# Package 'tind'

February 2, 2026

**Title** A Common Representation of Time Indices of Different Types

**Version** 0.2.4

**Date** 2026-02-02

**Description** Provides an easy-to-use tind class representing time indices of different types (years, quarters, months, ISO 8601 weeks, dates, time of day, date-time, and arbitrary integer/numeric indices). Includes an extensive collection of functions for calendrical computations (including business applications), index conversions, index parsing, and other operations. Auxiliary classes representing time differences and time intervals (with set operations and index matching functionality) are also provided. All routines have been optimised for speed in order to facilitate computations on large datasets. More details regarding calendars in general and calendrical algorithms can be found in ``Calendar FAQ'' by Claus Tøndering <https://www.tondering.dk/claus/calendar.html>.

**Depends** R (>= 3.6.0)

**Suggests** crayon, methods, stats, testthat, knitr, rmarkdown, fansi, htmltools

**Enhances** utils, graphics, ggplot2, zoo, timeDate, chron, data.table, hms

**Collate** base-utils.R base-in.R base-yqmwd.R base-tz.R base-th.R base-res.R base-pretty.R base-locale.R base-types.R base-parse.R tind.R format.R parse.R coercion.R methods.R tdiff.R tinterval.R types.R resolution.R tzone.R tspan.R print_summ_str.R ops.R setops_match.R round.R cut.R seq.R ord_reg.R calendar.R other.R calendars.R business.R merge.R pretty.R plot-scales.R package.R

**License** GPL-3

**URL** https://github.com/dever-pl/tind

**BugReports** https://github.com/dever-pl/tind/issues

**Encoding** UTF-8

**NeedsCompilation** yes

**VignetteBuilder** knitr

**RoxygenNote** 7.3.3

**Author** Grzegorz Klima [aut, cre, cph]

**Maintainer** Grzegorz Klima <dever@post.pl>

**Repository** CRAN

**Date/Publication** 2026-02-02 21:10:02 UTC

# Contents

---

`tind-package` *A Common Representation of Time Indices of Different Types*

---

## Description

The goal of **tind** project is to provide users with a *single* class capable of representing time indices of different types. Currently, these include: years, quarters, months, ISO 8601 weeks, dates, time of day, date-time, and arbitrary integer/numeric indices. **tind** provides a unified interface and a collection of methods, which can be used with all supported index types.

The package provides an extensive collection of functions for calendrical computations (including business applications), index conversions, index parsing, and other operations. Examples of use of **tind** functions and methods can be found in their documentation and in package vignettes.

All routines have been optimised for speed in order to facilitate computations on large datasets.

## Details

### Options Controlling tind Behaviour

`tind.abbr.year.start` is a number in range 0–99 determining how two-digit years are interpreted during parsing. By default (69), two-digit numbers smaller than 69 are interpreted as years in 2000s and equal or greater than 69 as years in 1900s.

`tind.warn.diff.tz` is a logical value (`TRUE` by default) determining whether warnings should be raised when performing computations involving date-time indices with different time zone attributes.

### Implementation

The code structure is hierarchical by design. The first layer consists of low-level C and R functions working with particular time index types and converting between them. The second layer consists of exported (user-accessible) functions, which are responsible for argument checks and dispatching to the 1st layer functions.

Computations with **tind** for types other than date-time are implemented in C from scratch. Many algorithms were taken from *Calendar FAQ* by Claus Tøndering. For date-time computations, **tind** partially relies on base R infrastructure (`POSIXlt` class), that is on conversion `as.POSIXlt.numeric`.

## Author(s)

Grzegorz Klima <dever@post.pl>

## References

Claus Tøndering, *Calendar FAQ*, https://www.tondering.dk/claus/calendar.html.

as.tind *Conversion to* tind *Class*

## Description

as.tind method allows for conversion of numeric and character vectors as well as objects of Date, POSIXct, and POSIXlt classes to tind objects to tind.

as.tind method for tind class allows to change the type of index of an object of tind class. Convenience functions as.year, as.quarter, as.month, as.week, as.date, as.date_time, as.time allow to quickly convert argument to the indicated type.

## Usage

```
as.tind(x, ...)

## S3 method for class 'numeric'
as.tind(x, type = NULL, tz = NULL, ...)

## S3 method for class 'character'
as.tind(
  x,
  type = NULL,
  format = NULL,
  order = NULL,
  locale = NULL,
  tz = NULL,
  ...
)

## S3 method for class 'Date'
as.tind(x, ...)

## S3 method for class 'POSIXct'
as.tind(x, tz = NULL, digits = 0L, ...)

## S3 method for class 'POSIXlt'
as.tind(x, tz = NULL, digits = 0L, ...)

## S3 method for class 'data.frame'
as.tind(x, ...)

## S3 method for class 'tind'
as.tind(x, type = NULL, tz = NULL, ...)

as.year(x, ...)

as.quarter(x, ...)
```

```
as.month(x, ...)

as.week(x, ...)

as.date(x, ...)

as.date_time(x, tz = NULL, ...)

as.time(x, ...)
```

## Arguments

| | |
|---|---|
| x | an R object (e.g., a numeric vector, a character vector, a Date or POSIXct object). |
| ... | further arguments passed to or from other methods. |
| type | a character determining time index type or NULL. |
| tz | (optional) a character value determining the time zone (the default NULL is interpreted as the system time zone). See tzone documentation for information on time zones. |
| format | a character determining input format(s) as in strptind. (or NULL). |
| order | a character determining order(s) of time index components in the input as in parse_t (or NULL). |
| locale | (optional) a character value determining locale or NULL (the default, interpreted as the current system locale), see calendar-names for information on locale settings. |
| digits | an integer value (0–6) determining the number of decimal places for seconds to be preserved during conversion (0 by default). |

## Details

### Numeric vectors

The following numeric representations are automatically recognised (between year 1800 and 2199): YYYY (years), YYYYQ (quarters), YYYYMM (months), and YYYYMMDD (dates). Conversion from numeric vectors to other index types requires type specification via type argument.

Date-time indices are represented as number of seconds since the Epoch (1970-01-01 00:00 UTC). Time of day is represented as the number of seconds since midnight.

### Character vectors

as.tind automatically recognises many different formats. If automatic recognition fails, type argument could help identify the format. For less standard formats / representations, one can use either format argument (which is forwarded to strptind) or order argument (which is forwarded to parse_t).

### Data frames

If a data frame has one column, it is converted using appropriate method depending on the column type. In case there are two or more columns, the approach depends on column types. If there are

two columns representing dates and times (time of day) either as tinds or other classes recognised by the package, date-time indices are constructed using date_time function. When all columns are numeric, the method forwards to tind constructor. order argument has to be provided (via ...) indicating order of time index components in the columns. In the general case, all columns are pasted (with spaces separating them) and the resulting character vector is parsed. This will again require order argument (or type, or format). Due to pasting and subsequent parsing, this may not be computationally efficient for larger datasets.

**Date and POSIXt classes**

Conversion of Date objects returns time index of type "d" (date). POSIXct and POSIXlt classes are converted to index of type "t" (date-time). If time zone attribute is not set for the argument, system time zone is assumed.

**Other classes representing time indices**

For conversions between tind class and other classes (from packages other than **base**), see tind-other.

## Value

An object of tind class of length equal to the length of the argument. For data frame version the length of the result is equal to the number of rows in the data frame.

## See Also

tind constructor, strptind function for format specifications, parse_t function for order specifications, tind-coercion for conversions from tind, and tind-other for conversions between tind class and other classes (from packages other than **base**).

## Examples

```
## numeric and character arguments
# years
as.tind(1999)
as.tind("1999")
# quarters
as.tind(20043)
as.tind("20043")
# months
as.tind(200109)
as.tind("2001-09")
as.tind("200109")
# need proper locale to recognise English month names
as.tind("Sep 2001", locale = "C")
# weeks (ISO 8601)
# numeric YYYYWW representation is not automatically recognised, need type
as.tind(200936, "w")
as.tind("2009-W36")
# dates
as.tind(20200726)
as.tind("2020-07-26")
# need proper locale to recognise English month names
as.tind("Jul 26, 2020", locale = "C")
as.tind("07/26/20")
```

```
# date-time
as.tind("2000-08-16 08:17:38")
# time
as.tind("08:17:38")
as.tind(08 * 3600 + 17 * 60 + 38, type = "h")

## conversion from Date and POSIXct
as.tind(Sys.Date())
as.tind(Sys.time())

## as.year, ..., as.time
# today
(x <- today())
as.year(x)
as.quarter(x)
as.month(x)
as.week(x)
# midnight
as.date_time(x)
# current time
(x <- now())
as.year(x)
as.quarter(x)
as.month(x)
as.week(x)
as.date(x)
as.time(x)
```

---

`as.tzone` *Get the Same Date and Time in a Different Time Zone*

---

## Description

This method allows to determine time index representing the same date and time in a different time zone.

## Usage

```
as.tzone(x, tz)

## S3 method for class 'tind'
as.tzone(x, tz)

## S3 method for class 'POSIXct'
as.tzone(x, tz)

## S3 method for class 'POSIXlt'
as.tzone(x, tz)
```

```
## S3 method for class 'tinterval'
as.tzone(x, tz)
```

## Arguments

| | |
|---|---|
| x | an object of tind class or of POSIXct/POSIXlt classes (or of other class for which the method was implemented). |
| tz | a character value determining the new time zone. See tzone documentation for information on time zones. |

## Details

The underlying time (as measured by number of seconds since the Unix epoch in UTC) will change so that the date-time components in the new and old time zones are the same. For tind arguments, if (due to DST time changes or UTC offset changes) date-time indices do not occur in the new time zone, NAs are introduced with a warning. For tinterval arguments, the result is adjusted with a warning, in order not to create open-ended time intervals.

The method is implemented for objects of tind class of type "t" (date-time), objects of tinterval class of type "t" (time intervals), as well as base POSIXct and POSIXlt classes.

List of time zones supported by the particular R installation can be obtained via a call to OlsonNames function.

## Value

An object of the same class and length as x with adjusted underlying date-time representation and time zone set to tz.

## See Also

tzone method and date_time for construction od date-time indices from its components.

## Examples

```
if (all(c("Europe/Warsaw", "America/New_York") %in% OlsonNames())) {
# check time in one time zone
print(nw <- now(tz = "Europe/Warsaw"))
# the same date-time in a new time zone
print(nw2 <- as.tzone(nw, "America/New_York"))
# note the time difference (equal to the difference of UTC offsets)
# warning on different time zones will be issued
print(suppressWarnings(nw2 - nw))
try(nw2 - nw)
}
```

`axis.tind` *Add Time Axis — Plotting with* `graphics` *Package*

## Description

`axis.tind` adds time axis to a plot. Axes will be added automatically by **graphics**, but the default behaviour can be overridden by plotting without axis and then calling `axis.tind`, see Examples.

## Usage

```
axis.tind(
  side,
  x,
  at,
  format = NULL,
  locale = NULL,
  labels = TRUE,
  n.breaks = 5L,
  ...
)
```

## Arguments

| | |
|---|---|
| `side` | see `axis`. |
| `x` | time indices for which an axis is to be created. |
| `at` | (optional) time indices at which manual tick-marks and labels should be placed. |
| `format` | (optional) a character string determining label format or a formatting function, see `format`. |
| `locale` | (optional) a character string determining locale to be used for formatting labels, see calendar-names for information on locale settings. |
| `labels` | a logical value determining whether automatic labels should be placed at tick-marks or a character vector of labels. |
| `n.breaks` | an integer value, desired number of breaks. |
| `...` | further arguments passed to `axis`. |

## Value

Same as for `axis`, used for its side effect, which is to add time axis to an existing plot.

## See Also

`pretty` for computing pretty breakpoints, `axis_t` for calculating axis parameters, `scale_tind` for creating axes with **ggplot2**.

## Examples

```
# load graphics
library(graphics)
# artificial data
N <- 100
df <- data.frame(d = today() + (-N + 1):0, y = cumsum(rnorm(N)))
# default axis
plot(df$d, df$y, type = "l")
# custom date format with potentially more breaks and a smaller font
plot(df$d, df$y, type = "l", xaxt = "n")
axis.tind(1, df$d, format = "%m/%d/%y", n.breaks = 7L, cex.axis = .9)
```

---

`axis_t` *Compute Time Axis Parameters for Plotting*

---

## Description

Auxiliary function `axis_t` returns a six-element list with axis limits (in Cartesian coordinates), tick-mark positions (in Cartesian coordinates), tick-mark labels (character vector), positioning of minor tick-marks (in Cartesian coordinates), resolution of indices (in Cartesian coordinates), and `limits` argument converted to a `tinterval` of the same index type as `x`. The results can be used for manual creation of axes in plots.

## Usage

```
axis_t(
  x,
  limits = NULL,
  format = NULL,
  locale = NULL,
  expand = FALSE,
  n.breaks = 5L
)
```

## Arguments

| | |
|---|---|
| `x` | time indices for which an axis is to be created. |
| `limits` | `NULL` for automatic limits, `tinterval` of length 1 or `tind` of length 2. |
| `format` | (optional) a character string determining label format (see `format`) or a custom formatting function. |
| `locale` | (optional) a character string determining locale to be used for formatting labels, see calendar-names for information on locale settings. |
| `expand` | a logical value. If `TRUE`, limits are expanded by 3% on both sides. |
| `n.breaks` | an integer value, desired number of breaks. |

## Value

A six-element list with scale limits (`lim`), vector of tick-mark positions (`at`), character vector with tick-mark labels (`labels`), vector of minor tick-mark positions (`minor`), resolution (in Cartesian coordinates) of time indices (`resolution`), and `limits` argument converted to a `tinterval` of the same index type as `x` (`limits`).

## See Also

`pretty` for computing pretty breakpoints, `axis.tind` for creating axes with **graphics** package, `scale_tind` for creating axes with **ggplot2**.

## Examples

```
# load graphics
library(graphics)
# artificial data
N <- 180
df <- data.frame(d = today() + (-N + 1):0, y = cumsum(rnorm(N)))
(axt <- axis_t(df$d, format = "%m/%d/%y", n.breaks = 6L))
# custom time axis with minor breaks
plot(df$d, df$y, xlim = axt$lim, type = "l", xaxt = "n", xaxs = "i")
axis(1, at = axt$lim, labels = FALSE, lwd = 1, lwd.ticks = 0)
axis(1, at = axt$at, labels = axt$labels, lwd = 0, lwd.ticks = .7)
axis(1, at = axt$minor, labels = FALSE, lwd = 0, lwd.ticks = .4)
```

---

`bizday` *Business Days*

---

## Description

`bizday` computes the nearest business day from the date given a calendar function and one of the date rolling conventions (see Details).

`bizday_advance` advances date(s) by `n` business days.

`next_bizdays` determines the following `n` business days after a date.

`first_bizday_in_month/quarter` and `last_bizday_in_month/quarter` determine the first and the last business day in a month and a quarter.

