Package 'datey'

November 12, 2026

Title Exact Date and Duration Arithmetic on an Annual Grid

Version 0.1.3

Description Standardised mapping of dates onto a discrete annual grid, together with exact date and duration arithmetic. This matters when the primary unit is years but the input data uses dates. Examples are actuarial mortality experience analysis and valuation of life assurance and annuities, for which mortality rates are defined per year but experience and valuation data are typically defined using dates.

License MIT + file LICENSE

URL https://r-datey.logmu.org/, https://github.com/logmu-org/r-datey

BugReports https://github.com/logmu-org/r-datey/issues

Depends R (>= 4.0.0)

Suggests knitr, pillar, rmarkdown, testthat (>= 3.0.0)

LinkingTo cpp11

VignetteBuilder knitr

Config/roxygen2/markdown TRUE

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

NeedsCompilation yes

Author Tim Gordon [aut, cre] (ORCID: <https://orcid.org/0009-0006-3087-4384>)

Maintainer Tim Gordon <tim@timgord.com>

Repository CRAN

Date/Publication 2026-11-12 15:20:10 UTC

Contents

all_of_time . . . 2
anniversary_days . . . 3
as_years_datey . . . 4
as_years_durationy . . . 5
combine . . . 6
datey . . . 7
datey_components . . . 11
datey_interval . . . 12
durationy . . . 14
integer_constants . . . 16
interval_includes . . . 17
interval_nature . . . 18
interval_properties . . . 20
is_leap_year . . . 21
is_NA . . . 22
is_type . . . 24
is_xxx_day . . . 25
max_min . . . 26
mean . . . 27
NAs . . . 28
ops . . . 28
replicate . . . 31
seq . . . 32
subset . . . 32
text_from_datey . . . 34
text_from_datey_interval . . . 35
text_from_durationy . . . 36

Index **38**

`all_of_time` *'All of time' — the maximum valid* `datey_interval`

Description

`all_of_time` is the `datey_interval` [1000-01-01.0, 3000-01-01.0), spanning the full valid date range. It is referred to as 'all of time' throughout the **datey** documentation.

It is the value produced by `datey_interval(TRUE)` and is used when a `datey_interval` is intersected with a logical `TRUE` via `&`.

Usage

```
all_of_time
```

Value

A scalar `datey_interval` spanning the full valid date range, [1000-01-01.0, 3000-01-01.0).

See Also

datey_interval, ops

Examples

```
all_of_time
```

`anniversary_days` *Anniversary days*

Description

The position of a date relative to the start of its calendar year depends on whether that calendar year is a leap year or not. This means that in **datey** (and reality!) the start of 2020-04-01 plus 1 year is not equal to the start of 2021-04-01.

This is by design and is usually precisely what's required. But this behaviour can be awkward when anniversaries are the focus and, fictitious as it may be, the position of a date in a year is required to be the same regardless of whether the year is or is not a leap year.

To help with such cases, **datey** also provides anniversary days, which assume every year has 366 days but that 29 February is just under a quarter of the length of the other days. This convenient fiction ensures that anniversaries always occur at the same point in the year.

These functions are intended for special cases, e.g. when anniversaries are used to define datasets. *For standard calculations, the normal approach is recommended.*

Usage

```
start_anniversary_day_to_ymd(x)

start_anniversary_day(year, month, day, strict = TRUE)

is_start_anniversary_day(x)
```

Arguments

`x`	The `datey` to test or deconstruct.
`year`, `month`, `day`	The year (1000 to 3000), month (1 to 12) and day (1 to 31). Note that 29 February is valid for all years.
`strict`	How non-compliant *non-NA* inputs should be handled. If `strict` is `TRUE` – the default – then execution is stopped. If `strict` is `FALSE` then `NA` is returned.

Value

`start_anniversary_day()` returns a `datey` the same length as its arguments.

`is_start_anniversary_day()` returns a `logical` the same length as its argument.

`start_anniversary_day_to_ymd()` returns a list of integer vector `year`, integer vector `month`, and integer vector `day`, all with the same length.

If the `datey` is not the start of an anniversary day then all components are `NA`.

See Also

datey, datey_components, is_xxx_day

Examples

```
# Adding whole years to the start of a day can leave the day boundary,
# because days in a leap year are shorter:
t <- start_day(2020, 4, 1) + durationy(0:4)
t
is_start_day(t)

# Anniversary days stay on the anniversary:
anniversary <- start_anniversary_day(2020, 4, 1) + durationy(0:4)
is_start_anniversary_day(anniversary)
as.data.frame(start_anniversary_day_to_ymd(anniversary))

# 29 February is an anniversary day in every year:
leap_day <- start_anniversary_day(2020, 2, 29) + durationy(0:4)
as.data.frame(start_anniversary_day_to_ymd(leap_day))

# The cost is that anniversary days rarely match calendar days. Indeed, in
# 2020, a leap year, the 1 April anniversary day starts on 31 March:
anniversary[1:2]

# Anniversary day deconstruction is NA for non-anniversary days:
as.data.frame(start_anniversary_day_to_ymd(start_day(2020, 4, 1)))
```

`as_years_datey` *Convert a* `datey` *to calendar years (including fractional part)*

Description

Converts a `datey` to calendar years, including a fractional part that represents the proportion of the calendar year that has elapsed.

For example, the middle of 2000-10-01 is precisely three-quarters through the (leap) year 2000 and so `as.double(mid_day(2000,10,1))` results in `2000.75`.

`as.numeric()` is the same as `as.double()`.

`as.integer()` gives the calendar year as an `integer`, e.g. `as.integer(datey(2000.75))` is `2000`. It is also the case that if `x` is a `datey` then `as.integer(x)` is the same as `as.integer(as.double(x))`.

Usage

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

## S3 method for class 'datey'
as.integer(x, ...)
```

Arguments

x	The datey to convert to years.
...	Not used.

Value

A vector of double.

See Also

datey, as_years_durationy, ops

Examples

```
t <- datey(2000.75)
t                   # 2000-10-01.5
as.double(t)        # 2000.75
as.numeric(t)       # 2000.75
as.integer(t)       # 2000
identical(as.integer(t), 2000L) # TRUE
```

---

as_years_durationy *Convert a* durationy *to duration in years*

Description

Converts a durationy to its duration measured in years.

as.numeric() is the same as as.double().

as.integer() obtains the integer part as an integer, e.g. as.integer(durationy(1.75)) is 1 and as.integer(durationy(-1.75)) is -1 (i.e. rounding towards 0). It is also the case that if x is a durationy then as.integer(x) is the same as as.integer(as.double(x)).

Usage

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

## S3 method for class 'durationy'
as.integer(x, ...)
```

Arguments

x	The duration to convert to years.
...	Not used.

Value

A vector of double.

