Use a sub-day unit ("hour", "min", or "sec") for a date-time sequence.
`...`	Arguments passed to or from methods.

Details

If `from`/`to` (or `by`) carry a time of day, the sequence is generated at the requested sub-day granularity (e.g. `by = "hour"`) via `POSIXct`, and each element of the result keeps a time of day. Otherwise, dates are sequenced by calendar day (or another day-based `by`, e.g. `"week"` or `"month"`). Because years are numbered astronomically (proleptic Gregorian, with a year zero), a sequence that spans the BCE/CE boundary passes through the astronomical year zero (= 1 BCE) rather than jumping straight from `-0001` to `0001`.

Examples

```
seq(mdate("-0001-12-20"), mdate("0001-01-10"))
# a range's endpoints are used when only 'from' is given
seq(as_messydate("2012-01-01..2012-01-05"))
# date-time sequences use a sub-day 'by'
seq(as_messydate("2019-03-01 09:00"), as_messydate("2019-03-01 12:00"),
    by = "hour")
```

md_problems *Diagnose unparseable dates*

Description

Reports which elements of a vector `as_messydate()` cannot represent, and why. Where `as_messydate()` stops at the first problem, or returns `NA` for text it could not read, `md_problems()` inspects every element and returns one row per failure. This is intended for checking a column of dates before coercing it.

Usage

```
md_problems(x)
```

Arguments

`x`	A vector of dates, usually character.

Value

A data frame with one row per problematic element and the columns `index` (its position in `x`), `input` (the offending value), and `reason`. A zero-row data frame means every element can be coerced.

Examples

```
md_problems(c("2019-01-01", "2019-02-30", "2019-W12", "not a date"))
# a clean vector returns no rows
md_problems(c("2019-01-01", "2019-01"))
```

operate_arithmetic *Arithmetic operations for messydates*

Description

These operations allow users to add or subtract dates messydate objects. Messydate objects include incomplete or uncertain dates, ranges of dates, negative dates, and date sets.

Usage

```
## S3 method for class 'mdate'
e1 + e2

## S3 method for class 'mdate'
e1 - e2
```

Arguments

e1	An mdate or date object.
e2	An mdate, date, or numeric object. Must be a scalar.

Details

When e2 is a number or a string naming a unit ("day", "week", "month", "year", or, for date-times, "hour", "min"/"minute", or "sec"/"second"), e1 is shifted by that amount:

- A bare number, or a "day"/"week" unit, shifts by that many days.
- "month" and "year" shift the calendar component itself (preserving the day of month and any time of day), rolling back to the last valid day where necessary, e.g. adding a month to "2012-01-31" gives "2012-02-29" (2012 being a leap year), not an invalid "2012-02-31".
- Sub-day units, or any unit at all when e1 carries a time of day, shift the instant in seconds via POSIXct, promoting a date-only e1 to a time if the shift is sub-day.

When e2 is itself an mdate, +/- instead treat both sides as sets of dates: + returns their union (as a multiset, in the most succinct mdate notation; see also ?operate_set for %union%, which returns a plain vector of the member dates instead), and - removes the dates in e2 from e1.

Value

A messydates vector

Examples

```
d <- as_messydate(c("2008-03-25", "-2012-02-27", "2001-01?", "~2001",
"2001-01-01..2001-02-02", "{2001-01-01,2001-02-02}",
"2008-XX-31", "..2002-02-03", "2001-01-03..", "28 BC"))
data.frame(date = d, add = d + 1, subtract = d - 1)
data.frame(date = d, add = d + "1 year", subtract = d - "1 year")
as_messydate("2001-01-01") + as_messydate("2001-01-02..2001-01-04")
as_messydate("2001-01-01") + as_messydate("2001-01-03")
as_messydate("2001-01-01..2001-01-04") - as_messydate("2001-01-02")
#as_messydate("2001-01-01") - as_messydate("2001-01-03")
# calendar (month/year) arithmetic keeps the day of month and time of day
as_messydate("2012-01-31 09:00") + "1 month"
# sub-day units shift the instant
as_messydate("2012-01-01 14:30:00") + "2 hours"
```

---

operate_inequalities *Logical operations on messy dates*

Description

These operators (<, >, <=, >=) compare mdate objects with each other, or with Date/POSIXct/POSIXlt objects, by comparing the range of dates each side could represent (its minimum and maximum), rather than requiring a single, precise value on both sides. A comparison returns NA wherever the two ranges overlap and the order cannot be determined; see the examples below. For a measure of *how much* of one side precedes or follows the other, rather than a strict TRUE/FALSE/NA, see ?operate_proportional.