`bizdays_in_month`, `bizdays_in_quarter`, and `bizdays_in_year` return the number of business days in particular time period.

`bizday_diff` computes the number of business days between two dates.

## Usage

```
bizday(d, convention, calendar)

bizday_advance(d, n = 1L, calendar)

next_bizdays(d, n = 1L, calendar)

first_bizday_in_month(m, calendar)

last_bizday_in_month(m, calendar)

first_bizday_in_quarter(q, calendar)

last_bizday_in_quarter(q, calendar)

bizdays_in_month(m, calendar)

bizdays_in_quarter(q, calendar)

bizdays_in_year(y, calendar)

bizday_diff(d1, d2, calendar, start.incl = TRUE, end.incl = FALSE)
```

## Arguments

| | |
|---|---|
| d | an object of tind class or an R object coercible to it, dates. |
| convention | a character value determining date rolling convention, see Details. |
| calendar | a function determining working days and holidays (see Details) or NULL. |
| n | an integer vector (for bizday_advance) or integer value (for next_bizdays), numebr of business days. |
| m | an object of tind class or an R object coercible to it, months. |
| q | an object of tind class or an R object coercible to it, quarters. |
| y | an object of tind class or an R object coercible to it, years. |
| d1 | an object of tind class or an R object coercible to it, start dates. |
| d2 | an object of tind class or an R object coercible to it, end dates. |
| start.incl | a logical value, if TRUE, the starting date is included in computation of the number of business days between two dates (TRUE by default). |
| end.incl | a logical value, if TRUE, the end date is included in computation of the number of business days between two dates (FALSE by default). |

## Details

The vectorised implementations of bizday, bizday_advance, and next_bizdays work under the assumption of at least one business day in a week and could return NAs for pathological calendar functions.

`bizday_advance` with increment 0 will adjust the date to the first preceding business day if it is not a business day (will act as `bizday(d, "p", *)`).

`next_bizdays` accepts negative arguments and returns an increasing sequence of business dates prior to `d` of length `abs(n)`.

### Conventions

The following date rolling conventions are supported (applied when the day is not a business day):

`"p"` preceding, the previous business day,

`"f"` following, the next business day,

`"mp"` modified preceding, the previous business day unless it falls in the previous month, in which case the next business day is chosen,

`"mf"` modified following, the next business day unless it falls in the next month, in which case the the previous business day is chosen,

`"mf2"` modified following bimonthly, the next business day unless it falls in the next month or the next half of the month (after 15th), in which case the the previous business day is chosen.

### Calendar Functions

Calendar function should take a vector of days as an argument and return a logical vector of the same length marking business days (as `TRUE`) or a list of two or three logical vectors of the same length with the first marking business days. See also calendars for real-life examples of calendar functions. When `calendar` function is not supplied, Monday-Friday are marked as business days.

## Value

`bizday` and `first/last_bizday_in_month/quarter` return vectors of dates of the same length as their first argument.

`bizday_advance` returns a vector of dates of length equal to length of the longer of the first two arguments (`d` and `n`).

`next_bizdays` returns a vector of dates of length `abs(n)`.

`bizday_diff` and `bizdays_in_month/quarter/year` return integer vectors.

## See Also

`calendars` for examples of calendar functions, `daycount_frac` for computations of day count fractions / accrual factors.

## Examples

```
# a trivial calendar function (Mon-Fri)
monfri <- function(d) (day_of_week(d) <= 5L)
# 2022-10-01 was Saturday
calendar("2022-10", monfri)
(d <- as.date("2022-10-01"))
bizday(d, "p", monfri)
bizday(d, "mp", monfri)
bizday(d, "f", monfri)
bizday(d, "mf", monfri)
```

```
bizday(d, "mf2", monfri)
# 2022-10-15 was Saturday again
calendar("2022-10", monfri)
(d <- as.date("2022-10-15"))
bizday(d, "p", monfri)
bizday(d, "mp", monfri)
bizday(d, "f", monfri)
bizday(d, "mf", monfri)
bizday(d, "mf2", monfri)
# 2022-12-31 was also Saturday
calendar("2022-12", monfri)
(d <- as.date("2022-12-31"))
bizday(d, "p", monfri)
bizday(d, "mp", monfri)
bizday(d, "f", monfri)
bizday(d, "mf", monfri)
bizday(d, "mf2", monfri)
```

---

`calendar-names` *Calendar Names*

---

## Description

These three functions return (abbreviated and full) names of months and days of week as well as AM/PM indicators in the current or user-provided locale.

How the month and weekday names are actually returned depends both on the selected locale and character set / code page setting.

## Usage

```
month_names(locale = NULL, abbreviate = TRUE)

weekday_names(locale = NULL, abbreviate = TRUE)

ampm_indicators(locale = NULL)
```

## Arguments

| | |
|---|---|
| `locale` | a character value determining locale or `NULL` (default, interpreted as the current system locale). |
| `abbreviate` | a logical value, if `TRUE`, abbreviated names are returned; if `FALSE`, full names are returned. `TRUE` by default. |

## Value

A character vector of length 12, 7, or 2.

## Locale Settings

Unfortunately, locale and character set naming were not standardised across different operating systems for many years. On modern operating systems, however, locale is usually of the form `xx_XX` (`xx` for language, `XX` for country) optionally followed by a dot and a character set identifier, for example, `UTF-8`.

`"C"` is a special locale that should always be available and defaults to American English.

## See Also

`format` for formatting objects of `tind` class.

## Examples

```
# current system locale
month_names()
weekday_names()
try(
ampm_indicators()
)

try({
# English abbreviated month names
print(month_names("en_GB"))
# French month names
print(month_names("fr_FR.UTF-8", FALSE))
# German abbreviated month names
print(month_names("de_DE.UTF-8"))
# Polish abbreviated month names
print(month_names("pl_PL.UTF-8"))
# English weekday names
print(weekday_names("en_GB", FALSE))
# French abbreviated weekday names
print(weekday_names("fr_FR.UTF-8"))
# German weekday names
print(weekday_names("de_DE.UTF-8", FALSE))
# Polish abbreviated weekday names
print(weekday_names("pl_PL.UTF-8"))
# US am/pm indicators
print(ampm_indicators("en_US"))
# UK am/pm indicators
print(ampm_indicators("en_GB"))
})
```

---

`calendars` *Working with Custom Calendars*

---

## Description

**tind** package provides an extensive collection of functions for calendrical computations allowing users to write custom calendar functions that can be used to mark business days, holidays, and other observances. See *Writing custom calendar functions* for an introduction to such functions and Examples for two real-life examples. These functions can later be used for pretty printing calendars on the console (using `calendar` function), quick identification of business days, holidays, and other observances (using `eval_calendar` function) and business day computations (see `bizday`).

`calendar` pretty prints a calendar for year(s) or month(s) on the console using user-provided calendar function determining business days and holidays.

`eval_calendar` applies user-provided calendar function to a sequence of dates. It is designed to be used by developers who wish to implement new applications of custom calendars.

## Usage

```
calendar(ym, calendar = NULL, name = NULL, locale = NULL)

eval_calendar(d, calendar)
```

## Arguments

| | |
|---|---|
| `ym` | an object of `tind` class or an R object coercible to it determining the year or month for which calendar is to be printed, current month (and preceding or following months) is the default. |
| `calendar` | a function determining working days and holidays (see Details), or `NULL`. |
| `name` | (optional) a character value (a short string) to be printed beside the year or month (or `NULL`). |
| `locale` | (optional) a character value determining locale or `NULL` (the default, interpreted as the current system locale), see calendar-names for information on locale settings. |
| `d` | an object of `tind` class or an R object coercible to it representing consecutive dates. |

## Details

`calendar` uses `crayon` package (when available) for highlighting dates. When calendar function (`calendar` argument) is not provided, Monday-Friday are marked as working days. Current date is marked by square brackets (`[]`).

For months, `calendar` additionally prints information about observances in a month provided that values in the list returned by the function passed as `calendar` argument are named.

## Value

`calendar` returns invisible `NULL` and is used for its side effects. `eval_calendar` returns a 3-element list of `tind` objects representing business days (`$bizdays`), holidays (`$holidays`), and other observances (`$otherobs`).

## Writing Custom Calendar Functions

Calendar function should take a vector of dates as an argument and return logical vector of the same length marking business days (as TRUE) or a list of two or three logical vectors of the same length with the first marking business days (as TRUE), the second marking holidays (as TRUE), and the third marking other observances / events (as TRUE). The second and the third returned logical vectors can be named indicating which observances are marked.

The 4 basic functions to be used when writing calendars are: year, month, day, and day_of_week. These can be used to mark fixed observances and weekends.

nth_dw_in_month and last_dw_in_month can be used to determine dates of movable observances falling on the nth or the last occurrence of a day of week in a particular month.

easter function can be used to determine date of Easter in a year as well as of other movable observances with a fixed distance from Easter.

Two examples of calendar functions are provided below. These two functions can be used as templates for developing custom calendar functions. In the examples, one will also find a programming trick to easily name holidays and other observances.

### A Note on the Design

One could argue that a design in which calendar functions should return logical vectors or lists of logical vectors is not intuitive and tind (vectors of dates) should be returned instead (thus making eval_calendar function redundant). Firstly, logical vectors are easy and fast to work with. By definition, a business day or a holiday must satisfy some conditions, which leads to logical values. Secondly, in some applications (for example counting business days) one would have to use matching to get integers or logical values back from time indices.

## Note

**tind** package does not provide a calendar for any country, region, or market. Instead, it gives users all the tools necessary to create customised calendars. See Examples section below for real-life examples of calendar functions, which could be used as templates.

## See Also

time-index-components, calendrical-computations, Ops, bizday.

## Examples

```
# US (federal) calendar with holiday names
calendar_US <- function(dd)
{
    dd <- as.tind(dd)
    y <- year(dd)
    m <- month(dd)
    d <- day(dd)
    newyear <- (m == 1) & (d == 1)
    martinlking <- (y >= 2000) & (m == 1) & (dd == nth_dw_in_month(3, 1, dd))
    presidentsday <- (m == 2) & (dd == nth_dw_in_month(3, 1, dd))
    memorialday <- (m == 5) & (dd == last_dw_in_month(1, dd))
    juneteenth <- (y >= 2021) & (m == 6) & (d == 19)
    independence <- (m == 7) & (d == 4)
```

```
    labor <- (m == 9) & (dd == nth_dw_in_month(1, 1, dd))
    columbus <- (m == 10) & (dd == nth_dw_in_month(2, 1, dd))
    veterans <- (m == 11) & (d == 11)
    thanksgiving <- (m == 11) & (dd == nth_dw_in_month(4, 4, dd))
    christmas <- (m == 12) & (d == 25)
    holiday <- newyear | martinlking | presidentsday |
               memorialday | juneteenth | independence |
               labor | columbus | veterans | thanksgiving |
               christmas
    # holiday names - a programming trick
    # names of holnms should be the same as names of logical vectors above
    names(holiday) <- rep("", length(holiday))
    holnms <- c(newyear = "New Year's Day",
                martinlking = "Birthday of Martin Luther King, Jr.",
                presidentsday = "Washington's Birthday",
                memorialday = "Memorial Day",
                juneteenth = "Juneteenth National Independence Day",
                independence = "Independence Day",
                labor = "Labor Day",
                columbus = "Columbus Day",
                veterans = "Veterans Day",
                thanksgiving = "Thanksgiving Day",
                christmas = "Christmas Day")
    lapply(names(holnms), function(nm) names(holiday)[get(nm)] <<- holnms[nm])
    # business days
    business <- !holiday & (day_of_week(dd) %in% 1:5)
    return (list(bizdays = business, holiday = holiday))
}

# Polish calendar from 1990 on with holiday names as well as other
# observances named
calendar_PL <- function(dd)
{
    dd <- as.tind(dd)
    y <- year(dd)
    m <- month(dd)
    d <- day(dd)
    # public holidays
    newyear <- (m == 1L) & (d == 1L)
    epiphany <- (y >= 2011L) & (m == 1L) & (d == 6L)
    easterd <- easter(dd) == dd
    eastermon <- easter(dd) + 1L == dd
    labour <- (m == 5L) & (d == 1L)
    constitution <- (m == 5L) & (d == 3L)
    pentecost <- easter(dd) + 49L == dd
    corpuschristi <- easter(dd) + 60L == dd
    assumption <- (m == 8L) & (d == 15L)
    allsaints <- (m == 11L) & (d == 1L)
    independence <- (m == 11L) & (d == 11L)
    christmaseve <- (m == 12L) & (d == 24L) & (y >= 2025)
    christmas <- (m == 12L) & (d == 25L)
    christmas2 <- (m == 12L) & (d == 26L)
    holiday <- newyear | epiphany |
```

```
            easterd | eastermon |
            labour | constitution |
            pentecost | corpuschristi |
            assumption |
            allsaints | independence |
            christmaseve | christmas | christmas2
# holiday names
names(holiday) <- rep("", length(holiday))
holnms <- c(newyear = "New Year", epiphany = "Epiphany",
             easterd = "Easter", eastermon = "Easter Monday",
             labour = "Labour Day", constitution = "Constitution Day",
             pentecost = "Pentecost", corpuschristi = "Corpus Christi",
             assumption = "Assumption of Mary",
             allsaints = "All Saints Day", independence = "Independence Day",
             christmaseve = "Christmas Eve",
             christmas = "Christmas", christmas2 = "Christmas (2nd day)")
lapply(names(holnms), function(nm) names(holiday)[get(nm)] <<- holnms[nm])
# working/business days
work <- !holiday & (day_of_week(dd) <= 5L)
# other observances
fatthursday <- easter(dd) - 52L == dd
shrovetuesday <- easter(dd) - 47L == dd
ashwednesday <- easter(dd) - 46L == dd
goodfriday <- easter(dd) - 2L == dd
primaaprilis <- (m == 4L) & (d == 1L)
flagday <- (m == 5L) & (d == 2L)
mothersday <- (m == 5L) & (d == 26L)
childrensday <- (m == 6L) & (d == 1L)
saintjohnseve <- (m == 6L) & (d == 23L)
allsoulsday <- (m == 11L) & (d == 2L)
saintandrewseve <- (m == 11L) & (d == 29L)
saintnicholasday <- (m == 12L) & (d == 6L)
christmaseve <- (m == 12L) & (d == 24L) & (y < 2025)
newyeareve <- (m == 12L) & (d == 31L)
other <- fatthursday | shrovetuesday | ashwednesday |
         goodfriday |
         primaaprilis |
         flagday |
         mothersday | childrensday | saintjohnseve |
         allsoulsday |
         saintandrewseve |
         saintnicholasday | christmaseve |
         newyeareve
names(other) <- rep("", length(other))
othernms <- c(fatthursday = "Fat Thursday",
              shrovetuesday = "Shrove Tuesday",
              ashwednesday = "Ash Wednesday",
              goodfriday = "Good Friday",
              primaaprilis = "All Fool's Day",
              flagday = "Flag Day",
              mothersday = "Mother's Day",
              childrensday = "Children's Day",
              saintjohnseve = "Saint John's Eve",
```

```
                    allsoulsday = "All Souls' Day",
                    saintandrewseve = "Saint Andrew's Eve",
                    saintnicholasday = "Saint Nicholas Day",
                    christmaseve = "Christmas Eve",
                    newyeareve = "New Year's Eve")
    lapply(names(othernms), function(nm) names(other)[get(nm)] <<- othernms[nm])

    return (list(work = work, holiday = holiday, other = other))
}

# print the calendar for the current and the previous/next month and the current year
# (Mon-Fri marked as working days)
calendar()
calendar(as.year(today()))

# print Polish and US calendars for 2020 and the current year
calendar(2020, calendar = calendar_PL)
calendar(2020, calendar = calendar_US)
calendar(as.year(today()), calendar = calendar_PL)
calendar(as.year(today()), calendar = calendar_US)

# print Polish and US calendars for 2020-01 and the current and the previous/next month
calendar("2020-01", calendar = calendar_PL)
calendar("2020-01", calendar = calendar_US)
calendar(calendar = calendar_PL)
calendar(calendar = calendar_US)

# get list of business days, holidays for 2020-01 and the current month
# using Polish and US calendars
d202001 <- seq(as.date("2020-01-01"), "2020-01-31")
dcurrmnth <- seq(floor_t(today(), "m"), last_day_in_month(today()))
eval_calendar(d202001, calendar_PL)
eval_calendar(d202001, calendar_US)
eval_calendar(dcurrmnth, calendar_PL)
eval_calendar(dcurrmnth, calendar_US)

# print calendars with names
calendar(calendar = calendar_PL, name = "PL")
calendar(calendar = calendar_US, name = "US (federal)")

# print Polish calendar using Polish locale
try(
calendar(calendar = calendar_PL, locale = "pl_PL.UTF-8")
)
```

---

calendrical-computations

*Calendrical Computations in* **tind** *Package*

---

## Description

The following functions can be used for calendrical computations, especially determining dates of movable observances. All function are vectorised.