See Also

duration, as_years_datey, ops

Examples

```
d <- duration(1.75)
d                   # 1.75 yr
as.double(d)        # 1.75
as.numeric(d)       # 1.75
as.integer(d)       # 1
as.integer(-d)      # -1
identical(as.integer(d), 1L) # TRUE
```

---

combine *Combine multiple* datey, duration *or* datey_interval *vectors*

Description

Combines (flattens) datey, duration or datey_interval into a single vector.

All arguments must have the same class, i.e. all dateys, all durations or all datey_intervals.

If the first element in c(...) is not a datey, duration or datey_interval then this method will not be called. For instance, c(NA, datey("2000-01-01.0")) results in c(NA_integer_, 1068720000L).

Usage

```
## S3 method for class 'datey'
c(..., recursive = FALSE)

## S3 method for class 'duration'
c(..., recursive = FALSE)

## S3 method for class 'datey_interval'
c(..., recursive = FALSE)
```

Arguments

...	The items to combine.
recursive	Unused.

Value

`c()` returns a `datey`, `durationy` or `datey_interval` depending on the first argument.

See Also

datey, durationy, datey_interval, subset

Examples

```
c(datey(2000:2019), datey("2020-01-01.0"))
```

`datey` *Create a* `datey`

Description

To create a `datey` use one of the following:

- `start_day()` and `end_day()` are the points in time at the *start* and *end* of the day respectively.
- `mid_day()` is the the *middle* of the day, commonly used to represent a point in time *during* the day.
- `datey()` is the underlying S3 generic function. `start_day()`, `mid_day()` and `end_day()` call through to `datey` with an explicit `day_fraction` or `0`, `0.5` and `1` respectively.

The generic types are as follows:

- `double` and `integer`. These are interpreted as calendar year, optionally with a fractional part in the case of `double`.

  Valid years are from 1000 to 3000 (although the only legal date in 3000 is the start of 3000-01-01).

  Either

  (a) `month` and `day` (and for `datey()`, `day_fraction`) parameters are all provided, in which case `x` must be integral, or

  (b) none of those parameters are provided, in which case `x` is interpreted as a fractional calendar year and rounded to the nearest 1 / 534360 of a year (Banker's rounding). This unit is called a *click* and is the resolution of all **datey** arithmetic. For instance, `datey(2000.5)` means halfway through the year 2000.
- `Date`. This base R date type is interpreted strictly. (It is possible to end up with an unintentionally fractional underlying value, e.g. by taking a mean of `Dates`.) A `day_fraction` argument is always required.
- `POSIXct` and `POSIXlt`. How these base R date-time types are interpreted depends on whether a `day_fraction` is provided.

  If `day_fraction` *is* provided then these are interpreted strictly using the date component only – the time component is ignored completely.

  If `day_fraction` is *not* provided then the day fraction is determined using the hours, minutes, and seconds. For instance, `datey(as.POSIXct("2000-03-21 12:00"))` means the *middle* of 2000-03-21.

- `character`. How text is parsed depends on whether a `day_fraction` is provided.

  If `day_fraction` *is* provided then the text must be in ISO 8601 extended format, i.e. `YYYY-MM-DD`.

  If `day_fraction` is *not* provided then the text must be formatted as `YYYY-MM-DD[.F..]`., where `[.F...]` is the optional day fraction. This means that e.g. `"2000-01-01"` represents the *start* of 1 January 2000.

  If `blank_is_NA` is `TRUE` then blanks are treated as `NA` (regardless of `strict`).

- `datey`. This is interpreted as is but with the optional `day_fraction` override. Note that a `day_fraction` of 1 will add a day to a day boundary, *even if it was originally defined as an end day*.

The lengths of vector arguments must be multiples of each other.

Beware that `end_day()` will add a day to a `datey` that is already on a day boundary, *even if it was originally defined as an end day*.

There is no `is_end_day()` predicate: end days are stored identically to the start of the following day, so `is_start_day()` is the correct test.

NA arguments *of the appropriate type* result in `NA_datey_` – they do not stop execution (regardless of `strict`). Note that `NA` is `logical` and therefore it *will* cause an error.

Usage

```
datey(x, ...)

## Default S3 method:
datey(x, ...)

## S3 method for class 'datey'
datey(x, day_fraction = NULL, strict = TRUE, ...)

## S3 method for class 'integer'
datey(x, month = NULL, day = NULL, day_fraction = NULL, strict = TRUE, ...)

## S3 method for class 'double'
datey(x, month = NULL, day = NULL, day_fraction = NULL, strict = TRUE, ...)

## S3 method for class 'Date'
datey(x, day_fraction, strict = TRUE, ...)

## S3 method for class 'POSIXct'
datey(x, day_fraction = NULL, strict = TRUE, ...)

## S3 method for class 'POSIXlt'
datey(x, day_fraction = NULL, strict = TRUE, ...)

## S3 method for class 'character'
datey(x, day_fraction = NULL, strict = TRUE, blank_is_NA = FALSE, ...)

start_day(x, month = NULL, day = NULL, strict = TRUE, blank_is_NA = FALSE)
```

```
mid_day(x, month = NULL, day = NULL, strict = TRUE, blank_is_NA = FALSE)

end_day(x, month = NULL, day = NULL, strict = TRUE, blank_is_NA = FALSE)
```

Arguments

x	The argument to convert to a datey.
...	Not used.
day_fraction	The day_fraction override. Defaults to NULL except for the Date type, in which case it must always be provided. If day_fraction *is* provided (which is implicitly the case for start_day(), mid_day() and end_day()) then x is used solely to derive the calendar year, month and day, while day_fraction provides the position in the day. day_fraction must lie in the inclusive interval [0,1], with 0 meaning the start of the day, 0.5 meaning the middle of the day, and 1 meaning the end of the day (which is identical to the start of the next day). For text this means that there should be *no* trailing fraction, e.g. start_day("2020-01-01"). If day_fraction is *not* provided then x is used to derive both the calendar year, month, day *and* the day fraction.
strict	How non-compliant *non-NA* inputs should be handled. If strict is TRUE – the default – then execution is stopped. If strict is FALSE then NA is returned.
month, day	The month (1–12) and day (1–31). Valid only for for numeric x. If one of month or day is provided then both must be and, for datey(), day_fraction must also be provided.
blank_is_NA	Whether "" should be treated as NA (regardless of strict). When x is "" then if blank_is_NA is TRUE then "" results in NA, otherwise execution is stopped. Defaults to FALSE. Valid only for for character x.

Value

A vector of datey.