Usage

```
## S3 method for class 'mdate'
e1 < e2

## S3 method for class 'mdate'
e1 > e2

## S3 method for class 'mdate'
e1 <= e2

## S3 method for class 'mdate'
e1 >= e2
```

Arguments

e1, e2	mdate or other class objects

Value

A logical vector the same length as the longer of e1 and e2.

Functions

- < : tests whether the dates in the first vector precede the dates in the second vector. Returns NA when the date order can't be determined.
- > : tests whether the dates in the first vector succeed the dates in the second vector. Returns NA when the date order can't be determined.
- <= : tests whether the dates in the first vector are equal to or precede the dates in the second vector. Returns NA when the date order can't be determined.
- >= : tests whether the dates in the first vector are equal to or succeed the dates in the second vector. Returns NA when the date order can't be determined.

Examples

```
as_messydate("2012-06-02") > as.Date("2012-06-01") # TRUE
# 2012-06-XX could mean 2012-06-03, so unknown if it comes before 2012-06-02
as_messydate("2012-06-XX") < as.Date("2012-06-02") # NA
# But 2012-06-XX cannot be before 2012-06-01
as_messydate("2012-06-XX") >= as.Date("2012-06-01") # TRUE
# times of day are compared for two dates on the same day
```

```
as_messydate("2012-06-02 09:00") < as_messydate("2012-06-02 17:00") # TRUE
```

`operate_proportional` *Proportion of messy dates meeting logical test*

Description

These functions provide various proportional tests for messy date objects, complementing the strict logical comparisons in `?operate_inequalities`. Where a plain </>/etc. comparison can only return `TRUE`, `FALSE`, or `NA` for a messy (imprecise) date, these functions instead report *what proportion* of the dates implied by `e1` satisfy the comparison against `e2`, by expanding both to their full sets of possible dates first.

Usage

```
e1 %l% e2

## S3 method for class 'mdate'
e1 %l% e2

e1 %g% e2

## S3 method for class 'mdate'
e1 %g% e2

e1 %ge% e2

## S3 method for class 'mdate'
e1 %ge% e2

e1 %le% e2

## S3 method for class 'mdate'
e1 %le% e2

e1 %><% e2

## S3 method for class 'mdate'
e1 %><% e2

e1 %>=<% e2

## S3 method for class 'mdate'
e1 %>=<% e2
```

Arguments

`e1`, `e2`	`mdate` or other class objects; must be of equal length.

Details

Both kinds of set are expanded to their members, so `{}` ("all members of") and `[]` ("one member of") sets give the same proportion. For a `[]` set that proportion reads as a probability that the comparison holds, under the assumption that each candidate is equally likely, in the same way as for a range or an unspecified component. For a `{}` set it instead reads as the share of the recorded occurrences that satisfy it.

Value

A numeric vector, the same length as `e1` and `e2`, of proportions between 0 and 1.

Functions

- `%l%` : Tests proportion of dates in the first vector that precede the minimum in the second vector.
- `%g%` : Tests proportion of dates in the first vector that follow the maximum in the second vector.
- `%ge%` : Tests proportion of dates in the first vector that follow or are equal to the maximum in the second vector.
- `%le%` : Tests proportion of dates in the first vector that precede or are equal to the minimum in the second vector.
- `%><%` : Tests proportion of dates in the first vector that are between the minimum and maximum dates in the second vector.
- `%>=<%` : Tests proportion of dates in the first vector that are between the minimum and maximum dates in the second vector, inclusive.

Examples

```
as_messydate("2012-06") < as.Date("2012-06-02")
as_messydate("2012-06") %l% as_messydate("2012-06-02")
as_messydate("2012-06") > as.Date("2012-06-02")
as_messydate("2012-06") %g% as_messydate("2012-06-02")
as_messydate("2012-06") >= as.Date("2012-06-02")
as_messydate("2012-06") %ge% as_messydate("2012-06-02")
as_messydate("2012-06") <= as.Date("2012-06-02")
as_messydate("2012-06") %le% "2012-06-02"
as_messydate("2012-06") %><% as_messydate("2012-06-15..2012-07-15")
as_messydate("2012-06") %>=<% as_messydate("2012-06-15..2012-07-15")
```

`operate_set` *Set operations for messy dates*

Description

Performs intersection (`%intersect%`) and union (`%union%`) on the dates or date-times implied by messy date class objects, treating each as the (day-granularity) set of dates it expands to. Both return a plain character vector of the individual member dates.