`nth_day_of_year` returns the date of the nth day of a year.

`last_day_in_month` and `last_day_in_quarter` return the date of the last day in a month or a quarter.

`nth_dw_in_month` returns the date of the nth day of week in a month.

`last_dw_in_month` returns the date of the last day of week in a month.

`nth_dw_after` and `nth_dw_before` calculate the nth occurrence of a day of week after or before given date.

`easter` returns the date of Easter in a year.

## Usage

```
nth_day_of_year(nth, y)

last_day_in_month(m)

last_day_in_quarter(q)

nth_dw_in_month(nth, dw, m)

last_dw_in_month(dw, m)

nth_dw_after(nth, dw, d)

nth_dw_before(nth, dw, d)

easter(y)
```

## Arguments

| | |
|---|---|
| `nth` | a numeric value or vector of indices (1–366 for `nth_day_of_year`, 1–5 for `nth_dw_in_month`). A positive integer (vector) for `nth_dw_after` and `nth_dw_before`. |
| `y`, `q`, `m`, `d` | an object of `tind` class or an R object coercible to it. |
| `dw` | a numeric value or vector of days of week (values in range 1–7 with Monday as the 1st day). |

## Value

An object of `tind` class with dates (type `"d"`).

## See Also

time-index-components, time-index-properties, Ops. Further examples of application of these functions can be found in `calendar` documentation. For calendrical computations involving business days see `bizday`.

## Examples

```
# Thanksgiving in the US is observed on the fourth Thursday of November,
# which in 2019 was on:
nth_dw_in_month(4, 4, 201911)
# and Black Friday?
nth_dw_in_month(4, 4, 201911) + 1

# Daylight Saving Time in the EU in 2019 began on the last Sunday in March,
# which was on:
last_dw_in_month(7, 201903)

# International Monetary Market dates in 2022 - 3rd Wednesday
# of March, June, September, and December
nth_dw_in_month(3, 3, tind(y = 2022, m = 3 * 1:4))

# determine frequencies of Easter months over the last 100 years
# Easter months
em <- month(easter(as.year(today()) + (-99:0)), labels = TRUE)
# table and barplot
table(em) / length(em) * 100
if (require("graphics", quietly = TRUE)) {
    barplot(table(em) / length(em) * 100, ylim = c(0, 100), col = "#faf06d")
}
```

---

# current-date-time *Current Date and Time*

---

## Description

today returns the current date and now returns the current date and time (in the system time zone or the time zone provided by the user).

## Usage

```
today(tz = NULL)

now(tz = NULL, digits = 0)
```

## Arguments

| | |
|---|---|
| tz | (optional) a character value determining the time zone (the default NULL is interpreted as the system time zone). See tzone documentation for information on time zones. |
| digits | an integer value giving the number of decimal places for seconds (0–6, 0 by default). |

## Value

today and now return an object of class tind of length 1 and type "d" (date) and "t" (date-time), respectively.

## Examples

```
today()
now()
# millisecond accuracy
now(digits = 3)
# check current date and time in different time zones
if ("Asia/Tokyo" %in% OlsonNames()) {
now("Asia/Tokyo")
today("Asia/Tokyo")
}
if ("Europe/Warsaw" %in% OlsonNames()) {
now("Europe/Warsaw")
today("Europe/Warsaw")
}
if ("America/New_York" %in% OlsonNames()) {
now("America/New_York")
today("America/New_York")
}
```

---

# cut *Group Time Indices into Periods / Convert to a Factor*

---

## Description

cut method for objects of tind class allows to map / group time indices into periods. The periods can be determined based on indices provided by the user or by (multiples of) units of time.

## Usage

```
## S3 method for class 'tind'
cut(x, breaks, labels = TRUE, right = FALSE, ...)
```

## Arguments

| | |
|---|---|
| x | an object of tind class. |
| breaks | a numeric value or a character string determining intervals, or an object of tind class with cut points, see Details. |
| labels | a logical value controlling the return type, which can be a factor (if TRUE, the default), integer vector, or a 2-element list. |
| right | a logical value determing whether indices should be matched to the closest left cut point or to the closest right cut point, see Details. |
| ... | (ignored) further arguments passed to or from other methods. |

## Details

breaks argument controls how indices are grouped. It can be a number or a character string determining resolution (or an object of tdiff class). Alternatively, breaks can be an object of tind class with cut points.

When breaks determines resolution, only selected multiples of units are allowed, similarly to floor_t function. Documentation of admissible units and multiples can be found in Details section of resolution_t method documentation. If selected resolution corresponds to an index of different type (for example grouping dates to 2-month periods), conversion takes place.

This method differs from cut.POSIXt and cut.Date in two aspects. Firstly, the periods are selected differently, they are always aligned to resolution, see Examples. Secondly, as it does not rely on seq but rounding of indices, the levels may be discontinuous. If users want to replicate behaviour of cut from **base**, they should provide tind constructed via seq.tind method as breaks argument.

When breaks is a tind object, it is expected to be sorted without NAs. By default, indices in x are matched to the closest index to the left (largest index that is not greater than the argument). If right is set to TRUE, indices are matched to the closest index to the right (smallest index that is not smaller than the argument). right cannot be set to TRUE if breaks is not a tind. It is acceptable that breaks is of lower resolution than x provided that x is convertible to it. In such situations, right cannot be set to TRUE.

By default, cut.tind returns a factor with levels created using as.character method. If labels argument is set to FALSE, only the integer vector (of the same length as argument) of mappings to intervals is returned (as in **base** method). If set to NA, a 2-element list is returned, with integer vector of mappings as the first element and time indices determining intervals (grouping, levels) as the second. labels can only take TRUE/FALSE/NA values.

## Value

A factor if labels is TRUE, an integer vector if FALSE, and a 2-element list if NA, see Details.

## See Also

rounding and resolution_t for description of admissible units and multiples that can be used for breaks argument. match_t for matching time indices to other indices and time intervals.

## Examples

```
# basic use
(d <- seq.tind("2023-09-14", "2023-12-16"))
cut(d, "15d")
cut(d, "m")
cut(d, "2m")
# tind given as breaks
cut(d, as.date(c("2023-09-01", "2023-11-16", "2023-12-16")))
cut(d, seq.tind("2023-01", "2023-12"))
# random order with NAs
(d <- sample(c(d, NA)))
cut(d, "15d")
cut(d, "m")
cut(d, "2m")
```

```
# different behaviour of cut for tind and Date (alignment to 2 month resolution,
# which means Jan, Mar, May, Jul, Sep, Nov)
(d <- seq.tind("2023-12-16", "2024-03-01"))
cut(d, "2 months")
cut(as.Date(d), "2 months")
# replicate behviour of cut.Date by providing sequence of months
cut(d, seq.tind("2023-12", "2024-03", by = "2m"))
# same
cut(d, seq.tind(as.month(min(d)), as.month(max(d)), by = "2m"))
# check
all.equal(cut(as.Date(d), "2 months", labels = FALSE),
          cut(d, seq.tind("2023-12", "2024-03", by = "2m"), labels = FALSE))
```

---

`date2num` *Conversion between Dates and Their Integer Representations*

---

## Description

`date2num` and `num2date` support conversion between `tind` dates and integer representations of dates (days since ...) found in different software packages.

## Usage

```
date2num(x, format)

num2date(x, format)
```

## Arguments

| | |
|---|---|
| `x` | a `tind` with dates or an integer vector. |
| `format` | a character value determing numeric representation of date; currently, the following are supproted: `"R"`, `"MATLAB"`, `"Excel"`, `"SAS"`, `"JDN"` (Julian Day Number). |

## Value

`date2num` returns an integer vector and `num2date` returns `tind` representing dates.

## See Also

`jdn` for description of Julian Day Numbers.

## Examples

```
(td <- today())
fmts <- c("R", "MATLAB", "Excel", "SAS", "JDN")
(n <- sapply(fmts, function(fmt) date2num(td, fmt)))
lapply(fmts, function(fmt) num2date(n[fmt], fmt))
```

`date_time` *Construct Date-Time Indices from Date and Time Components*

## Description

`date_time` can be used to create date-time indices from its components: date and time of day (hour, minute, and second).

`date_time_split` performs the opposite computation: given date-time indices, it returns a two-element list with vectors of dates and times.

## Usage

```
date_time(d, H, M, S, tz = NULL, grid = FALSE)

date_time_split(x)
```

## Arguments

| | |
|---|---|
| `d` | an object of `tind` class of type date (type `"d"`) or an R object coercible to it. |
| `H` | a numeric vector with hour values or an R object coercible to time index of time-of-day type (type `"h"`). |
| `M` | (optional) a numeric vector with minutes. |
| `S` | (optional) a numeric vector with seconds. |
| `tz` | (optional) a character value determining the time zone (the default `NULL` is interpreted as the system time zone). See `tzone` documentation for information on time zones. |
| `grid` | a logical value, if `TRUE` date-time indices are constructed from all combinations of dates and times (`FALSE` by default). |
| `x` | an object of `tind` class of type date-time (type `"t"`) or an R object coercible to it. |

## Details

If arguments of `date_time` are of different length, they are recycled.

When `grid` is set to `TRUE`, date-time indices are constructed from all combinations of dates and times in a way similar to how functions `expand.grid` and `kronecker` work, see Examples.

If `H` argument is numeric, time of day is constructed from `H`, `M`, and `S` arguments. In the last step date and time are combined in order to construct date-time index. If `H` is not numeric, `M` and `S` should not be supplied and time of day is constructed from `H` argument only.

When provided without `H` argument `date_time` behaves just like `as.date_time` i.e. returns the beginning of a day.

When an hour occurs twice in a day (due to DST/UTC offset changes), the second occurrence is selected with a warning. When hour is missing (for the same reason), `NA` is returned with a warning. See Examples.

## Value

`date_time` returns an object of `tind` class with date-time indices (type `"t"`). `date_time_split` returns a two-element list with vectors of dates (`$date`) and times (`$time`).

## See Also

`tind` constructor, time-index-components, `tzone`.

## Examples

```
date_time(today() + (0:1), "11:25:20.75")
date_time(today() + (0:1), as.time("11:25:20.75"))
date_time(today() + (0:1), 11, 25, 20.75)
date_time(today() + (0:1), "11:25:20")
date_time(today() + (0:1), 11, 25, 20)
date_time(today() + (0:1), "11:25")
date_time(today() + (0:1), 11, 25)
date_time(today() + (0:1), "11")
date_time(today() + (0:1), 11)
date_time(today() + (0:1))
# using 'grid' argument
date_time(today() + 0:2, c(8, 12, 16))
date_time(today() + 0:2, c(8, 12, 16), grid = TRUE)

# split date-time
(nw <- now())
date_time_split(nw)

# corner cases (with warnings)
if ("Europe/Warsaw" %in% OlsonNames()) try({
    # 2020-10-25 had 25h with 02:00 repeated
    date_time("2020-10-25", 0:2, tz = "Europe/Warsaw")
})
if ("Europe/Warsaw" %in% OlsonNames()) try({
    # 2021-03-28 had 23h with 02:00 missing
    date_time("2021-03-28", 0:2, tz = "Europe/Warsaw")
})
```

---

# daycount_frac *Differences Between Dates as Year Fractions / Accrual Factors*

---

## Description

This function computes difference between two dates as year fraction given day count convention.

## Usage

```
daycount_frac(d1, d2, convention)
```

## Arguments

| | |
|---|---|
| d1 | an object of tind class representing start date(s) or an R object coercible to it. |
| d2 | an object of tind class representing end date(s) or an R object coercible to it. |
| convention | a chacter string determining day count convention to be used, see Details. |

## Details

Currently, the following day count conventions are supported:

30/360 also known as 30/360 Bond Basis or 360/360, described in ISDA 2006 Section 4.16(f). The formula is as follows:

$$\frac{360(y_2 - y_1) + 30(m_2 - m_1) + (d_2 - d_1)}{360},$$

where $y$ denotes year, $m$ month, $d$ day of month. Dates are adjusted accoring to the following rules: if $d_1$ is 31, it is changed to 30, if $d_2$ is 31 and $d_1$ is 30 or 31, $d_2$ is changed to 30.

30E/360 also known as Eurobond basis, 30/360 ICMA or 30/360 ISMA, described in ICMA Rule 251.1(ii), 251.2 and ISDA 2006 Section 4.16(g). The formula used is the same as above but the adjustment of dates is different: if $d_1$ or $d_2$ is 31, it is changed to 30.

ACT/ACT also known as Actual/Actual and Actual/Actual ISDA, described in ISDA 2006 Section 4.16(b). Days between the dates (start included, end excluded) are divided into two groups: falling in leap and non-leap years. The number of days in leap years is divided by 366, the number of days in non-leap years is divided by 365. Finally, the two fractions are added.