See Also

anniversary_days, durationy, datey_interval, text_from_datey, as_years_datey, datey_components, is_xxx_day, is_leap_year, is_NA, ops, vignette("why-datey", package = "datey") for the annual-grid design, vignette("datey", package = "datey") for a worked introduction

Examples

```
start_day(2001, 2, 3)
mid_day(2001, 2, 3)
end_day(2001, 2, 3)

# Must specify month and day for a numeric if day_fraction is provided
# implicitly or explicitly:
try(start_day(2001))
```

```
try(mid_day(2001))
try(end_day(2001))
try(datey(2001, day_fraction = 0))

datey(2000) # Start of a year
datey(2000.5) # Middle of a leap year
datey(2001.5) # Middle of a non-leap year

# Convert base R date
r_date <- as.Date("2001-02-03")
c(start_day(r_date), mid_day(r_date), end_day(r_date))
try(datey(r_date)) # Must specify day_fraction for a `Date`

# Convert base R datetime
c_date <- as.POSIXct("2001-02-03 12:00:00") # Midday!
c(start_day(c_date), mid_day(c_date), end_day(c_date))
# An R datetime implies a position within a day:
datey(c_date) # 2001-02-03.5

# Use `strict` to control error behaviour for invalid years:
try(end_day(0999, 12, 31))
try(datey(3000.1))
end_day(0999, 12, 31, strict = FALSE)
datey(3000.1, strict = FALSE)

# NAs are passed through regardless of `strict`
# (provided they are numeric)
end_day(NA_real_, 12, 31, strict = TRUE)
datey(NA_real_, strict = FALSE)

# Text:
start_day("2001-02-03")
mid_day("2001-02-03")
end_day("2001-02-03")
datey("2001-02-03")
datey("2001-02-03.0")
datey("2001-02-03", day_fraction = 0)
datey("2001-02-03.5")
datey("2001-02-03", day_fraction = 0.5)

# Text round trips:
t <- datey(2001.234)
identical(t, datey(as.character(t))) # TRUE

# Day fraction cannot be present
# both in the text and as an argument
# implicitly or explicitly:
try(start_day("2001-02-03.0"))
try(datey("2001-02-03.0", day_fraction = 0))

# Handling blanks:
try(start_day(""))
start_day("", blank_is_NA = TRUE)
```

```
# Invalids:
try(mid_day("abc"))
try(mid_day("0999-01-01"))
end_day("abc", strict = FALSE) # NA
end_day("0999-01-01", strict = FALSE) # NA
```

---

datey_components *Get year, month, day or day_fraction breakdown of a* datey

---

Description

To extract the year, month, day or day_fraction breakdown of a datey, use either

- the list-like syntax $year, $month, $day or $day_fraction direct, or
- if you need several components at once, to_ymdf(), which returns an actual list of year, month, day and day_fraction.

In this breakdown,

- year is an integer in [1000,3000],
- month is an integer in [1,12],
- day is an integer in [1,N], where N is the number of days in the month specified by year and month, and
- day_fraction is a double in [0,1) representing the fraction of the day, where e.g. 0 means the start and 0.5 means the middle of the day.

If the datey was constructed using end_day or day_fraction = 1 then to_ymdf() will return the *start* of the *next* day with day_fraction = 0.

Usage

```
to_ymdf(x)

## S3 method for class 'datey'
x$name
```

Arguments

x	The datey to be deconstructed.
name	The name of the component for the list-like syntax. Must be year, month, day or day_fraction.

Value

to_ymdf() returns a list of integer vector year, integer vector month, integer vector day, and double vector day_fraction, all with the same length. The list-like syntax returns these components individually.

See Also

datey, text_from_datey, anniversary_days

Examples

```
t <- datey(2001, 2, 3, 0.5)
t
to_ymdf(t)
t$year
t$month
t$day
t$day_fraction
```

`datey_interval` *Create a* `datey_interval`

Description

Creates a `datey_interval`, a closed-open ('clopen') interval [`start`, `end`).

A `datey` `t` in included in the interval if `start` <= t < `end`.

There are two syntaxes to create a `datey_interval` from `start` and `end`:

- operator: `start %to% end`
- function: `datey_interval(start, end)`

in which `start` and `end` are `datey` or numeric (interpreted as years). These are equivalent other than `strict` is always on for the operator version. The lengths of vector arguments must be multiples of each other.

A `datey_interval` can also be created from `logical`, mapping

- `TRUE` to [1000,3000), which is referred to as 'all of time' in **datey** documentation, and
- `FALSE` and `NA` to `NA_datey_interval_`.

Arguments *of the correct type* but which are NA result in `NA_datey_interval_` – they do not stop execution (regardless of `strict`).

Common operations on intervals are

- testing for *inclusion*, i.e. with an interval includes a date – `interval %includes% t`, and
- obtaining the *intersection*, which uses the & operator – `interval_a & interval b`.

Usage

```
datey_interval(x, ...)

## Default S3 method:
datey_interval(x, ...)

## S3 method for class 'datey_interval'
datey_interval(x, ...)

## S3 method for class 'logical'
datey_interval(x, ...)

## S3 method for class 'datey'
datey_interval(x, end, strict = TRUE, ...)

## S3 method for class 'double'
datey_interval(x, end, strict = TRUE, ...)

## S3 method for class 'integer'
datey_interval(x, end, strict = TRUE, ...)

start %to% end
```

Arguments

x	Argument to S3 method `datey_interval()`. If `x` is a `datey` then it represents the start (inclusive) of an interval and an `end` argument is also required. If `x` is a `logical` then it is mapped to 'all of time' or `NA_datey_interval_`.
...	Not used.
strict	How invalid *non-NA* `datey` inputs should be handled. If `strict` is `TRUE` – the default – then execution is stopped. If `strict` is `FALSE` then `NA` is returned.
start, end	The start (inclusive) and end of the interval (exclusive). These can be any type that is convertible to a `datey`.

Value

A vector of `datey_interval`.

See Also

interval_properties, interval_nature, interval_includes, all_of_time, duration, ops, is_NA, `vignette("datey", package = "datey")` for a worked introduction

Examples

```
start <- datey(2000)
end <- datey(2001)
interval <- start %to% end
```

```
interval  # [2000-01-01.0, 2001-01-01.0)

# Alternative functional syntax:
identical(interval, datey_interval(start, end)) # TRUE

# Can use numeric arguments:
2000 %to% 2001  # [2000-01-01.0, 2001-01-01.0)

# Can use vector arguments:
2000 %to% 2001:2003 # Vector of 3 intervals

# Logical values are mapped to 'all of time' or `NA_datey_interval_`:
datey_interval(c(TRUE, FALSE, NA)) # [1000-01-01.0, 3000-01-01.0) <NA> <NA>

# Test for inclusion in [start, end):
interval %includes% mid_day(1999, 12, 31) # FALSE
interval %includes% start                 # TRUE -- start *is* included
interval %includes% datey(2000.5)         # TRUE
interval %includes% end                   # FALSE -- end is *not* included

# Obtain the intersection of two intervals
interval2 <- start_day(2000, 12, 1) %to% 2010
interval & interval2  # [2000-12-01.0, 2001-01-01.0)
```