For a union that instead returns an `mdate` object in its most succinct (contracted) notation, e.g. a range rather than a list of every day within it, use `+` (see `?operate_arithmetic`) instead.

Usage

```
e1 %intersect% e2

## S3 method for class 'mdate'
e1 %intersect% e2

e1 %union% e2

## S3 method for class 'mdate'
e1 %union% e2
```

Arguments

`e1`, `e2`	Messy date or other class objects

Details

Both kinds of set expand to the same members, so these operators return the same dates for `{}` ("all members of") and `[]` ("one member of") alike. What differs is the reading: for a `{}` set the result lists dates that are shared with (or covered by) the other operand, whereas for a `[]` set it lists the *candidates* that remain possible, so an empty intersection rules the other operand out rather than showing no overlap.

Value

A vector of the same mode for `%intersect%`, or a common mode for `%union%`.

Functions

- `%intersect%` : Find intersection of sets of messy dates
- `%union%` : Find union of sets of messy dates

Examples

```
as_messydate("2012-01-01..2012-01-20") %intersect% as_messydate("2012-01")
as_messydate("2012-01-01..2012-01-20") %union% as_messydate("2012-01")
```

operate_statements *Logical statements on messy dates*

Description

These functions provide various logical statements about messy date objects.

Usage

```
is_messydate(x)

is_intersecting(x, y)

is_subset(x, y)

is_similar(x, y)

is_precise(x)

is_uncertain(x)

is_approximate(x)

is_bce(x)
```

Arguments

x, y	mdate or other class objects

Value

A logical vector the same length as the mdate passed.

Functions

- is_messydate(): tests whether the object inherits the mdate class. If more rigorous validation is required, see validate_messydate().
- is_intersecting(): tests whether there is any intersection between two messy dates, leveraging intersect().
- is_subset(): tests whether one or more messy date can be found within a messy date range or set.
- is_similar(): tests whether two dates contain similar components. This can be useful for identifying dates that may be typos of one another.
- is_precise(): tests whether a date (or date-time) is precise, i.e. a full "yyyy-mm-dd", optionally with a time of day (down to the hour, minute, or second) and time zone. Non-precise dates contain markers that they are approximate (i.e. ~), unreliable (i.e. ?), are incomplete

(e.g. year or year-month only, or a date with an unspecified `X` component), or are date ranges and sets.

- `is_uncertain()`: tests whether a date is uncertain (i.e. contains ?).
- `is_approximate()`: tests whether a date is approximate (i.e. contains ~).
- `is_bce()`: tests whether one or more messy dates are found before the common era, i.e. carry a negative (astronomical) year. Note that astronomical year zero (`"0000"`, historically 1 BCE) is not sign-negative and so returns `FALSE`; it is handled natively as a non-negative year throughout, which R's proleptic Gregorian `Date` supports directly.

Examples

```
is_messydate(as_messydate("2012-01-01"))
is_messydate(as.Date("2012-01-01"))
is_intersecting(as_messydate("2012-01"),
as_messydate("2012-01-01..2012-02-22"))
is_intersecting(as_messydate("2012-01"),
as_messydate("2012-02-01..2012-02-22"))
is_subset(as_messydate("2012-01-01"), as_messydate("2012-01"))
is_subset(as_messydate("2012-01-01..2012-01-03"), as_messydate("2012-01"))
is_subset(as_messydate("2012-01-01"), as_messydate("2012-02"))
is_similar(as_messydate("2012-06-02"), as_messydate("2012-02-06"))
is_similar(as_messydate("2012-06-22"), as_messydate("2012-02-06"))
is_precise(as_messydate(c("2012-06-02", "2012-06")))
is_precise(as_messydate(c("2012-06-02 14:30:00", "2012-06-02 ~14")))
is_uncertain(as_messydate(c("2012-06-02", "2012-06-02?")))
is_approximate(as_messydate(c("2012-06-02~", "2012-06-02")))
is_bce(as_messydate(c("2012-06-02", "-2012-06-02")))
```

`resolve_extrema` *Resolves messy dates into an extrema*

Description

This collection of S3 methods 'resolve' messy dates into a single date according to some explicit bias, such as returning the minimum or maximum date, the mean, median, or modal date, or a random date from among the possible resolutions for each messy date. If the date is not 'messy' (i.e. has no annotations) then just that precise date is returned. This can be useful for various descriptive or inferential projects.