ACT/365F also known as Actual/365 Fixed, described in ISDA 2006 Section 4.16(d). Difference in days between dates divided by 365.

ACT/360 also known as Actual/360, described in ISDA 2006 Section 4.16(e). Difference in days between dates divided by 360.

## Value

A numeric vector.

## References

International Swaps and Derivatives Association, Inc., *2006 ISDA Definitions*, New York, 2006.

## See Also

year_frac, bizday.

## Examples

```
daycount_frac("2023-01-29", "2023-03-31", "30/360")
1/6 + 2/360
daycount_frac("2023-01-29", "2023-03-31", "30E/360")
1/6 + 1/360
daycount_frac("2023-01-29", "2023-03-31", "ACT/ACT")
61 / 365
```

```
daycount_frac("2024-01-29", "2024-03-31", "ACT/ACT")
62 / 366
daycount_frac("2023-01-29", "2023-03-31", "ACT/365F")
61 / 365
daycount_frac("2024-01-29", "2024-03-31", "ACT/365F")
62 / 365
daycount_frac("2023-01-29", "2023-03-31", "ACT/360")
61 / 360
daycount_frac("2024-01-29", "2024-03-31", "ACT/360")
62 / 360
```

---

`diff` *Lagged Differences for* `tind` *and* `tdiff` *Objects*

---

## Description

`diff` method for `tind` and `tdiff` works in a standard way. For all index types except for integer and numeric indices, differences of time indices are retuned as objects of `tdiff` class.

## Usage

```
## S3 method for class 'tind'
diff(x, lag = 1L, differences = 1L, ...)

## S3 method for class 'tdiff'
diff(x, lag = 1L, differences = 1L, ...)
```

## Arguments

| | |
|---|---|
| `x` | an object of `tind` class or `tdiff` class. |
| `lag` | an integer value. |
| `differences` | an integer value. |
| `...` | (ignored) further arguments passed to or from other methods. |

## Value

An object of `tdiff` class, except for `x` argument of `tind` class of type `"i"` or `"n"` (integer/numeric indices), in which case an integer or numeric vector is returned.

## See Also

Ops.

## Examples

```
(nn <- sample(1:10))
(x <- today() + nn)
# all 3 should be the same
diff(x, 2, 2)
as.tdiff(diff(nn, 2, 2), "d")
diff(as.tdiff(nn, "d"), 2, 2)
```

---

`format` *Conversion between Objects of* `tind` *Class and Character Vectors*

---

## Description

`format` method converts objects of `tind` class to character vectors given format and locale information. `strptind` function accepts character vector with time indices and parses to create object of `tind` class.

## Usage

```
## S3 method for class 'tind'
format(x, format, locale = NULL, ...)

strptind(x, format, locale = NULL, type = NULL, tz = NULL)
```

## Arguments

| | |
|---|---|
| `x` | an object to be converted, a character vector for `strptind`, an object of `tind` class for `format`. |
| `format` | a character string or character vector determining string format(s) (see Details). |
| `locale` | a character value determining locale to be used for `%a`, `%A`, `%b`, `%B`, and `%p` specifiers (month names, weekday names, and AM/PM indicators) or `NULL` (default, interpreted as the current system locale), see calendar-names for information on locale settings. |
| `...` | (ignored) further arguments passed to or from other methods. |
| `type` | (optional) a character value determining time index type. |
| `tz` | (optional) a character value determining the time zone (the default `NULL` is interpreted as the system time zone). See `tzone` documentation for information on time zones. |

## Details

Names of accepted format specifiers except for `%q` are conformant with those used by `format.Date`, `format.Date` and `strptime`. Accepted specifiers are listed below:

`%a` Abbreviated weekday name.

`%A` Weekday name.

`%b` Abbreviated month name.

`%B` Month name.

`%d` Day of month (01–31).

`%e` Day of month (1–31) with a leading space for a single-digit number.

`%D` American/C99 date representation `%m/%d/%y`.

`%F` ISO 8601 date `%Y-%m-%d`.

`%g` The last two digits of the week-based year.

`%G` The week-based year.

`%H` Hour (00–23).

`%I` Hour, 12-hour clock (1–12).

`%p` AM/PM indicator.

`%j` Day of year (001–366).

`%m` Month (01–12).

`%M` Minute (00–59).

`%n` Newline.

`%OS[n]` Second with `n` (0–6) decimal places.

`%OS` Second with up to 6 decimal places (automatically detected precision during parsing).

`%q` (Not supported by base R.) Quarter (1–4).

`%R` Same as `%H:%M`.

`%S` Second (00–59), leap seconds are not accepted on input.

`%t` Tab or whitespace.

`%T` Same as `%H:%M:%S`.

`%u` Weekday (1–7) with Monday as the first day in a week (ISO 8601).

`%V` Week (01–53) as defined in ISO 8601.

`%y` 2-digit year (00–99), values 00–68 are prefixed by 20 and 69–99 by 19.

`%Y` 4-digit year (0–9999), 0 is allowed by ISO 8601.

`%z` Signed offset in hours and minutes from UTC, accepted input formats are `+-HHMM`, `+-HH`, `+-HH:MM`, and letter `Z` for UTC.

`%Z` Time zone abbreviation (also supported on input).

On very rare occasions (the need to use formats unsupported by `strptind`) users will have to call `strptime` and then `as.tind` method or perform some regex preprocessing before calling `strptind`.

`type` argument is optional as `strptind` automatically determines index type from components. However, it can be set as a safeguard against format misspecifications.

## Value

`strptime` returns an object of `tind` class, `format` returns a character vector.

## Note

The following strptime specifiers (as well as some others) are not supported (most often because they are locale specific or do not comply with ISO 8601):

`%c` Locale-specific date and time.

`%C` Century (00–99).

`%h` Equivalent to `%b`.

`%r` 12-hour clock time using AM/PM indicator.

`%U` Week of the year (US convention).

`%w` Weekday (0–6).

`%W` Week of the year (UK convention).

`%x` Locale-specific date.

`%X` Locale-specific time.

## See Also

parse_t for easier to use index parsing requiring order specification only, calendar-names for information on locale settings.

## Examples

```
## years
# four-digit year
(ti <- strptind(as.character(1998:2002), "%Y"))
format(ti, "%Y")
# two-digit year
(ti <- strptind(c("98", "99", "00", "01", "02"), "%y"))
format(ti, "%y")
# mixture of four-digit and two-digit years
strptind(c("1998", "1999", "00", "01", "02"), c("%Y", "%y"))

## quarters
(ti <- strptind(c("2020Q1", "2020Q2", "2020Q3", "2020Q4"), "%YQ%q"))
format(ti, "%YQ%q")
format(ti, "%Yq%q")
format(ti, "%Y.%q")

## months
(ti <- strptind(c("2020-03", "2020-06", "2020-09", "2020-12"), "%Y-%m"))
format(ti, "%Y-%m")
(ti <- strptind(c("03/20", "06/20", "09/20", "12/20"), "%m/%y"))
format(ti, "%m/%y")
format(ti, "%b '%y")

## weeks
(ti <- strptind(c("2020-W01", "2020-W05", "2020-W09", "2020-W13"), "%G-W%V"))
format(ti, "%G-W%V")
format(ti, "%G, week: %V")
strptind(c("2020, week: 13"), "%G, week: %V")
```

```
## dates
# ISO format
(ti <- strptind(c("2025-03-19", "2025-06-18", "2025-09-17", "2025-12-17"), "%F"))
format(ti, "%F")
strptind(c("2025-03-19", "2025-06-18", "2025-09-17", "2025-12-17"), "%Y-%m-%d")
format(ti, "%Y-%m-%d")
# US format
strptind(c("03/19/25", "06/18/25", "09/17/25", "12/17/25"), "%D")
format(ti, "%D")
strptind(c("03/19/25", "06/18/25", "09/17/25", "12/17/25"), "%m/%d/%y")
format(ti, "%m/%d/%y")
# European format
strptind(c("19.03.2025", "18.06.2025", "17.09.2025", "17.12.2025"), "%d.%m.%Y")
format(ti, "%d.%m.%Y")
# mixed formats
strptind(c("03/19/25", "06/18/25", "17.09.2025", "17.12.2025"),
         c("%m/%d/%y", "%d.%m.%Y"))
strptind(c("03/19/25", "06/18/25", "2025-09-17", "2025-12-17"),
         c("%D", "%F"))

## time of day
(ti <- strptind("13:03:34.534", "%H:%M:%OS"))
format(ti, "%H:%M:%OS3")
format(ti, "%H:%M:%OS2")
format(ti, "%H:%M:%OS1")
strptind("13:03:34", "%H:%M:%S")
format(ti, "%H:%M:%S")
strptind("13:03", "%H:%M")
format(ti, "%H:%M")
strptind("13", "%H")
format(ti, "%H")
strptind("01:03:44 pm", "%I:%M:%S %p")
format(ti, "%I:%M:%S %p")
strptind("1:03:44 pm", "%I:%M:%S %p")
strptind(c("1am", "1pm"), "%I%p")

## date-time
(ti <- strptind("2025-02-01 13:03:34.534", "%F %H:%M:%OS"))
format(ti, "%F %H:%M:%S")
format(ti, "%F %H:%M:%OS2")
format(ti, "%F %H:%M:%S%z")
format(ti, "%F %H:%M:%OS2 %Z")
strptind("02/01/25 01:03:34pm", "%D %I:%M:%OS%p")
```

---

jdn *Date and Date-Time Conversion to and from Julian Day Number (JDN)*

---

## Description

jdn computes the JDNs for dates or date-time indices and jdn2tind returns dates or date-time indices given JDNs.

For date arguments jdn returns the numbers of days since November 24, 4714 BC in the proleptic Gregorian calendar as an integer vector. For date-time arguments day fraction in UTC is computed and the return value is a numeric vector.

For integer arguments jdn2tind return tind of type "d" (date), for non-integer arguments — tind of type "t" (date-time). If tz argument is provided the return value is always of type "t" (date-time).

## Usage

```
jdn(x)

jdn2tind(x, tz = NULL)
```

## Arguments

| | |
|---|---|
| x | an object of tind class or an R object coercible to it for jdn, an integer or numeric vector for jdn2tind. |
| tz | (optional) a character value determining the time zone (the default NULL is interpreted as the system time zone). See tzone documentation for information on time zones. |

## Value

An integer or numeric vector for jdn, an object of tind class (type "d" or "t") for jdn2tind.

## Note

For date-time indices JDN is computed based on day fraction since noon UTC and not midnight, so 0.5 offset will be observable.

## See Also

date2num for conversion between dates and their integer representations found different software packages.

## Examples

```
# JDN of 2000-01-01 is 2451545
jdn("2000-01-01")
jdn2tind(2451545)
# JDN today, now?
jdn(today())
jdn(now())
# notice the .5 offset
jdn(today(tz = "UTC"))
format(jdn(as.date_time(today(tz = "UTC"), tz = "UTC")), digits = 8)
```

match_t *Matching Time Indices*

## Description

match_t and %in_t% allow for matching time indices to time intervals and to other sets of time indices including cases when table argument is of different type than x (of lower resolution).

## Usage

```
match_t(x, table, nomatch = NA_integer_)

x %in_t% table
```

## Arguments

| | |
|---|---|
| x | an object of tind class. |
| table | an object of tinterval or tind class. |
| nomatch | an integer value to be returned when no match is found. |

## Details

%in_t% always returns TRUE/FALSE. NAs in x argument are *never* matched (FALSE is returned).

## Value

match_t and %in_t% return integer and logical vectors, respectively. The length of the result equals length of x.

## Note

Since match and %in% are not implemented in **base** as S3 generics, new functions had to be implemented.

## Examples

```
# match dates to months
(x <- as.date("2025-03-02") + 15 * (0:5))
(table <- as.month("2025-03") + -1:1)
match_t(x, table)
# match dates to time intervals representing months
(table <- (as.date("2025-03-01") %--% as.date("2025-03-31")) %+m% (-1:1))
match_t(x, table)
# are dates in March 2025?
x %in_t% "2025-03"
# NAs are _never_ matched
(x <- as.date("2025-03-02") + c(NA, 15 * (0:5)))
(table <- as.month("2025-03") + c(NA, -1:1))
```

```
match_t(x, table)
x %in_t% table
```

---

`merge` *Merging Time-indexed Data*

---

## Description

`merge` method for `tind` allows to join two (or more) time-indexed datasets also in cases when the indices are of different types. The method is intended for advanced users.

The method takes two `tind` vectors (`x` and `y`) and returns a three-element list containing resulting indices and mappings (integer indices) from the original indices to the final ones allowing to select appropriate rows from dataset indexed by `x` and `y`, see Examples.

## Usage

```
## S3 method for class 'tind'
merge(x, y, ..., all = FALSE, all.x = all, all.y = all)
```

## Arguments

| | |
|---|---|
| `x, y` | an object of `tind` class. |
| `...` | (optional) further time indices. |
| `all` | a logical value, equivalent to setting both `all.x` and `all.y` to the same value. Alternatively, a logical vector in case of more than 2 arguments. See Details. |
| `all.x` | a logical value, if `TRUE`, all `x` observations are included in the result even if there are no corresponding time indices in `y`. |
| `all.y` | a logical value, analogous to `all.x`. |

## Details

By default (`all = FALSE`), inner join is performed. `x` and `y` can be indices of different types but conversion of the higher resolution to the lower should be possible.

If `all.x = TRUE`, left join is performed. All indices from `x` are preserved. `y` can then be of the same or lower resolution than `x`.

If `all.y = TRUE`, right join is performed. All indices from `y` are preserved. `x` can then be of the same or lower resolution than `y`.

If `all = TRUE`, outer join is performed. All indices from `x` and `y` are preserved. Indices in `x` and `y` have to be of the same type in this case.

Setting `all` argument silently overrides both `all.x` and `all.y`.

`NA`s are *never* matched.

The method is optimized in case both indices are strictly increasing without `NA`s (time series applications). In other cases, it employs `merge` method for specially constructed data frames.

The method also accepts more than two arguments (time indices). In this case, it is expected that all are strictly increasing without NAs (time series applications only). `all.x` and `all.y` cannot be used with more than two arguments.

`all` can be a vector of logical values indicating which indices have to always be included in the result (`TRUE`) and which have to be matched (`FALSE`). In 2-argument case, for example, `all = c(TRUE, FALSE)` is equivalent to `all.x = TRUE` and `all = c(FALSE, TRUE)` to `all.y = TRUE`.

## Value

A three-element list with the first element (`index`) containing the final time indices, and the remaining two (`xi` and `yi`) mappings from `x` and `y` to these indices. If additional time indices are provided, the length of the returned list equals the number of all arguments (including `x` and `y`) plus one (for the final index at the beginning of the list).

## See Also

`match_t` for matching time indices.