`durationy` *Create a* `durationy` *from an annual duration*

Description

`durationy()` to create a `durationy` from the following types:

- `integer`. The value is interpreted as the specified number of years.
- `double`. The value is interpreted as the specified number of years, rounded to fixed precision of a `durationy`. This means that `durationy(0.5)` is precise but `durationy(0.01)` is not.
- `datey_interval`. The duration of the interval *provided it is proper* (i.e. start <= end). If the interval is improper then the result is `NA_durationy_`. When `x` is a `datey_interval` then `x$duration` is identical to `durationy(x)`. (`strict` is ignored.)
- `character`. Valid text is of the form `[S]...Y[.F...][ U...]` where:
  - `[S]` is an optional plus or a minus sign, i.e. one of '+' (U+002B), true minus (U+2212) or ASCII hyphen-minus '-' (U+002D).
  - `...Y` is number of whole years (leading zeros allowed).
  - `[.F...]` is an optional fractional part of year, including '.' to represent the decimal point.
  - `[ U...]` is the unit name for one year preceded by a space if the unit name is not blank. The unit name cannot be longer than 20 UTF-8 bytes or contain control characters.

  If `blank_is_NA` is `TRUE` then blanks are treated as `NA`. If `strict` is `TRUE` (the default) then non-compliant text will stop execution. If the text is NA then NA is returned. This is the same format as produced by `as.character.durationy()`.

- durationy. Value is passed through unchanged.

NA arguments *of the appropriate type* result in NA_durationy_ – they do not stop execution (regardless of strict). Note that NA is logical and therefore it *will* cause an error.

Usage

```
durationy(x, ...)

## Default S3 method:
durationy(x, ...)

## S3 method for class 'durationy'
durationy(x, ...)

## S3 method for class 'integer'
durationy(x, strict = TRUE, ...)

## S3 method for class 'double'
durationy(x, strict = TRUE, ...)

## S3 method for class 'datey_interval'
durationy(x, ...)

## S3 method for class 'character'
durationy(x, strict = TRUE, blank_is_NA = FALSE, year_unit = "yr", ...)
```

Arguments

x	The argument to convert to a durationy.
...	Not used.
strict	How non-compliant non-NA x, e.g. years greater than 2000 in magnitude or invalid text, should be handled. If strict is TRUE – the default – then execution is stopped. If strict is FALSE then NA is returned.
blank_is_NA	Whether blanks should be treated as NA. Defaults to FALSE.
year_unit	The year unit name to expect. If not blank then the value is expected to be followed by a space and this unit text. Cannot be more than 20 characters (UTF-8 bytes) or contain control characters. Defaults to "yr".

Value

A vector of durationy.

See Also

datey, datey_interval, text_from_durationy, as_years_durationy, ops, is_NA, vignette("why-datey", package = "datey") for the annual-grid design, vignette("datey", package = "datey") for a worked introduction

Examples

```
durationy(1)    # 1 yr
durationy(0.5)  # 0.5 yr
durationy(-2.3) # -2.3 yr
durationy(2001 %to% 2002) # 1 yr
durationy(2002 %to% 2001) # `NA_durationy_` because interval is improper

# NA:
durationy(NA_real_)
try(durationy(NA)) # NA is logical, not numeric

# Invalid durations:
try(durationy(3000.1)) # default strict = TRUE
durationy(3000.1, strict = FALSE)

# Text:
durationy("10 yr")
durationy("+10 yr")
durationy("-10 yr")
durationy("10", year_unit = "")
durationy("10 a", year_unit = "a")

# Text round trips:
d <- durationy(1.234)
identical(d, durationy(as.character(d))) # TRUE

# Handling blank text:
try(durationy(""))
durationy("", blank_is_NA = TRUE)

# Invalid text:
try(durationy("abc"))
try(durationy("2000.000001 yr"))
durationy("abc", strict = FALSE) # NA
durationy("2000.000001 yr", strict = FALSE) # NA
durationy("2000.000000 yr") # This is valid
```

`integer_constants` *Integer constants*

Description

The following integer constants may make code clearer.

Constant	Value	Meaning
`valid_years_start`	`1000L`	The first valid calendar year for a `datey`.
`valid_years_end`	`3000L`	The final valid calendar year for a `datey` (noting that only the start of this year is val
`valid_duration_years_max`	`2000L`	The maximum valid duration in years for a `durationy`.

Usage

```
valid_years_start

valid_years_end

valid_duration_years_max
```

Value

Each of these constants is a scalar `integer` giving one of the boundaries of the valid `datey` or `durationy` range described above.

See Also

is_NA, NAs, datey, all_of_time

Examples

```
datey(valid_years_start - 0.001, strict = FALSE)
datey(valid_years_start)
datey(valid_years_end)
datey(valid_years_end + 0.001, strict = FALSE)
durationy(valid_duration_years_max)
durationy(valid_duration_years_max + 0.001, strict = FALSE)
```

---

interval_includes *Whether a* `datey_interval` *includes a* `datey`

Description

Test whether a `datey_interval`, $[a, b)$, includes a `datey` t, i.e. $a \leq t$ and $t < b$.

The `%includes%` operator is syntactic sugar for `interval_includes()`.

An NA interval is treated as empty and an NA date is treated as not being in any interval, so these methods are guaranteed to return `TRUE` or `FALSE`.

Usage

```
interval_includes(interval, value)

interval %includes% value
```

Arguments

`interval`	The `datey_interval`.
`value`	The `datey` to test for inclusion. A numeric value is also accepted and is coerced to `datey`.

Value

A vector of `logical` corresponding to whether the interval includes the value. Always TRUE or FALSE – NAs result in FALSE.

See Also

datey_interval, interval_properties, interval_nature, ops

Examples

```
t_2000 <- datey(2000)
t_2001 <- datey(2001)
t_2002 <- datey(2002)
t_2003 <- datey(2003)
t_2004 <- datey(2004)

interval <- t_2000 %to% t_2003
interval %includes% t_2000
interval %includes% t_2001 # Start of interval *is* included
interval %includes% t_2002
interval %includes% t_2003 # End of interval *not* included
interval %includes% t_2004
interval %includes% NA_datey_        # NAs are FALSE
NA_datey_interval_ %includes% t_2004 # NAs are FALSE
interval_includes(NA_datey_interval_, t_2002) # NAs are FALSE

# Function syntax:
interval_includes(interval, t_2002)
```

`interval_nature` *Properties of a* `datey_interval`

Description

Test whether intervals, $[a, b)$, are 'proper' or 'collapsed':

- A *proper* interval does not end before its start, i.e. $a \leq b$.
- A *collapsed* interval does not start before its end, i.e. $a \geq b$.

An NA interval is treated as collapsed and improper.

These definitions imply the following:

- A collapsed interval could be empty or improper.
- To test for an empty interval, i.e. $[a, a)$, test that it is both proper and collapsed.