Usage

```
vmin(..., na.rm = FALSE)

## S3 method for class 'mdate'
vmin(..., na.rm = FALSE)

## S3 method for class 'mdate'
min(..., na.rm = FALSE)
```

```
vmax(..., na.rm = FALSE)

## S3 method for class 'mdate'
vmax(..., na.rm = FALSE)

## S3 method for class 'mdate'
max(..., na.rm = FALSE)
```

Arguments

...	a mdate object
na.rm	Should NAs be removed? FALSE by default.

Details

vmin()/min() and vmax()/max() work directly on the annotated string (dropping ~/?/% and, for vmax()/max(), filling in unspecified X components) rather than via expand(), which makes them considerably faster than resolving through the full expanded set. min() and max() further resolve a vector to a single, overall extremum (matching the usual behaviour of these generics), while vmin() and vmax() resolve each element separately.

Dates that carry a time of day are already precise and so pass through these functions unchanged, keeping their time; a time of day plays no further role in choosing the minimum or maximum.

Value

A single scalar or vector of dates

Examples

```
d <- as_messydate(c("2008-03-25", "?2012-02-27", "2001-01?", "2001~",
  "2001-01-01..2001-02-02", "{2001-01-01,2001-02-02}",
  "{2001-01,2001-02-02}", "2008-XX-31", "-0050-01-01"))
d
# a precise date-time is returned unchanged
vmin(as_messydate("2012-01-01 14:30:00"))
vmin(d)
min(d)
vmax(d)
max(d)
```

---

resolve_tendency *Resolves messy dates into a central tendency*

Description

These functions resolve messydates by their central tendency. While the functions `mean()`, `median()`, and `modal()` expand *all* elements of the vector into one combined set of dates and summarise it to a single value (matching the usual behaviour of these generics), the `v*()` versions resolve each element separately and so return a vector of the same length as the input.

Usage

```
## S3 method for class 'mdate'
median(..., na.rm = TRUE)

vmedian(..., na.rm = TRUE)

## S3 method for class 'mdate'
vmedian(..., na.rm = TRUE)

## S3 method for class 'mdate'
mean(..., trim = 0, na.rm = TRUE)

vmean(..., na.rm = TRUE)

## S3 method for class 'mdate'
vmean(..., trim = 0, na.rm = TRUE)

modal(..., na.rm = TRUE)

## S3 method for class 'mdate'
modal(..., na.rm = TRUE)

vmodal(..., na.rm = TRUE)

## S3 method for class 'mdate'
vmodal(..., na.rm = TRUE)

random(..., na.rm = TRUE)

## S3 method for class 'mdate'
random(..., na.rm = TRUE)

vrandom(..., na.rm = TRUE)

## S3 method for class 'mdate'
vrandom(..., na.rm = TRUE)
```

Arguments

`...`	a mdate object
`na.rm`	Should NAs be removed? FALSE by default.

`trim` the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.

Details

All of these functions work by calling `expand()` to enumerate the dates or date-times consistent with each messy value, then summarising that expanded set. For `median()` and `mean()`, an even number of expanded values is resolved by averaging the two middle values (via `POSIXct` when a time of day is present, or `Date` otherwise); an odd number simply returns the middle value.

Both kinds of set are expanded to their members and summarised the same way, but the result means different things for each. For a `{}` ("all members of") set, which records that something happened on several dates, the central tendency describes where those occurrences sit. For a `[]` ("one member of") set, which records that something happened on exactly one of those dates but not which, the central tendency is a point estimate of that single unknown date, as it is for a range or an unspecified component. `vrandom()` draws a member uniformly in either case, which for a `[]` set is a draw from the candidates themselves.

Averaging across the BCE/CE boundary, or between two BCE dates, is not currently supported: `median()` falls back to the earlier of the two middle values in that case, and `mean()`/`vmean()` may be unreliable for solely negative-year inputs (a documented limitation, not a supported feature).

Examples

```
d <- as_messydate(c("2008-03-25", "?2012-02-27", "2001-01?", "2001~",
  "2001-01-01..2001-02-02", "{2001-01-01,2001-02-02}",
  "{2001-01,2001-02-02}", "2008-XX-31", "-0050-01-01"))
d
# the time of day is honoured when averaging precise date-times
r <- as_messydate(c("2012-06-01 09:00", "2012-06-01 17:00"))
median(r)
mean(r)
median(d)
vmedian(d)
mean(d)
vmean(d)
modal(d)
vmodal(d)
random(d)
vrandom(d)
```

Index