## Examples

```
# construct sample data frames
(dates1 <- tind(y = 2023, m = rep(1:4, each = 2), d = c(1, 16)))
(dates2 <- dates1 %+m% 1)
(mnths <- as.month("2022-12") + 0:3)
(df1 <- data.frame(dates1, nd1 = as.numeric(dates1),
                   downame = day_of_week(dates1, labels = TRUE, abbreviate = FALSE)))
(df2 <- data.frame(dates2, nd2 = as.numeric(dates2), dow = day_of_week(dates2)))
(df3 <- data.frame(mnths, nm = as.numeric(mnths),
                   mname = month(mnths, labels = TRUE, abbreviate = FALSE)))
# inner join - dates
(mti <- merge(df1[[1L]], df2[[1L]]))
data.frame(index = mti[[1L]],
           df1[mti[[2L]], -1L, drop = FALSE],
           df2[mti[[3L]], -1L, drop = FALSE])
# inner join - dates and months
(mti <- merge(df1[[1L]], df3[[1L]]))
data.frame(index = mti[[1L]],
           df1[mti[[2L]], -1L, drop = FALSE],
           df3[mti[[3L]], -1L, drop = FALSE])
# left join - dates
(mti <- merge(df1[[1L]], df2[[1L]], all.x = TRUE))
data.frame(index = mti[[1L]],
           df1[mti[[2L]], -1L, drop = FALSE],
           df2[mti[[3L]], -1L, drop = FALSE])
# left join - dates and months
(mti <- merge(df1[[1L]], df3[[1L]], all.x = TRUE))
data.frame(index = mti[[1L]],
           df1[mti[[2L]], -1L, drop = FALSE],
           df3[mti[[3L]], -1L, drop = FALSE])
# right join - dates
(mti <- merge(df1[[1L]], df2[[1L]], all.y = TRUE))
```

```
data.frame(index = mti[[1L]],
           df1[mti[[2L]], -1L, drop = FALSE],
           df2[mti[[3L]], -1L, drop = FALSE])
# right join - months and dates
(mti <- merge(df3[[1L]], df2[[1L]], all.y = TRUE))
data.frame(index = mti[[1L]],
           df3[mti[[2L]], -1L, drop = FALSE],
           df2[mti[[3L]], -1L, drop = FALSE])
# outer join - dates
(mti <- merge(df1[[1L]], df2[[1L]], all = TRUE))
data.frame(index = mti[[1L]],
           df1[mti[[2L]], -1L, drop = FALSE],
           df2[mti[[3L]], -1L, drop = FALSE])
```

---

`Ops` *Operations Involving Time Indices, Time Differences, and Time Intervals*

---

## Description

Basic arithmetic and comparison operators are implemented for time indices, time differences, and time intervals where applicable.

Operators `+` and `-` allow for shifting time indices and computing differences between two indices. Time intervals can be shifted using these, too. When the second operand in `+` and `-` is numeric the underlying time unit is used. For time of day and date-time indices this is always a second.

Convenience operators `%+y%`, `%-y%`, `%+q%`, `%-q%`, `%+m%`, `%-m%`, `%+w%`, `%-w%`, `%+d%`, `%-d%`, `%+h%`, `%-h%`, `%+min%`, `%-min%`, `%+s%`, and `%-s%` can used to shift time indices (and intervals) by years, quarters, months, weeks, days, hours minutes, and seconds. See Details for their behaviour in corner cases.

For all index types except for integer and numeric indices differences between time indices are retuned as objects of `tdiff` class.

Comparison operators are available for time indices (`tind`) and time differences (`tdiff`).

## Usage

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

e1 %+y% e2

e1 %-y% e2

e1 %+q% e2

e1 %-q% e2

e1 %+m% e2
```

```
e1 %-m% e2

e1 %+w% e2

e1 %-w% e2

e1 %+d% e2

e1 %-d% e2

e1 %+h% e2

e1 %-h% e2

e1 %+min% e2

e1 %-min% e2

e1 %+s% e2

e1 %-s% e2
```

## Arguments

| | |
|---|---|
| e1, e2 | a tind, tdiff, tinterval, or a numeric vector. |

## Details

One can only subtract from time indices and divide time differences.

Unary + and - operators are supported for tdiff only.

Results of arithmetic operations are always validated and can become NAs (when out of range) or be rounded, for example, when dividing time differences in days by numbers that are not divisors.

Shifting time intervals beyond valid index ranges can lead to spurious results as beginnings or ends of time intervals become NAs and intervals can become entire line.

== and != operators for time indices only accept same types of indices. The remaining comparison operators accept different types provided that conversion can be performed.

### Corner Cases

When shifting dates by months one is faced with a dilemma: should month after March 31st be April 30th or May 1st? The convention in **tind** is that the result of %+m% n always falls in the nth month after the month in which a given date falls and in corner cases the last day in the resulting month is returned. Similar logic is applied to shifts by years and quarters as well as shifts of weeks by years (some years have 53 weeks). See Examples.

Shifting date-time indices by days can also be problematic in front of DST changes, when the resulting date has 23 hours (one hour missing) or 25 hours (one hour repeated). When hour is missing, the next hour is selected (no NA is returned). When hour is doubled, the second occurrence is selected. See Examples.

## Value

Comparison operators and ! return logical vectors.

Differences of time indices are returned as objects of `tdiff` class, except for arguments of type `"i"` or `"n"` (integer/numeric indices), in which case an integer or numeric vector is returned.

Shifting time indices and time intervals produces time indices and time intervals, respectively.

Operations involving time differences return time differences.

## See Also

`tind` class and its constructor, `calendrical-computations` for calendrical computations.

## Examples

```
# list the last 10 days including today
today() + (-9:0)

# how many days have passed since the beginning of the year?
today() - floor_t(today(), "y")
# same but the result is not tdiff
day_of_year(today()) - 1

# single time interval
x <- "2024-06-01 08:00" %--% "2024-06-01 16:00"
# shift by 0, 1, ..., 5 days
x %+d% 0:5

# are we in or after 2026?
today() >= 2026
# are we after 2025?
today() > 2025

# corner cases - ends of months and shifts by months
as.date("2024-01-31") %+m% 0:5
# same
as.date("2024-01-31") + mnths(0:5)

# corner cases - 53rd week of year
as.week(202053) %-y% 0:5

# corner cases - DST changes and shifts by days
if ("Europe/Warsaw" %in% OlsonNames()) {
    # 2020-10-25 had 25h with 02:00 repeated
    print(as.date_time("2020-10-24 02:00", tz = "Europe/Warsaw") %+h% 23:26)
    print(as.date_time("2020-10-24 02:00", tz = "Europe/Warsaw") %+d% 1)
    print(as.date_time("2020-10-26 02:00", tz = "Europe/Warsaw") %-h% 26:23)
    print(as.date_time("2020-10-26 02:00", tz = "Europe/Warsaw") %-d% 1)
    # 2021-03-28 had 23h with 02:00 missing
    print(as.date_time("2021-03-27 02:00", tz = "Europe/Warsaw") %+h% 22:25)
    print(as.date_time("2021-03-27 02:00", tz = "Europe/Warsaw") %+d% 1)
    print(as.date_time("2021-03-29 02:00", tz = "Europe/Warsaw") %-h% 25:22)
    print(as.date_time("2021-03-29 02:00", tz = "Europe/Warsaw") %-d% 1)
```

```
}
```

---

`ordered-regular` *Working with Ordered and Regularly Spaced Time Indices*

---

## Description

`is.ordered_t` method checks if time indices form a strictly increasing sequence without `NA` values.

`is.regular` method checks if time indices form a strictly increasing, regularly spaced sequence without `NA` values.

`as.regular` returns regularly spaced sequence of time indices based on strictly increasing time indices provided.

`extend_regular` extends strictly increasing sequence of time indices by `n` points after the last taking into account the resolution of the sequence provided.

## Usage

```
is.ordered_t(x)

## S3 method for class 'tind'
is.ordered_t(x)

is.regular(x)

## S3 method for class 'tind'
is.regular(x)

as.regular(x, ...)

## S3 method for class 'tind'
as.regular(x, ...)

extend_regular(x, n)
```

## Arguments

| | |
|---|---|
| `x` | an object of `tind` class or of other time index class supported by **tind** package. |
| `...` | further arguments passed to or from other methods. |
| `n` | an integer value, number of time stamps to be added, see Details. |

## Details

`n` argument of `extend_regular` can be negative. In that case `-n` points are added before the first index. The function may fail in corner cases (if the result would be out of range).

Creating regular date-time sequences in front of DST/UTC offset changes can be impossible. If the algorithm fails, an error is issued. In general, this should not be a problem when DST change is by 1 hour and the resolution of the indices is 1 hour or higher.

## Value

A logical value for `is.ordered_t` and `is.regular`. An object of `tind` class for `as.regular` and `extend_regular`.

## See Also

`resolution_t` method.

## Examples

```
# months, resulution 2m
(ms <- tind(y = 2023, m = 1 + 2 * (0:5)))
is.regular(ms)
extend_regular(ms, 3)
(ms <- tind(y = 2023, m = c(1, 3, 5, 9)))
is.regular(ms)
as.regular(ms)
# date, resulution 15d
(ds <- tind(y = 2024, m = rep(1:3, each = 2), d = c(1, 16)))
is.regular(ds)
extend_regular(ds, -4)
(ds <- ds[-2L])
is.regular(ds)
as.regular(ds)

# corner cases
tz <- "Europe/Warsaw"
if (tz %in% OlsonNames()) {
    # switch to DST
    print(hours_in_day("2025-03-30", tz = tz))
    # this will work with step from 1am to 3am
    tt <- date_time("2025-03-30", H = c(0, 4:8), tz = tz)
    print(resolution_t(tt))
    as.regular(tt)
}
if (tz %in% OlsonNames()) {
    # this will fail due to missing 2am
    tt <- date_time("2025-03-30", H = c(0, 4, 6, 8), tz = tz)
    print(resolution_t(tt))
    try(as.regular(tt))
}
if (tz %in% OlsonNames()) {
    # this will work again (step by 4h)
    tt <- date_time("2025-03-30", H = c(0, 4, 12), tz = tz)
    print(resolution_t(tt))
    as.regular(tt)
}
```

parse_t *Parse Character Representation of Time Indices Given the Order of Components*

## Description

parse_t parses character vector to create an object of tind class based on provided order(s) of time index components. Index type is inferred from components given.

## Usage

```
parse_t(x, order, locale = NULL, type = NULL, tz = NULL)
```

## Arguments

| | |
|---|---|
| x | a character vector of time indices to be parsed. |
| order | a character string or a character vector describing order(s) of time index components in the input (x), see Details. |
| locale | (optional) a character value determining locale or NULL (the default, interpreted as the current system locale), see calendar-names for information on locale settings. |
| type | (optional) a character value determining time index type. |
| tz | (optional) a character value determining the time zone (the default NULL is interpreted as the system time zone). See tzone documentation for information on time zones. |

## Details

Accepted names of components are:

- y year.
- q quarter.
- m month, number (1–12) or name.
- d day.
- j day of year.
- w week (01–53) as defined in ISO 8601.
- u day of week, number (1–7 with Monday as the first day, ISO 8601) or name.
- H hour.
- I hour, 12-hour clock.
- p AM/PM indicator.
- M minute.
- S second.

`z` UTC offset (`+-HHMM`, `+-HH`, `+-HH:MM`, or letter `Z` for UTC) or time zone abbreviation (like `CET` or `CEST`).

The following combinations of components (in any order) are accepted for different index types (whitespace between specifiers is ignored):

**year (type `"y"`):** `y`.

**quarter (type `"q"`):** `y` and `q`.

**month (type `"m"`):** `y` and `m`.

**week (type `"w"`):** `y` and `w`.

**date (type `"d"`):** `y`, `m`, and `d`; `y`, and `j`; `y`, `w`, and `u`.

**time of day (type `"h"`):** at least hour component with optional minutes and seconds.

**date-time (type `"t"`):** any valid combination for date and at least hour component with optional minutes and seconds.

During parsing all non-digits are skipped in front of `"y"`, `"q"`, `"w"`, `"d"`, `"j"`, `"H"`, `"I"`, `"M"`, `"S"` specifiers and all non-alphanumeric characters are skipped in front of `"m"`, `"u"`, `"p"`. Only whitespace is ignored in front of `"z"` specifier.

`parse_t` was inspired by `ymd`, `mdy`, etc. family of functions from package **lubridate** but independently implemented from scratch (and is a bit faster).

## Value

An object of `tind` class.

## See Also

`strptind` for index parsing requiring strict format specification, calendar-names for information on locale settings.

## Examples

```
## years
# four-digit year
parse_t(as.character(1998:2002), "y")
# two-digit year
parse_t(c("98", "99", "00", "01", "02"), "y")
# mixture of four-digit and two-digit years
parse_t(c("1998", "1999", "00", "01", "02"), "y")

## quarters
parse_t(c("2020Q1", "2020Q2", "2020Q3", "2020Q4"), "yq")

## months
parse_t(c("2020-03", "2020-06", "2020-09", "2020-12"), "ym")
parse_t(c("03/20", "06/20", "09/20", "12/20"), "my")
# missing leading zeros are also handled
parse_t(c("3/20", "6/20", "9/20", "12/20"), "my")
```

```
## weeks
# standard format
parse_t(c("2020-W01", "2020-W05", "2020-W09", "2020-W13"), "yw")
# non-standard format
parse_t(c("2020, week: 01", "2020, week: 05", "2020, week: 09", "2020, week: 13"), "yw")
# missing leading zeros are also handled
parse_t(c("2020, week: 1", "2020, week: 5", "2020, week: 9", "2020, week: 13"), "yw")

## dates
# ISO format
parse_t(c("2025-03-19", "2025-06-18", "2025-09-17", "2025-12-17"), "ymd")
# US format
parse_t(c("03/19/25", "06/18/25", "09/17/25", "12/17/25"), "mdy")
# missing leading zeros are handled
parse_t(c("3/19/25", "6/18/25", "9/17/25", "12/17/25"), "mdy")
# European format
parse_t(c("19.03.2025", "18.06.2025", "17.09.2025", "17.12.2025"), "dmy")
# mixed formats
parse_t(c("03/19/25", "06/18/25", "17.09.2025", "17.12.2025"), c("mdy", "dmy"))
parse_t(c("03/19/25", "06/18/25", "2025-09-17", "2025-12-17"),  c("mdy", "ymd"))

## time of day
parse_t("13:03:34.534", "HMS")
parse_t("13:03:34", "HMS")
parse_t("13:03", "HM")
parse_t("13", "H")
parse_t("1:03:44 pm", "IMSp")
parse_t("1pm", "Ip")