These methods are guaranteed to return `TRUE` or `FALSE`, i.e. not `NA` (provided the argument is an interval).

Vector versions mapping each element of `x` to `TRUE` or `FALSE`:

`is_proper(x)` tests whether the elements of `x` are proper. `is_collapsed(x)` tests whether the elements of `x` are collapsed.

Scalar versions mapping `x` to a scalar `TRUE` or `FALSE`:

`all_proper(x)` tests whether all the elements of `x` are proper. `all_collapsed(x)` tests whether all the elements of `x` are collapsed. `any_collapsed(x)` tests whether at least one of the elements of `x` is collapsed.

(`any_proper()` is not implemented because there is no obvious use case.)

These are S3 generic functions.

Usage

```
is_proper(x)

## Default S3 method:
is_proper(x)

## S3 method for class 'datey_interval'
is_proper(x)

all_proper(x)

## Default S3 method:
all_proper(x)

## S3 method for class 'datey_interval'
all_proper(x)

is_collapsed(x)

## Default S3 method:
is_collapsed(x)

## S3 method for class 'datey_interval'
is_collapsed(x)

all_collapsed(x)

## Default S3 method:
all_collapsed(x)

## S3 method for class 'datey_interval'
all_collapsed(x)

any_collapsed(x)

## Default S3 method:
any_collapsed(x)
```

```
## S3 method for class 'datey_interval'
any_collapsed(x)
```

Arguments

x	The interval to test.

Value

- `is_XXX` functions return a logical vector corresponding the property.
- `all_XXX` and `any_XXX` functions return a logical scalar.

See Also

datey_interval, interval_properties, interval_includes

Examples

```
non_empty   <- 1900 %to% 2000
empty       <- 2000 %to% 2000
improper    <- 2000 %to% 1900
na          <- NA_datey_interval_

is_collapsed(non_empty) # FALSE
is_collapsed(empty)     # TRUE
is_collapsed(improper)  # TRUE
is_collapsed(na)        # TRUE

is_proper(non_empty)    # TRUE
is_proper(empty)        # TRUE
is_proper(improper)     # FALSE
is_proper(na)           # FALSE
```

`interval_properties` *Get the start, end or duration of a* `datey_interval`

Description

Get the start, end or duration of a `datey_interval` using the syntax `$start`, `$end` or `$duration` respectively.
`$duration` is the duration of the interval *provided it is proper* (i.e. start <= end). If the interval is improper then `$durationis` `NA_durationy_`. When `x` is a `datey_interval` then `x$duration` is identical to `durationy(x)`.

Usage

```
## S3 method for class 'datey_interval'
x$name
```

Arguments

x	The datey_interval.
name	Must be start, end or duration.

Value

start and end return a vector of datey; duration returns a vector of durationy.

See Also

datey_interval, interval_nature, interval_includes

Examples

```
t_1 <- start_day(2001, 1, 1)
t_2 <- start_day(2002, 2, 2)
interval <- datey_interval(t_1, t_2)
interval
interval$start
interval$end
interval$duration
```

---

is_leap_year *Is* x *a leap year?*

Description

Tests whether a date or year is a leap year.

This returns NA if the corresponding datey would be invalid, including years before 1000 and dates after 3000-01-01.0.

For numerics, this returns NA for years values less than 1000 or greater than 3000.

This is an S3 generic. This package provides methods for:

- numeric types double and integer (interpreted as calendar years, e.g. 2000.9 means the calendar year 2000), and
- date types datey, Date, POSIXct and POSIXlt.

Usage

```
is_leap_year(x)

## Default S3 method:
is_leap_year(x)

## S3 method for class 'integer'
is_leap_year(x)
```

```
## S3 method for class 'double'
is_leap_year(x)

## S3 method for class 'datey'
is_leap_year(x)

## S3 method for class 'Date'
is_leap_year(x)

## S3 method for class 'POSIXct'
is_leap_year(x)

## S3 method for class 'POSIXlt'
is_leap_year(x)
```

Arguments

x	A vector date type or numeric year.

Value

NA if x is not interpretable as a year or date, or outside [1000,3000], TRUE if x is a leap year, otherwise FALSE.

See Also

datey

Examples

```
any(is_leap_year(c(1900, 1901, 2001))) #FALSE
all(is_leap_year(c(1904.1, 2000.5, 2004.9))) #TRUE
```

---

is_NA *Whether* datey, durationy *or* datey_interval *are NA*

Description

Valid datey system ranges:

- Valid dates are from the start of 1000 to the start of 3000.
- Valid durations are 2000 years or less in magnitude.

Values outside the above ranges are treated as NA.

is.na() tests whether a datey, durationy or datey_interval is NA by element.

anyNA() tests whether any element of a datey, durationy or datey_interval is NA.

For convenience,

- the constants NA_datey_, NA_durationy_ and NA_datey_interval_ are the `datey`, `durationy` and `datey_interval` versions of NA respectively, and
- integer constants describing the above valid ranges are also provided.

For performance reasons, intermediate **datey** system calculations are *not* required to check for NAs.

Throughout the **datey** package, `NA` will cause an error when used where a `datey`, `durationy` or `datey_interval` is expected. This is because its type is `logical` and potentially indicates user error. If you want an NA value with a **datey** system type, use one of `NA_datey_`, `NA_durationy_` or `NA_datey_interval_`.

Usage

```
## S3 method for class 'datey'
is.na(x)

## S3 method for class 'datey'
anyNA(x, recursive = FALSE)

## S3 method for class 'datey_interval'
is.na(x)

## S3 method for class 'datey_interval'
anyNA(x, recursive = FALSE)

## S3 method for class 'durationy'
is.na(x)

## S3 method for class 'durationy'
anyNA(x, recursive = FALSE)
```

Arguments

`x`	The `datey`, `durationy` or `datey_interval` to test for NA.
`recursive`	Currently required to be `FALSE` (the default).

Value

`is.na()` returns a vector of logical the same length as `x`. `anyNA()` always returns `TRUE` or `FALSE`, never `NA` and never anything other than a single value.

See Also

NA_datey_, NA_durationy_, NA_datey_interval_, integer_constants, datey, durationy, datey_interval

Examples

```
t <- c(NA_datey_, datey(2000), datey(999.99, strict = FALSE))
is.na(t)
anyNA(t)
```

```
d <- c(NA_durationy_, durationy(1.5))
is.na(d)
anyNA(d)

i <- c(NA_datey_interval_, 2000 %to% 2001)
is.na(i)
anyNA(i)
```

---

is_type *Is* x *a* datey, durationy *or* datey_interval*?*

Description

These methods will always return a scalar logical TRUE or FALSE:

- is_datey() tests whether an object is a datey.
- is_durationy() tests whether an object is a durationy.
- is_datey_interval() tests whether an object is a datey_interval.

Usage

```
is_datey(x)

is_datey_interval(x)

is_durationy(x)
```

Arguments

x	The object to test.

Value

A logical scalar indicating whether x a datey, durationy or datey_interval as appropriate. Always FALSE or TRUE; never NULL or NA.

Examples

```
t <- datey(2000:2001)
t
is_datey(t)
is_datey(NULL)
is_datey(NA)

d <- durationy(0:2)
d
is_durationy(d)
is_durationy(NULL)
```

```
is_durationy(NA)

interval <- datey(2000:2001) %to% datey(2001:2002)
interval
is_datey_interval(interval)
is_datey_interval(NULL)
is_datey_interval(NA)
```

`is_xxx_day` *Is a* `datey` *the start (or end) or middle of a day?*

Description

`is_start_day()` checks whether `x` is the start or end of a day.

`is_mid_day()` checks whether `x` is the middle of a day.

These properties are *not* necessarily preserved when a duration of *n* years is added or subtracted. If that's critical (e.g. for dataset anniversaries) then use anniversary days.

Usage

```
is_start_day(x)

is_mid_day(x)
```

Arguments

`x`	The `datey` to test.

Value

A vector of `logical`. Elements of the `datey` vector that are `NA_datey_` will result in `NA` elements of the result vector.

See Also

datey, anniversary_days

Examples

```
# Create (NA, 0 days, 1/4 day, 1/2 day):
t <- datey(c(NA_real_, 2000, 2000 + 0.25/366, 2000 + 0.5/366))
t # <NA>  2000-01-01.0  2000-01-01.25 2000-01-01.5

is_start_day(t) # NA  TRUE FALSE FALSE
is_mid_day(t)   # NA FALSE FALSE  TRUE

# Properties are not necessarily preserved between years:
t <- start_day(2000,7,1) # Leap year
is_start_day(t) # TRUE
is_start_day(t + durationy(1)) # FALSE
```

max_min *Minimum, maximum or range of* datey *or* durationy

Description

Gets the minimum, maximum or range of one or more datey or durationy vectors. All arguments must be of the same type.

Returns a typed NA (NA_datey_ or NA_durationy_) for empty input or when all values are NA and na.rm = TRUE.

These are S3 methods for the Summary group generic.

Usage

```
## S3 method for class 'datey_interval'
Summary(..., na.rm = FALSE)

## S3 method for class 'datey'
Summary(..., na.rm = FALSE)

## S3 method for class 'durationy'
Summary(..., na.rm = FALSE)
```

Arguments

...	One or more datey or durationy vectors. All must be the same type.
na.rm	A logical (TRUE or FALSE) indicating whether NA values should be removed before the computation.

Value

min and max return a scalar. range returns a two element vector, the first element being the minimum and the second the maximum.

See Also

datey, durationy, mean.datey

Examples

```
t <- datey(2000:2003)
t
min(t)
max(t)
range(t)
```

mean *Mean value of* datey *or* durationy

Description

Gets the mean value of a vector of datey or durationy as a scalar.

This will entail rounding if the mean of the underlying click counts is not an integer.

These are S3 methods for the mean generic.

Usage

```
## S3 method for class 'datey_interval'
mean(x, ..., na.rm = FALSE)

## S3 method for class 'datey'
mean(x, ..., na.rm = FALSE)

## S3 method for class 'durationy'
mean(x, ..., na.rm = FALSE)
```

Arguments

x	The datey or durationy.
...	Not used.
na.rm	A logical (TRUE or FALSE) indicating whether NA values should be removed before the computation.

Value

A scalar of datey or durationy as appropriate.

See Also

datey, durationy, max_min

Examples

```
t <- datey(2000:2003)
t
mean(t)
```

NAs *The* datey, durationy *and* datey_interval *versions of NA*

Description

Throughout the **datey** package, NA will cause an error when used where a datey, durationy or datey_interval is expected. This is because its type is logical and potentially indicates user error. If you want an NA value with a **datey** system type, use one of NA_datey_, NA_durationy_ or NA_datey_interval_.

Usage

```
NA_datey_

NA_datey_interval_

NA_durationy_
```

Value

Each is a scalar holding the missing-value (NA) representation of its type: NA_datey_ is a datey, NA_durationy_ is a durationy and NA_datey_interval_ is a datey_interval.

See Also

is_NA, integer_constants, datey, durationy, datey_interval

Examples

```
is_datey(NA_datey_)
is.na(NA_datey_)
is_durationy(NA_durationy_)
is.na(NA_durationy_)
is_datey_interval(NA_datey_interval_)
```

ops *Operators for* datey, durationy *and* datey_interval

Description

The unary - operator can be applied to a durationy to change its sign.

The following are the available binary operations on datey,durationy and datey_interval only operands, and their meaning:

Left	Operators	Right	Result	Notes

datey	== != < <= > >=	datey	logical	Comparisons for dates
durationy	== != < <= > >=	durationy	logical	Comparisons for durations
datey_interval	== !=	datey_interval	logical	Equality for date intervals
datey	-	datey	durationy	Duration between two dates
datey	+ -	durationy	datey	A date offset by a duration
durationy	+	datey	datey	A date offset by a duration
durationy	+ -	durationy	durationy	Duration addition and subtraction
datey	%to%	datey	datey_interval	Create a date interval – syntactic sugar for dat
datey_interval	%includes%	datey	logical	Whether an interval includes a date – syntactic
datey_interval	&	datey_interval	datey_interval	Intersection of two date intervals – NA_datey_

dateys and durationys can also be mixed with numeric operands, in which case the datey or durationy is first converted to years, The following operations are implemented

- Comparison, i.e. a datey or durationy == != < <= > >= numeric or vice versa. Result is logical.
- datey addition and subtraction, i.e. a datey + - a numeric or vice versa. Result is double.
- durationy arithmetic, i.e. a durationy + - * / a numeric or vice versa. Result is double.
- The %to% operator accepts numbers, which are treated as years and coerced to datey.
- The %includes% operator accepts a number as its right hand operand, which is treated as years and coerced to datey.

When applied to datey_intervals, & is the 'intersection' operator. For intervals that do not intersect the result of & depends on whether the intervals are adjacent. If they are adjacent then the result is an empty interval starting (and ending) at the point in time they touch. Otherwise it is NA_datey_interval_. You can test whether intervals a and b intersect using is_collapsed(a & b).

Throughout the **datey** package, NA will cause an error when used where a datey, durationy or datey_interval is expected. This is because its type is logical and potentially indicates user error. If you want an NA value with a **datey** system type, use one of NA_datey_, NA_durationy_ or NA_datey_interval_.

Usage

```
## S3 method for class 'datey_type'
Ops(e1, e2)
```

Arguments

e1	First parameter.
e2	Second parameter (missing if a unary operator).