## date-time
parse_t("2025-02-01 13:03:34.534", "ymdHMS")
parse_t("2025-02-01 13:03:34.534", "ymdHMS", tz = "UTC")
parse_t("02/01/25 01:03:34pm", "mdyIMSp")
parse_t("02/01/25 01:03:34pm", "mdyIMSp", tz = "UTC")
```

---

`pretty` *Pretty Breakpoints for Time Indices*

---

## Description

Determine locations of pretty breakpoints for time indices. `pretty` method for objects of `tind` class employs separate algorithms for each type / resolution, see Details.

## Usage

```
## S3 method for class 'tind'
pretty(x, n = 5L, min.n = n%/%2L, ...)
```

## Arguments

| | |
|---|---|
| x | an object of tind class. |
| n | an integer value giving the expected number of intervals. |
| min.n | an integer value giving the minimal number of intervals. |
| ... | (ignored) further arguments passed to or from other methods |

## Details

Resolution of ticks (see resolution_t method) is always the same or lower than the resolution of the argument and lower resolutions have to be multiples of the resolution of the argument. This way, the ticks are never placed, for example, every 5 years for indices with a 2-year resolution.

For years, the tics are placed at powers of 10 times 1, 2, or 5.

For quarters, ticks are placed every quarter, every second quarter (1st and 3rd), or on first the quarters of years selected by separate procedure for years.

For months, ticks are placed every 1, 2, 3, 4, 6 months or on January of years selected by the separate procedure for years.

For weeks, ticks are placed every 1, 2, 4, 13, 26 weeks or on the first weeks of years selected by the separate procedure for years.

For dates, depending on the number of observations, ticks can be placed:

- every day,
- every Monday, Wednesday, and Friday,
- every Monday and Thursday,
- every Monday,
- every 1st and 16th day of a month,
- on 1st days of months selected by the separate procedure for months.

For date-time and time of day, the placement of ticks depends on the resolution of indices. When all indices are at full hours, ticks are placed at full hours only. Similar approach is taken for minutes and seconds. Divisors of 24 are used for hours and divisors of 60 for minutes and seconds. For date-time indices spanning more than a couple of days, ticks are placed on midnights of days selected by the separate procedure for dates.

Due to the design of the algorithm, in some corner cases (esp. for time of day and weeks) the number of intervals might differ significantly from the expected number n.

## Value

An object of tind class.

## See Also

resolution_t method, axis_t for computing time axis parameters for plotting, axis.tind for creating axes with **graphics** package, scale_tind for creating axes with **ggplot2**.

## Examples

```
(td <- tind(y = sample(2010:2018, 4, replace = TRUE),
            m = sample(1:12, 4, replace = TRUE),
            d = sample(1:2, 4, replace = TRUE)))
pretty(td)
pretty(td, 3)
pretty(td, 10)
(th <- tind(H = sample(0:23, 4, replace = TRUE),
            M = sample(0:3 * 15, 4, replace = TRUE)))
pretty(th)
pretty(th, 3)
pretty(th, 10)
(tdt <- date_time(td[1], th))
pretty(tdt)
pretty(tdt, 3)
pretty(tdt, 10)
```

---

# resolution_t *Determine the Resolution of Time Indices*

---

## Description

`resolution_t` method determines resolution of time indices.

For time index types other than integer index (`"i"`) and numeric index (`"n"`), `resolution_t` returns an object of `tdiff` class. The following multiples of units can be returned by `resolution_t` method:

`"y"` **(years):** 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000.

`"q"` **(quarters):** 1, 2.

`"m"` **(months):** 1, 2, 3, 4, 6.

`"w"` **(weeks):** 1, 2, 4, 13, 26.

`"d"` **(days):** 1, 15 (1st and 16th day of a month).

`"h"` **(hours):** 1, 2, 3, 4, 6, 8, 12.

`"min"` **(minutes):** 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30.

`"s"` **(seconds):** 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30 and 1, 2, or 5 times negative powers of 10.

Basic resolution (1) for given type is always returned for vectors with 1 or no non-`NA` value.

An integer value is returned for type `"i"` (integer index) and a numeric value (possibly `NA_real_`) for type `"n"` (numeric index).

## Usage

```
resolution_t(x)

## S3 method for class 'tind'
resolution_t(x)
```

### Arguments

| | |
|---|---|
| x | an object of tind class or of other time index class supported by **tind** package. |

### Value

For all types except for integer index ("i") and numeric index ("n") resolution_t returns an object of tdiff class. An integer value for type "i" (integer index) and a numeric value (possibly NA_real_) for type "n" (numeric index) are returned.

### See Also

rounding for rounding time indices to specified resolution, is.regular method for checking if time indices form a regular sequence, tspan method for determining time span of indices.

### Examples

```
(ds <- tind(y = 2024, m = rep(1:3, each = 2), d = c(1, 16)))
resolution_t(ds)
(ms <- tind(y = 2023, m = 1 + 2 * (0:5)))
resolution_t(ms)
(th <- tind(H = 13, M = (0:3) * 15))
resolution_t(th)
(dt <- tind(y = 2025, m = 2, d = 1, H = 13, M = 27, S = (0:5) * 10))
resolution_t(dt)
```

---

## rounding *Rounding Time Indices*

---

### Description

Time indices can be rounded to different time units (depending on the type of time index at hand, see Details).

trunc_t rounds the indices down to a given unit with change of index type where applicable.

floor_t rounds the indices down to a (multiple of a) unit.

ceiling_t rounds the indices up to a (multiple of a) unit.

round_t rounds the indices to the closest multiple of a unit, i.e. the result of floor_t or ceiling_t, whichever is closer.

### Usage

```
trunc_t(x, unit)

floor_t(x, unit)

ceiling_t(x, unit, ceiling = c("default", "following", "last"))

round_t(x, unit, ceiling = c("default", "following", "last"))
```

## Arguments

| | |
|---|---|
| x | an object of tind class (or an R object coercible to it). |
| unit | a character string determining unit (expected by trunc_t), a number, an object of tdiff class, or a character string with a number and unit name. |
| ceiling | (optional) a character string determining the behaviour of ceiling_t (and round_t), see Details. |

## Details

### Units and Unit Multiples

For trunc_t, unit has to be a character string determining resolution / type to which indices should be truncated. For the remaining functions, unit argument can be a number (the default unit for index type will be used), a character string with unit name, an object of tdiff class, or a character string with a number and unit name.

The following unit names are accepted:

"y", "year", "years" years,

"q", "quarter", "quarters" quarters,

"m", "mon", "month", "months" months,

"w", "week", "weeks" weeks,

"d", "day", "days" days,

"h", "hour", "hours" hours,

"min", "minute", "minutes" minutes,

"s", "sec", "second", "seconds" seconds.

The default unit for date-time and time of day indices is a second.

The list of admissible multiples of units can be found in the documentation of resolution_t method.

For time indices of types "i" and "n" (integer and numeric indices) unit can be any finite, positive integer / numeric value.

In case of a tie (x - floor_t(x, *) equal to ceiling_t(x, *) - x), round_t returns the result of ceiling_t.

### Controlling behaviour of ceiling_t (and round_t)

For non-instant time indices, i.e indices that actually represent periods of time (days weeks, months, etc.) ceiling_t rounds to the first index in a period by default. For instance, when rounding dates to months, the first day of a month will be unchanged and other days will be rounded to the first day in the *following* month. This behaviour can be altered by setting ceiling argument. If set to "following", the first index (day in our example) in the following period will be returned. If set to "last", the last index (day in our example) in the period will be returned. See Examples.

## Value

An object of tind class of the same type and length as x except for trunc_t, for which the type of the result is determined based on unit argument.

## Note

Methods `floor`, `ceiling`, `round`, and `trunc` are not implemented for `tind` class due to generics' argument list limitations.

## See Also

`resolution_t` method.

## Examples

```
(d <- as.tind("2024-08-27"))
floor_t(d, "w")
trunc_t(d, "w")
ceiling_t(d, "w")
round_t(d, "w")
floor_t(d, "m")
trunc_t(d, "m")
ceiling_t(d, "m")
round_t(d, "m")
floor_t(d, "3m")
ceiling_t(d, "3m")
round_t(d, "3m")

(dt <- as.tind("2024-08-27 13:51:52"))
floor_t(dt, 10)
floor_t(dt, "10s")
ceiling_t(dt, "10s")
round_t(dt, "10s")
floor_t(dt, "min")
trunc_t(dt, "min")
ceiling_t(dt, "min")
round_t(dt, "min")
floor_t(dt, "h")
trunc_t(dt, "h")
ceiling_t(dt, "h")
round_t(dt, "h")
floor_t(dt, "2h")
ceiling_t(dt, "2h")
round_t(dt, "2h")
floor_t(dt, "d")
trunc_t(dt, "d")
ceiling_t(dt, "d")
round_t(dt, "d")

# corner cases - DST change (02:00 missing)
(dt <- date_time("2025-03-30", H = c(0:1, 3:6)))
floor_t(dt, "2h")
ceiling_t(dt, "2h")

# adjusting behaviour of ceiling_t for non-instant time indices
# a short sequence of dates covering two months
(ds <- as.tind("2023-01-01") + -2:2)
```

```
# default behaviour
ceiling_t(ds, "2m")
ceiling_t(ds, "2m", ceiling = "default")
# round up to the first day in the following 2-month period
ceiling_t(ds, "2m", ceiling = "following")
# round up to the last day in the current 2-month period
ceiling_t(ds, "2m", ceiling = "last")
# a corner case, note that we will get the next day as a result
ceiling_t(today(), "1d", ceiling = "following")
```

---

scale_tind *Time Scales for Plotting with* **ggplot2**

---

## Description

These functions provide **ggplot2** scales for `tind`. Scales will be added automatically by **ggplot2**, but the default behaviour can be overridden by adding `scale_*_tind` to the plot.

## Usage

```
scale_x_tind(
  name = ggplot2::waiver(),
  breaks = ggplot2::waiver(),
  minor_breaks = ggplot2::waiver(),
  n.breaks = 7L,
  labels = ggplot2::waiver(),
  limits = NULL,
  expand = ggplot2::waiver(),
  guide = ggplot2::waiver(),
  position = "bottom",
  format = NULL,
  locale = NULL
)

scale_y_tind(
  name = ggplot2::waiver(),
  breaks = ggplot2::waiver(),
  minor_breaks = ggplot2::waiver(),
  n.breaks = 7L,
  labels = ggplot2::waiver(),
  limits = NULL,
  expand = ggplot2::waiver(),
  guide = ggplot2::waiver(),
  position = "left",
  format = NULL,
  locale = NULL
)
```

## Arguments

| | |
|---|---|
| name | a character string with axis name, waiver() for default name, or NULL for none. |
| breaks | waiver() for default tick-marks, NULL for none, time indices at which tick-marks should be placed (tind), or a tdiff / character string determining distance between breaks. |
| minor_breaks | waiver() for default minor tick-marks, NULL for none, or time indices at which minor tick-marks should be placed. |
| n.breaks | an integer value, desired number of breaks. |
| labels | waiver() for default labels, NULL for none, or a character vector of labels. |
| limits | NULL for automatic limits, tinterval of length 1 or tind of length 2. |
| expand, guide | see scale_continuous. |
| position | a character string determining axis position, "bottom" or "top" for scale_x_tind, "left" or "right" for scale_y_tind. |
| format | (optional) a character string determining label format (see format) or a formatting function. |
| locale | (optional) a character string determining locale to be used for formatting labels, see calendar-names for information on locale settings. |

## Details

The algorithm determining positioning of breaks and minor breaks always takes the resolution of time indices (see resolution_t) into account. For example, for monthly data with breaks placed every three months (January, April, July, October) minor breaks will never be placed in the middle (mid of February, May, August, November) but rather every month. The algorithm overrides the default approach of **ggplot2** — axis limits are determined based on breaks and breaks on time indices and their resolution, whereas **ggplot2** starts with limits based on data and determines breaks based on limits only (ignoring the resolution of time indices).

Argument list is a bit different from that of to scale_*_date and scale_*_datetime. Firstly, n.breaks argument is supported, allowing users to set the expected number of breaks on time axis. Secondly, labels cannot be a function. Formatting functions can be provided via format argument, which also supports character string with format specification (see format) rendering date_labels argument redundant. locale argument controls the language used for month and weekday names, see calendar-names. breaks cannot be a function, but can be a tdiff / character string determining distance between breaks rendering date_breaks argument redundant. limits argument is expected to determine time interval and not limits in Cartesian coordinates. Open-ended intervals are supported.

breaks, minor_breaks, and limits cannot be functions as **ggplot2** assumes that breaks and limits can be properly set based on (automatic) limits only without taking into account the resolution of time indices, which is not true.

Secondary axes are not supported.

## Value

A continuous scale as returned by continuous_scale.

## Note

Due to the fact that `limits` method is currently (as of version 4.0.0) not exported, users cannot use `xlim` and `ylim` with `tind` scales. Limits on time-indexed axes can be set using `limits` argument to `scale_*_tind`.

## See Also

`pretty` for computing pretty breakpoints, `axis_t` for computing time axis parameters, `axis.tind` for creating axes with **graphics** package.

## Examples

```
# artificial data
d <- seq(floor_t(today(), "y"), ceiling_t(today(), "y", ceiling = "last"))
df <- data.frame(d = d, D = as.Date(d), y = cumsum(rnorm(length(d))),
                 q = paste0("Q", quarter(d)))
# load ggplot2
have_ggplot2 <- require("ggplot2", quietly = FALSE)
# default scale
if (have_ggplot2) {
    ggplot(df) + geom_line(aes(x = d, y = y)) + theme_bw()
}
# compare with the default scale for Date
if (have_ggplot2) {
    ggplot(df) + geom_line(aes(x = D, y = y)) + theme_bw()
}
# change format
if (have_ggplot2) {
    ggplot(df) + geom_line(aes(x = d, y = y)) + theme_bw() +
        scale_x_tind(format = "%b '%y")
}
# set breaks every 4 months
if (have_ggplot2) {
    ggplot(df) + geom_line(aes(x = d, y = y)) + theme_bw() +
        scale_x_tind(breaks = "4m")
}
# set limits
if (have_ggplot2) {
    ggplot(df) + geom_line(aes(x = d, y = y), na.rm = TRUE) + theme_bw() +
        scale_x_tind(limits = c(today() %-m% 4, today()))
}
# facets with custom formatting and reduced number of breaks
if (have_ggplot2) {
    ggplot(df) + geom_line(aes(x = d, y = y)) + theme_bw() +
        scale_x_tind(n.breaks = 4, format = "%b '%y") +
        facet_wrap(~q, scales = "free_x")
}
```

seq *Create a Sequence of Time Indices*

## Description

seq method for objects of tind class allows to easily construct sequences of time indices of all supported types.

## Usage