Value

See above table. In essence

- subtracting two dateys results in a durationy,

- comparing two Ts results in a logical,
- adding or subtracting a durationy to or from a T results in a T, and
- mixing durationy and datey with numeric operands first converts the durationy and datey to years and then results in standard numeric evaluation,

where T is either datey or durationy in each of the above.

See Also

datey, durationy, datey_interval, vignette("datey", package = "datey") for a worked introduction

Examples

```
t_2000 <- datey(2000)
t_2001 <- datey(2001)
d_0.5 <- durationy(0.5)

t_2000
t_2001
d_0.5

t_2001 - t_2000 # `datey` - `datey` is a `durationy`
t_2000 + d_0.5  # `datey` + `durationy` is a `datey`
t_2001 - d_0.5  # `datey` - `durationy` is a `datey`
t_2000 + 0.5    # Arithmetic with numerics results in a double
d_0.5 + d_0.5   # `durationy` + `durationy` is a `durationy`
d_0.5 + 0.5     # Arithmetic with numerics results in a double
d_0.5 * 2       # Arithmetic with numerics results in a double

interval <- t_2000 %to% t_2001
interval

# %to% also accepts numbers:
2000 %to% 2001

interval %includes% t_2000 # TRUE -- start *is* included in an interval
interval %includes% (t_2000 + d_0.5) # TRUE
interval %includes% t_2001 # FALSE -- end is *not* included in an interval

# %includes% also accepts a number as its right hand operand:
interval %includes% 2000.5

# %includes% handling of NAs:
interval %includes% NA_datey_         # FALSE (not NA)
NA_datey_interval_ %includes% t_2000  # FALSE (not NA)

(2000 %to% 2020) & (2010 %to% 2030) # [2010-01-01.0, 2020-01-01.0)

# Non-intersecting *adjacent* intervals:
(2000 %to% 2001) & (2001 %to% 2002) # [2001-01-01.0, 2001-01-01.0)
# Non-intersecting *non*-adjacent intervals:
```

```
(1900 %to% 1901) & (2001 %to% 2001) # <NA>
```

`replicate` *Replicate* `datey`*,* `durationy` *or* `datey_interval` *vectors*

Description

Replicates `datey`, `durationy` or `datey_interval` vectors.

Usage

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

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

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

Arguments

`x`	A `datey`, `durationy` or `datey_interval`.
`...`	Other arguments.

Value

The replicated vector.

See Also

datey, durationy, datey_interval

Examples

```
x <- datey(2001:2004)
rep(x, 2)
rep(x, each = 2)
pmax(x, datey(2002))
```

seq *Create* datey *or* durationy *sequence vector*

Description

Creates a datey or durationy vector by defining a sequence.

Usage

```
## S3 method for class 'datey'
seq(from, to, by, ...)

## S3 method for class 'durationy'
seq(from, to, by, ...)
```

Arguments

from	The first value in the sequence. A scalar datey or durationy.
to	The sequence stops before values exceed to. A scalar datey or durationy.
by	The increment of the sequence. A scalar durationy.
...	Not used.

Value

The sequence as a vector of datey (for seq.datey) or durationy (for seq.durationy).

See Also

datey, durationy

Examples

```
seq(from = datey(2000), to = datey(2005), by = durationy(2))
seq(from = datey(2000), to = datey(1999), by = durationy(-0.25))
```

subset *Subset* datey, durationy *or* datey_interval *vectors*

Description

Subsets datey, durationy or datey_interval vectors.

Usage

```
## S3 method for class 'datey'
x[i, ...]

## S3 method for class 'durationy'
x[i, ...]

## S3 method for class 'datey_interval'
x[i, ...]

## S3 replacement method for class 'datey'
x[i] <- value

## S3 replacement method for class 'durationy'
x[i] <- value

## S3 replacement method for class 'datey_interval'
x[i] <- value
```

Arguments

x	A datey, durationy or datey_interval.
i	Indices to extract.
...	Other arguments.
value	Value to assign.

Value

The subset.

See Also

datey, durationy, datey_interval, combine

Examples

```
x <- datey(2001:2004)
x
x[2:3]
x[2:3] <- datey(1999)
x
```

`text_from_datey` *Format or print a* `datey`

Description

A `datey` is printed as either

- `YYYY-MM-DD`, i.e. ISO 8601 extended date format, or
- `YYYY-MM-DD.F...` where `.F...` is the day fraction part.

If `include_day_fraction` is `TRUE` then `[.F...]` is included even if it is 0 (i.e. `.0`).

Note that a `datey` created as the end of a day (or with day fraction 1) will print as the start of the following day.

Usage

```
## S3 method for class 'datey'
as.character(x, ...)

## S3 method for class 'datey'
format(x, include_day_fraction = TRUE, ...)

## S3 method for class 'datey'
print(x, include_day_fraction = TRUE, max = NULL, ...)
```

Arguments

`x`	The `datey` to print or format.
`...`	Other arguments.
`include_day_fraction`	Whether to include the fractional day part. Defaults to `TRUE`.
`max`	Numeric or `NULL`, specifying the maximal number of entries to be printed. When `NULL`, `getOption("max.print")` used. Defaults to `NULL`.

Value

`as.character` and `format` return a vector of `character`. `print` invisibly returns `x`.

See Also

datey

Examples

```
start      <- start_day(2001, 2, 3)
fractional <- datey(    2001, 2, 3, day_fraction = 0.4444)
mid        <- mid_day(  2001, 2, 3)
end        <- end_day(  2001, 2, 3)

format(start)                                # "2001-02-03.0"
format(start, include_day_fraction = FALSE) # "2001-02-03"
format(fractional)                           # "2001-02-03.4447"
format(mid)                                  # "2001-02-03.5"
format(end)                                  # "2001-02-04.0"
```

---

text_from_datey_interval

Format or print a `datey_interval`

Description

A `datey_interval` is printed as "[start, end)", where start and end are printed either as

- `YYYY-MM-DD`, i.e. ISO 8601 extended date format, or
- `YYYY-MM-DD.F...` where `.F...` is the day fraction part.

If `include_day_fraction` is `TRUE` then `[.F...]` is included even if it is 0 (i.e. `.0`).

Usage

```
## S3 method for class 'datey_interval'
as.character(x, ...)

## S3 method for class 'datey_interval'
format(x, include_day_fraction = TRUE, ...)

## S3 method for class 'datey_interval'
print(x, include_day_fraction = TRUE, max = NULL, ...)
```

Arguments

`x`	The `datey_interval` to print or format.
`...`	Further arguments to be passed from or to other methods.
`include_day_fraction`	Whether to include the fractional day part. Defaults to `TRUE`.
`max`	Numeric or `NULL`, specifying the maximal number of entries to be printed. When `NULL`, `getOption("max.print")` used. Defaults to `NULL`.

Value

`as.character` and `format` return a vector of `character`. `print` invisibly returns `x`.

See Also

datey_interval

`text_from_durationy` *Format or print a* `durationy`

Description

A `durationy` is printed as a decimal.

This format is readable by `durationy.character()`.

Usage

```
## S3 method for class 'durationy'
as.character(x, ...)

## S3 method for class 'durationy'
format(x, include_plus = FALSE, use_true_minus = TRUE, year_unit = "yr", ...)

## S3 method for class 'durationy'
print(
  x,
  include_plus = FALSE,
  use_true_minus = TRUE,
  year_unit = "yr",
  max = NULL,
  ...
)
```

Arguments

`x`	The `durationy` to print or format.
`...`	Other arguments.
`include_plus`	Whether to include a plus ('+') sign for positive durations. Defaults to `FALSE`.
`use_true_minus`	Whether to use the true minus sign (U+2212) as opposed to the ASCII hyphen (U+002D). Defaults to `TRUE`.
`year_unit`	The year unit name to print. If not blank then the value is followed by a space and the unit. Cannot be more than 20 characters (UTF-8 bytes) or contain control characters. Defaults to `"yr"`.
`max`	Numeric or `NULL`, specifying the maximal number of entries to be printed. When `NULL`, `getOption("max.print")` used. Defaults to `NULL`.

Value

`as.character` and `format` return a vector of `character`. `print` invisibly returns `x`.

See Also

durationy

Examples

```
pos <- durationy(1)
neg <- durationy(-2.3)
format(pos) # "1 yr"
format(pos, include_plus = TRUE) # "+1 yr"
format(pos, year_unit = "") # "1"
format(neg) # U+2212 (true minus) followed by "2.3" (CRAN-compliance)
format(neg, use_true_minus = FALSE) # "-2.3 yr"
format(neg, use_true_minus = FALSE, year_unit = "a") # "-2.3 a"
```

Index

∗ **NA**
    is_NA, 22
∗ **combine**
    combine, 6
∗ **replicate**
    replicate, 31
∗ **sequence**
    seq, 32
∗ **subset**
    subset, 32
[.datey *(*subset*)*, 32
[.datey_interval *(*subset*)*, 32
[.duration y *(*subset*)*, 32
[<-.datey *(*subset*)*, 32
[<-.datey_interval *(*subset*)*, 32
[<-.duration y *(*subset*)*, 32
$.datey *(*datey_components*)*, 11
$.datey_interval *(*interval_properties*)*, 20
%includes% *(*interval_includes*)*, 17
%to% *(*datey_interval*)*, 12

all_collapsed *(*interval_nature*)*, 18
all_of_time, 2, *13*, *17*
all_proper *(*interval_nature*)*, 18
anniversary days, *25*
anniversary_days, 3, *9*, *12*, *25*
any_collapsed *(*interval_nature*)*, 18
anyNA.datey *(*is_NA*)*, 22
anyNA.datey_interval *(*is_NA*)*, 22
anyNA.duration y *(*is_NA*)*, 22
as.character.datey *(*text_from_datey*)*, 34
as.character.datey_interval *(*text_from_datey_interval*)*, 35
as.character.duration y *(*text_from_duration y*)*, 36
as.character.duration y(), *14*
as.double.datey *(*as_years_datey*)*, 4
as.double.duration y *(*as_years_duration y*)*, 5
as.integer.datey *(*as_years_datey*)*, 4
as.integer.duration y *(*as_years_duration y*)*, 5
as_years_datey, 4, *6*, *9*
as_years_duration y, *5*, 5, *15*

c(), *7*
c.datey *(*combine*)*, 6
c.datey_interval *(*combine*)*, 6
c.duration y *(*combine*)*, 6
combine, 6, *33*

datey, *4, 5*, *7*, 7, *12*, *15*, *17*, *22, 23*, *25–28*, *30–34*
datey_components, *4*, *9*, 11
datey_interval, *3*, *7*, *9*, 12, *15*, *18*, *20, 21*, *23*, *28*, *30, 31*, *33*, *36*
duration y, *6, 7*, *9*, *13*, 14, *23*, *26–28*, *30–33*, *37*
duration y.character(), *36*

end_day *(*datey*)*, 7

format.datey *(*text_from_datey*)*, 34
format.datey_interval *(*text_from_datey_interval*)*, 35
format.duration y *(*text_from_duration y*)*, 36

integer constants, *23*
integer_constants, 16, *23*, *28*
interval_includes, *13*, 17, *20, 21*
interval_nature, *13*, *18*, 18, *21*
interval_properties, *13*, *18*, *20*, 20
is.na.datey *(*is_NA*)*, 22
is.na.datey_interval *(*is_NA*)*, 22
is.na.duration y *(*is_NA*)*, 22
is_collapsed *(*interval_nature*)*, 18
is_datey *(*is_type*)*, 24
is_datey_interval *(*is_type*)*, 24
is_duration y *(*is_type*)*, 24

is_leap_year, *9*, 21
is_mid_day *(*is_xxx_day*)*, 25
is_NA, *9*, *13*, *15*, *17*, 22, *28*
is_proper *(*interval_nature*)*, 18
is_start_anniversary_day
*(*anniversary_days*)*, 3
is_start_day *(*is_xxx_day*)*, 25
is_type, 24
is_xxx_day, *4*, *9*, 25

max_min, 26, *27*
mean, 27
mean.datey, *26*
mid_day *(*datey*)*, 7

NA_datey_, *23*
NA_datey_ *(*NAs*)*, 28
NA_datey_interval_, *23*
NA_datey_interval_ *(*NAs*)*, 28
NA_durationy_, *23*
NA_durationy_ *(*NAs*)*, 28
NAs, *17*, 28

ops, *3*, *5*, *6*, *9*, *13*, *15*, *18*, 28
Ops.datey_type *(*ops*)*, 28

print.datey *(*text_from_datey*)*, 34
print.datey_interval
*(*text_from_datey_interval*)*, 35
print.durationy *(*text_from_durationy*)*,
36

rep.datey *(*replicate*)*, 31
rep.datey_interval *(*replicate*)*, 31
rep.durationy *(*replicate*)*, 31
replicate, 31

seq, 32
start_anniversary_day
*(*anniversary_days*)*, 3
start_anniversary_day_to_ymd
*(*anniversary_days*)*, 3
start_day *(*datey*)*, 7
subset, *7*, 32
Summary.datey *(*max_min*)*, 26
Summary.datey_interval *(*max_min*)*, 26
Summary.durationy *(*max_min*)*, 26

text_from_datey, *9*, *12*, 34
text_from_datey_interval, 35
text_from_durationy, *15*, 36
to_ymdf *(*datey_components*)*, 11

valid_duration_years_max
*(*integer_constants*)*, 16
valid_years_end *(*integer_constants*)*, 16
valid_years_start *(*integer_constants*)*,
16