```
## S3 method for class 'tind'
seq(from, to, by = 1, length.out = NULL, along.with = NULL, ...)
```

## Arguments

| | |
|---|---|
| from | an object of tind class or an R object coercible to it. |
| to | an object of tind class or an R object coercible to it. |
| by | a numeric value, a tdiff, or a character string determining increment. |
| length.out | an integer value, the desired length. |
| along.with | any R object, length of this argument will be taken as the desired length. |
| ... | (ignored) further arguments passed to or from other methods. |

## Details

seq method requires that exactly two of the three arguments from, to, and length.out are provided. If along.with is not NULL, its length is used as value of length.out.

by can be a number, an object of tdiff class (of length 1), or an object coercible to tdiff like "3w" denoting step by three weeks. by cannot be NA and cannot be 0 when both from and to are provided. Given both from and to, sign of by has to agree with the order of from and to. When by is a number, the underlying unit of time is assumed. For time of day and date-time indices this is always a second.

from and to can be of different types provided that conversion is possible. The result is of higher resolution. This allows to easily construct series like from begging of the month to today, from today till the end of next year, etc. See Examples.

seq.tind slightly differs from seq.Date in terms of interface and requirements with respect to arguments. Firstly, from argument can be missing (provided that to and length.out are given). Secondly, by has the default value of 1.

Both seq method for tind and seq.tind function are exported allowing for conversion to tind as in seq.tind("2025-01", "2025-12").

## Value

An object of tind class.

## Examples

```
# sequences of dates by 1 and 2 months from now
(td <- today())
seq(td, by = "1m", length.out = 12)
seq(td, by = "2m", length.out = 6)
# sequences of dates by 1 and 2 months to now
seq(to = td, by = "1m", length.out = 12)
seq(to = td, by = "2m", length.out = 6)
# sequence of dates from the beginning of the month till today
seq(floor_t(td, "m"), td)
# same
seq(as.month(td), td)
# sequence of dates from today till the end of the next month
seq(td, as.month(td) + 1)
# sequence of date-time from now to midnight by 1 hour
(nw <- now())
seq(nw, ceiling_t(nw, "1d"), by = "1h")
# same
seq(nw, as.date(nw), by = "1h")
# sequence (date-time) from full hour to now by 2 minutes
seq(floor_t(nw, "1h"), nw, by = "2min")
# sequence (time of day) from full hour to now by 2 minutes
seq(floor_t(as.time(nw), "1h"), as.time(nw), by = "2min")
# sequence (date-time) from now down to full hour by 2 minutes
seq(nw, floor_t(nw, "1h"), by = "-2min")
# sequence (time of day) from now down to full hour by 2 minutes
seq(as.time(nw), floor_t(as.time(nw), "1h"), by = "-2min")
# sequence (date-time) of length 10 from now down by 10 seconds
seq(nw, by = -10, length.out = 10)
# sequence (time of day) of length 10 from now down by 10 seconds
seq(as.time(nw), by = -10, length.out = 10)
# explicit call to seq.tind with conversion
seq.tind("2025-01", "2025-12")
## corner cases
# from 2025-12-30 23:00 till end of 2025, note that 2025-12-31 24:00
# (that is 2025-01-01 00:00) is excluded from the result as it is in the next year
seq(as.tind("2025-12-30 23:00", tz = "UTC"), "2025", by = "5h")
# from end of 2025 down to 2025-12-30 23:00, note that 2025-12-31 24:00
# (that is 2025-01-01 00:00) is excluded from the result as it is in the next year
seq(as.tind("2025"), as.tind("2025-12-30 23:00", tz = "UTC"), by = "-5h")
```

---

# set-ops *Set Operations on Time Intervals and Time Indices*

---

## Description

Time intervals can be thought of as subsets of time line (set of integers or real line, depending on index type). The following functions perform operations on these sets.

`unique` method for objects of `tinterval` class returns unique representation as ordered sum of disjoint, non-adjacent intervals.

`!` (negation) operator for objects of `tinterval` class returns set-theoretical complement of the argument.

`intersect_t`, `union_t`, `setdiff_t` return intersection, union and (asymmetric) set difference. These three functions accept both time intervals and time indices.

Behaviour of `!`, `intersect_t`, `union_t`, and `setdiff_t` is consistent with behaviour of `%in_t%` operator. Consistency is also assured under type conversions.

For time indices, `intersect_t`, `union_t`, `setdiff_t` behave just like `intersect`, `union`, `setdiff` from **base** (see sets) but preserve class attribute.

## Usage

```
## S3 method for class 'tinterval'
unique(x, ...)

## S3 method for class 'tinterval'
!x

intersect_t(x, y)

union_t(x, y)

setdiff_t(x, y)
```

## Arguments

| | |
|---|---|
| `x` | an object of `tinterval` class for `unique` method and `!` operator, `tinterval` or `tind` for `intersect_t`, `union_t`, or `setdiff_t`. |
| `...` | (ignored) further arguments passed to or from other methods. |
| `y` | an object of `tinterval` or `tind` class. |

## Details

For discrete time indices (represented as integers, i.e. years, quarters, months, weeks, dates, arbitrary integer indices) time intervals represent the following sets (ignoring empty, i.e. with $a_i > b_i$):

$$\bigcup_{i=1}^{n} A_i = \bigcup_{i=1}^{n} \{x : a_i \leq x \leq b_i\} = \bigcup_{i=1}^{n} \{a_i, a_i + 1, \ldots, b_i - 1, b_i\}.$$

`unique` returns the unique (canonical) representation of the set above:

$$\bigcup_{i=1}^{n'} A'_i = \bigcup_{i=1}^{n'} \{a'_i, a'_i + 1, \ldots, b'_i - 1, b'_i\}$$

with $a'_i \leq b'_i < a'_{i+1} - 1$, i.e. as a sum of ordered, non-empty, non-adjacent intervals.

For continuous time indices (representing point in time, i.e. date-time, time of day, arbitrary numeric indices) time intervals represent the following sets (ignoring empty, i.e. with $a_i \geq b_i$):

$$\bigcup_{i=1}^{n} A_i = \bigcup_{i=1}^{n} [a_i, b_i).$$

`unique` returns unique representation of the set above:

$$\bigcup_{i=1}^{n'} A'_i = \bigcup_{i=1}^{n'} [a'_i, b'_i)$$

with $a'_i < b'_i < a'_{i+1}$, i.e. as a sum of ordered, non-empty, non-adjacent intervals.

Complement of a single interval for integer indices

$$\{x : a \leq x \leq b\} = \{a, a+1, \ldots, b-1, b\}$$

is:

$$\{x : x < a\} \cup \{x : x > b\} = \{x : x \leq a-1\} \cup \{x : x \geq b+1\} = \{\ldots, a-2, a-1\} \cup \{b+1, b+2, \ldots\}.$$

Complement of a single interval for continuous indices

$$[a, b)$$

is:

$$(-\infty, a) \cup [b, \infty).$$

Complement of a sum of intervals is the intersection of complements.

Set operations always return results in the canonical representation.

## Value

An object of `tinterval` or `tind` class representing result of the set operation.

## See Also

`tinterval` for an overview of time interval class, `match_t` for matching time indices

## Examples

```
(x <- tinterval("2025-03-15", "2025-03-20") + c(0, 4, 14))
# unique representation (non-overlapping intervals)
unique(x)
# complement
!x
# binary set operations
(y <- tinterval("2025-03-01", "2025-03-31"))
intersect_t(x, y)
union_t(x, y)
setdiff_t(x, y)
setdiff_t(y, x)
```

```
# different types of indices
(y <- tinterval("2025-W10", "2025-W11"))
intersect_t(x, y)
union_t(x, y)
setdiff_t(x, y)
setdiff_t(y, x)
# check
(y <- as.tinterval(y, type = "d"))
intersect_t(x, y)
union_t(x, y)
setdiff_t(x, y)
setdiff_t(y, x)
```

---

tdiff *Time Differences*

---

## Description

Objects of tdiff class represent time differences and are similar to difftime objects. tdiff objects are created by subtracting two time indices (of types other than "i" and "n") or via calls to as.tdiff method. An alternative way of constructing tdiff objects is to call years, qrtrs, mnths, weeks, days, hours, mins, and secs convenience functions.

The following units (argument unit) are supported:

"y" ("year", "years") differences in years,

"q" ("quarter", "quarters") differences in quarters,

"m" ("month", "months") differences in months,

"w" ("week", "weeks") differences in weeks,

"d" ("day", "days") differences in days,

"h" ("hour", "hours") differences in hours,

"min" ("mins", "minute", "minutes") differences in minutes,

"s" ("secs", "second", "seconds") differences in seconds.

Standard methods for vectors and conversion from / to numeric and character vectors are implemented for this class.

## Usage

```
is.tdiff(x)

as.tdiff(x, ...)

## S3 method for class 'numeric'
as.tdiff(x, unit, ...)
```

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

## S3 method for class 'difftime'
as.tdiff(x, ...)

years(x)

qrtrs(x)

mnths(x)

weeks(x)

days(x)

hours(x)

mins(x)

secs(x)

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

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

## S3 method for class 'tdiff'
as.data.frame(x, ...)

## S3 method for class 'tdiff'
x[i]

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

## S3 method for class 'tdiff'
x[[i]]

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

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

## S3 method for class 'tdiff'
c(...)
```

```
## S3 method for class 'tdiff'
Math(x, ...)

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

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

## S3 method for class 'tdiff'
unique(x, ...)

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

## S3 method for class 'tdiff'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| x | a numeric vector or any R object coercible to tdiff, for as.tdiff and years, qrtrs, etc.; an object of tdiff class for methods. |
| ... | further arguments passed to or from other methods. |
| unit | a character string, name of the time unit, see Details. |
| i | an integer vector of indices or a logical vector indicating selection. |
| value | replacement value. |
| na.rm | a logical value indicating whether missing values should be removed. |
| object | an object of tdiff class. |

## Details

tdiff objects are implemented as vectors of integers (for differences in years, quarters, months, weeks, and days) or vectors of doubles (for time differences). Time differences are internally represented in seconds but when printing the actual time unit (hour, minute, second) is automatically inferred and used.

Valid ranges for tdiff values depend on unit. These are defined by differences of the maximal and minimal valid time indices of the type corresponding to the time unit.

## Value

as.tdiff as well as convenience functions years, qrtrs, etc., return objects of tdiff class.

is.tdiff returns a logical value.

In general, methods for tdiff return objects of tdiff class.

as.character and format return character vectors.

`as.data.frame` returns a data frame with a single column and the number of rows equal to the length of the argument.

`print` returns its argument invisibly and is used for its side effect.

`summary` returns an object of class `c("summaryDefault", "table")`.

## Note

Since `as.difftime` is not implemented in **base** as an S3 generic, conversion from `tdiff` to `difftime` is not provided.

## See Also

Ops for operations on time indices and time differences.

## Examples

```
# how many days have passed since Jan 1, 2000?
today() - as.date("2000-01-01")
# how many months have passed since Sep 2008?
as.month(today()) - as.month("2008-09")
# create time differences in quarters
as.tdiff(-2:2, "q")
# same
(x <- qrtrs(-2:2))
# add to today
today() + x
```

---

# time-index-components *Time Index Components (Years, Months, Days, ...)*

---

## Description

The following functions can be used to retrieve components of time indices.

`year`, `quarter`, `month`, `week`, `day` return year (0–9999), quarter (1–4), month (1–12), week (1–53, ISO 8601), and day (1–31) indices as integers. When `month` is invoked with `labels` argument set to `TRUE`, an ordered factor is returned.

`day_of_week` returns index (1–7) of weekday with Monday as the first day (as in ISO 8601). When invoked with `labels` argument set to `TRUE`, an ordered factor is returned.

`day_of_year` returns index (1–366) of the day of year as integer.

`hour`, `minute`, `second` return hour (0–23), minute (0–59), and second (0–59,999999) indices as integers and reals (for seconds). `am` and `pm` functions determine whether time falls in the first or second half of the day.

Methods `weekdays`, `months`, and `quarters` from package **base** are implemented but users are encouraged to use functions from **tind** package.

## Usage

```
year(x)

quarter(x)

## S3 method for class 'tind'
quarters(x, abbreviate)

month(x, labels = FALSE, abbreviate = TRUE, locale = NULL)

## S3 method for class 'tind'
months(x, abbreviate = FALSE)

week(x)

day(x)

day_of_year(x)

day_of_week(x, labels = FALSE, abbreviate = TRUE, locale = NULL)

## S3 method for class 'tind'
weekdays(x, abbreviate = FALSE)

hour(x)

am(x)

pm(x)

minute(x)

second(x)
```

## Arguments

| | |
|---|---|
| x | an object of tind class or an R object coercible to it. |
| abbreviate | a logical value, if TRUE, abbreviated names are returned; if FALSE, full names are returned. TRUE by default. |
| labels | a logical value, if TRUE month and weekday names are returned (as ordered factors) instead of integer indices (FALSE by default). |
| locale | (optional) a character value determining locale or NULL (the default, interpreted as the current system locale), see calendar-names for information on locale settings. |

## Details

year for week arguments need not necessarily return the same value as for days within the week in

question when the week is the first or the last in a year.

**tind** package does not provide replacement methods for time index components. In order to change, say, month one can use `tind` constructor or `%+m%` operator (and similar operators), see Examples.

## Value

All functions return integer vectors, except for `second`, which returns numeric vectors. Additionally, `month` and `day_of_week` return ordered factors if invoked with argument `labels` set to `TRUE`.

## See Also

`tind` class, Ops for index increments / decrements and index differences, and calendar-names for names of months and days of weeks in the current locale. Further examples of use of these functions can be found in `calendar` documentation.

## Examples

```
# current date and time
(nw <- now())
# show current year, quarter, month, ...
year(nw)
quarter(nw)
month(nw)
month(nw, labels = TRUE)
month(nw, labels = TRUE, abbreviate = FALSE)
week(nw)
day(nw)
day_of_week(nw)
day_of_week(nw, labels = TRUE)
day_of_week(nw, labels = TRUE, abbreviate = FALSE)
day_of_year(nw)
hour(nw)
minute(nw)
second(nw)

# alternatives to replacement, change month to December
(x <- as.date("2023-09-11"))
(x <- tind(y = year(x), m = 12, d = day(x)))
(x <- as.date("2023-09-11"))
(x <- x %+m% (12 - month(x)))
```

---

`time-index-properties` *Time Index Properties (Leap Years, Period Lengths, DST)*

---

## Description

The following functions can be used to determine whether years are leap years, compute the number of subperiods within a period, and determine whether Daylight Saving Time is on for particular date-time index. All function are vectorised.

`is.leap_year` returns `TRUE` for leap years and `FALSE` for non-leap ones.

`days_in_year` and `weeks_in_year` return the number of days (365–366) and the number of weeks (52–53) in a year.

`days_in_quarter` and `days_in_month` return the number of days in a quarter (90–92) or a month (28–31), respectively.

`hours_in_day` returns the number of hours in a day (24 most of the time, a different number during DST/UTC offset changes).

`is.dst` returns `TRUE` when Daylight Saving Time is on and `FALSE` otherwise.

## Usage

```
is.leap_year(x)

days_in_year(x)

weeks_in_year(x)

days_in_quarter(x)

days_in_month(x)

hours_in_day(x, tz = NULL)

is.dst(x)
```

## Arguments

| | |
|---|---|
| `x` | an object of `tind` class or an R object coercible to it. |
| `tz` | (optional) a character value determining the time zone (the default `NULL` is interpreted as the system time zone). See `tzone` documentation for information on time zones. |

## Value

`is.leap_year` and `is.dst` return logical vectors. The remaining functions return integer vectors, except for `hours_in_day`, which returns numeric vectors.

## See Also

time-index-components, calendrical-computations, Ops, `tzone`, `bizdays_in_month`.

tind *A Common Representation of Time Indices of Different Types*

## Description

tind is an S3 class representing time indices of different types (years, quarters, months, ISO 8601 weeks, dates, date-time, and arbitrary integer/numeric indices). Time indices are represented by vectors of integers or doubles with type attribute and time zone attribute (date-time only). Objects of tind behave like plain vectors and can be easily used in data frames.

A tind object would usually be created using as.tind method or using parse_t and strptind functions. tind constructor allows to create time indices from components (like year, month, day) and to create vectors of a given length filled with NA values.

is.tind function checks whether an object is of tind class.

## Usage

```
tind(..., length = 0L, type = NULL, tz = NULL)

is.tind(x)
```

## Arguments

| | |
|---|---|
| ... | components of time index to be constructed (in arbitrary order), the following are accepted:<br>**y** year.<br>**q** quarter.<br>**m** month.<br>**w** week (ISO 8601).<br>**d** day.<br>**j** day of year.<br>**u** day of week (ISO 8601).<br>**H** hour.<br>**M** minute.<br>**S** second. |
| length | an integer value specifying the desired length. |
| type | a character value determining time index type (y - years, q - quarters, m - months, w - weeks, d - dates, t - date-time, h - time of day, n - numeric value, i - integer value). |
| tz | (optional) a character value determining the time zone (the default NULL is interpreted as the system time zone). See tzone documentation for information on time zones. |
| x | any R object. |

## Details

`tind` class supports the following types of time indices:

**years** internal code `"y"`.

**quarters** internal code `"q"`.

**months** internal code `"m"`.

**weeks** internal code `"w"`.

**dates** internal code `"d"`.

**date-time** internal code `"t"`.

**time of day** internal code `"h"`.

**arbitrary integer index** `"i"`.

**arbitrary numeric index** `"n"`.

Valid ranges for time indices are:

**years** (`"y"`) `0000`–`9999`.

**quarters** (`"q"`) `0000q1`–`9999q4`.

**months** (`"m"`) `0000-01`–`9999-12`.

**weeks** (`"w"`) `0000-W01`–`9999-W52`.

**dates** (`"d"`) `0000-01-01`–`9999-12-31`.

**date-time** (`"t"`) from `0000-01-01 15:00:00Z` to `9999-12-31 09:00:00Z` (between `-62167165200` and `253402246800` seconds since the Unix epoch).

**time of day** (`"h"`) from `00:00` to `24:00` (between `0` and `86400` seconds since midnight).

## Value

An object of `tind` class for `tind` or a logical value for `is.tind`.

## See Also

`as.tind` for conversion to `tind`, `parse_t` and `strptind` functions for parsing character strings, `date_time` for construction of date-time indices from date and time components, tind-methods for basic methods.

## Examples

```
# years
tind(y = 2010:2020)
tind(type = "y")
tind(length = 11, type = "y")

# quarters
tind(y = rep(2020:2023, each = 4), q = 1:4)
tind(q = 1:4, y = rep(2020:2023, each = 4))
tind(type = "q")
tind(length = 4, type = "q")
```

```
# months
tind(y = 2023, m = 1:12)
tind(m = 1:12, y = 2023)
tind(type = "m")
tind(length = 12, type = "m")

# weeks
tind(y = 2024, w = 1 + 2 * (0:25))
tind(type = "w")
tind(length = 13, type = "w")

# dates
tind(m = 3, d = 15, y = 2024)
tind(y = 2024, m = rep(1:3, each = 2), d = c(1, 15))
tind(type = "d")
tind(length = 6, type = "d")

# time of day
tind(H = 16, M = (0:3) * 15)
tind(type = "h")
tind(length = 4, type = "h")

# date-time
# system time zone
tind(y = 2024, m = 8, d = 2, H = 16, M = (0:3) * 15)
tind(y = 2024, m = 8, d = 2, H = 16, M = (0:3) * 15)
tind(y = 2024, m = 8, d = 2, H = 16, M = 0, S = 10 * (0:5))
# time zone explicitly provided
tind(y = 2024, m = 8, d = 2, H = 16, M = (0:3) * 15, tz = "UTC")
tind(type = "t")
tind(type = "t", tz = "UTC")
tind(length = 4, type = "t")
tind(length = 4, type = "t", tz = "UTC")

# integer and numeric indices
# (cannot be constructed from components like above)
tind(length = 10, type = "i")
tind(length = 10, type = "n")
```

---

`tind-coercion` *Conversion of Objects of* `tind` *Class*

---

## Description

Objects of `tind` class can be easily converted to built-in R classes inluding `numeric`, `integer`, `character`, `Date`, `POSIXct`, `POSIXlt`, and `data.frame`.

## Usage

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

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

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

## S3 method for class 'tind'
as.Date(x, ...)

## S3 method for class 'tind'
as.POSIXct(x, tz = NULL, ...)

## S3 method for class 'tind'
as.POSIXlt(x, tz = NULL, ...)

## S3 method for class 'tind'
as.data.frame(x, ...)
```

## Arguments

| | |
|---|---|
| x | an object of tind class. |
| ... | further arguments passed to or from other methods. |
| tz | (optional) a character value determining the time zone (the default NULL is interpreted as the system time zone). See tzone documentation for information on time zones. |

## Details

as.double and as.numeric return internal representation for particular time index type (seconds, days, weeks etc. since ...).

For years, quarters, months, weeks, and dates, as.integer returns representation in the form YYYY, YYYYQ, YYYYMM, YYYYWW, and YYYYMMDD, respectively. For other index types, as.integer returns internal representation of time indices converted to integer.

as.character returns character vector with standard (ISO 8601) representation of time indices. For customisable output formats, see format.

as.Date, as.POSIXct, and as.POSIXlt return objects of classes Date, POSIXct, and POSIXlt, respectively.

as.data.frame returns a 1-column data frame with time indices and allows to work with time indices in data frames.

### Value

as.xxx returns an object of xxx class of the same length as the argument. as.data.frame returns a data frame with a single column and the number of rows equal to the length of the argument.

### See Also

format for customisable character output formats, as.tind for conversion to tind. For conversions between tind class and other classes (from packages other than **base**), see tind-other.

---

## tind-methods *Basic Methods for* tind *Class*

---

### Description

tind class supports all standard methods for vectors and and vector-based classes like Date or POSIXct.

### Usage

```
## S3 method for class 'tind'
x[i]

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

## S3 method for class 'tind'
x[[i]]

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

## S3 replacement method for class 'tind'
length(x) <- value

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

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

## S3 method for class 'tind'
unique(x, ...)

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

## S3 method for class 'tind'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| x | an object of tind class. |
| i | an integer vector of indices or a logical vector indicating selection. |
| value | replacement value. |
| ... | objects of tind class (for c, min, max, and range) or additional arguments passed to or from methods. |
| object | an object of tind class. |

## Details

tind class supports standard indexing via [] and [[]] operators, as well as replacement, In replacement, it is expected that the right hand side is of the same type as the indexed object.

length, length<-, and rep methods work in a standard way.

rev, head, tail, as they are implemented using [] operator, are also available for objects of tind class.

Concatenation method (c) works in a standard way. It is expected that all arguments are of the same type. Arguments that are not of tind class are converted.

min, max, and range work in a standard way. If the results are not proper time indices (for example maximum over a vector of length 0), NAs are returned.

unique, duplicated, order, sort, etc. work in a standard way.

print prints time indices on the console and invisibly returns its argument.

summary method returns summary information about time indices.

## Value

In general, methods return objects of tind class.

print returns its argument invisibly and is used for its side effect.

summary returns an object of class c("summaryDefault", "table").

## See Also

format for formatting time indices, Ops for operations on time indices.

## Examples

```
# test sample
(dd <- as.tind(20210131) + sample((0:9), 15, replace = TRUE))
# indexing
dd[1]
dd[[1]]
dd[[1]] <- dd[[1]] + 1
dd
dd[2:3] <- dd[2:3] + 1
dd
# this will generate an error
try(
```

```
dd[10] <- now()
)
# length, length<-
length(dd)
length(dd) <- 7
dd
# rep, head, tail, rev
rep(dd, 2)
head(dd, 3)
tail(dd, -5)
rev(dd)
# min, max, range
min(dd)
max(dd)
range(dd)
# unique, duplicated
unique(dd)
duplicated(dd)
# order, sort
order(dd)
sort(dd)
# concatenation
c(dd, rev(dd))
# attempt at concatenating different types will result in an error
try(
c(today(), now())
)
```

---

`tind-other` *Conversion between* `tind` *and Other Time Index Representations*

---

## Description

Besides `Date`, `POSIXct`, and `POSIXlt` classes from package **base**, **tind** currently supports conversion between `tind` and the following classes: `yearmon`, `yearqtr` (both from package **zoo**), `timeDate` (from package **timeDate**), `chron`, `dates`, `times` (from package **chron**), `IDate`, `ITime` (from package **data.table**), and `hms` (from package **hms**).

## Usage

```
## S3 method for class 'yearmon'
as.tind(x, ...)

as.yearmon(x, ...)

## S3 method for class 'yearqtr'
as.tind(x, ...)
```

```
as.yearqtr(x, ...)

## S3 method for class 'timeDate'
as.tind(x, digits = 0L, ...)

as.timeDate(x, ...)

## S3 method for class 'chron'
as.tind(x, digits = 0L, ...)

as.chron(x, ...)

## S3 method for class 'dates'
as.tind(x, ...)

as.dates(x, ...)

## S3 method for class 'times'
as.tind(x, digits = 0L, ...)

as.times(x)

## S3 method for class 'IDate'
as.tind(x, ...)

as.IDate(x, ...)

## S3 method for class 'ITime'
as.tind(x, ...)

as.ITime(x)

## S3 method for class 'hms'
as.tind(x, ...)

as_hms(x)
```

## Arguments

| | |
|---|---|
| x | an R object to be converted. |
| ... | (ignored) further arguments passed to or from other methods. |
| digits | an integer value (0–6) determining the number of decimal places for seconds to be preserved during conversion (0 by default). |

## Details

Date-time indices resulting from conversion of chron objects always have time zone set to UTC. Use tzone<- or as.tzone methods when necessary.

### Value

as.xxx returns an object of xxx class of the same length as the argument.

### See Also

as.tind and tind-coercion for conversions to and from tind, date2num and num2date for conversion between tind and integer representations of dates (days since ...) found in different software packages.

---

## tinterval *Time Intervals*

---

### Description

Objects of auxiliary tinterval class represent time intervals as pairs of time indices (start and end). Time intervals can be constructed via a call to tinterval function or using convenience %--% operator. Open-ended intervals are supported. The main applications of this class are set operations (see set-ops) and checking if a particular time index belongs to (one of) given interval(s) (see match_t).

### Usage

```
tinterval(start = NULL, end = NULL, ...)

start %--% end

is.tinterval(x)

as.tinterval(x, ...)

## S3 method for class 'character'
as.tinterval(x, sep, ...)

## S3 method for class 'tinterval'
as.tinterval(x, type = NULL, tz = NULL, ...)

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

## S3 method for class 'data.frame'
as.tinterval(x, ...)

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

## S3 method for class 'tinterval'
format(x, sep = " -- ", open = "...", aux = TRUE, empty = "-", ...)
```

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

## S3 method for class 'tinterval'
as.data.frame(x, ...)

## S3 method for class 'tinterval'
x[i]

## S3 method for class 'tinterval'
x[[i]]

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

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

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

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

## S3 method for class 'tinterval'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| start | an object of tind class or an R object coercible to it, beginning of the interval(s). |
| end | an object of tind class or an R object coercible to it, end of the interval(s). |
| ... | objects of tinterval class to be concatenated by c or additional arguments passed to or from methods. |
| x | an object of tinterval class or an R object passed to as.tinterval. |
| sep | a character string used as separator between start and end of an interval (" -- " by default). |
| type | a character determining time index type or NULL. |
| tz | (optional) a character value determining the time zone (the default NULL is interpreted as the system time zone). See tzone documentation for information on time zones. |
| open | a character string used to print open interval ends ("..." by default). |
| aux | a logical value, if TRUE (the default), auxiliary information (time spans of intervals) is added to the output. |
| empty | a character string used to mark empty intervals ("-" by default). |
| i | an integer vector of indices or a logical vector indicating selection. |

| | |
|---|---|
| value | replacement value, should be coercible to tinterval. |
| object | an object of tinterval class. |

## Details

tinterval constructor takes two arguments: beginnings and ends of intervals. Additional arguments (passed via ...) are forwarded to as.tind method. x %--% y is equivalent to tinterval(x, y).

as.tinterval can be used to construct time intervals from character strings, two-element lists, or two-column data frames. Additionally, as.tinterval allows to convert time intervals represented using one type of time indices to time intervals represented by time indices of higher resolution (for example months to dates).

Internally, time intervals are represented by lists of two vectors. However, in operations they behave like vectors with standard indexing and replacement operators implemented.

Interval limits can be accessed via $ operator: x$start returns vector of beginnings of intervals in x and x$end vector of ends.

For discrete time indices (represented as integers, i.e. years, quarters, months, weeks, dates, arbitrary integer indices) time interval a %--% b represents all indices falling in a or after and in b or before, i.e. the set: $\{x : a \leq x \wedge x \leq b\} = \{a, a+1, \ldots, b-1, b\}$. For continuous time indices (representing point in time, i.e. date-time, time of day, arbitrary numeric indices) time interval a %--% b represents all indices starting with a and *before* b, i.e. the set: $[a, b)$. The difference in interpretations between discrete and continuous time indices assures consistency during conversions. Consider time interval "2025-08-02" %--% "2025-08-03". This represents all date-time indices falling on one of those two days, so exactly 2025-08-02 00:00 or after but before 2025-08-04 00:00.

## Value

tinterval, %--%, and as.tinterval return objects of tinterval class.

is.tinterval returns a logical value.

In general, methods for tinterval return objects of tinterval class.

as.character and format return character vectors.

as.list and as.data.frame return a two-element list and a two-column data frame, respectively. Names are set to c("start", "end").

print returns its argument invisibly and is used for its side effect.

summary returns an object of class c("summaryDefault", "table").

## See Also

set-ops for the description of set operations on time intervals, match_t for matching time indices to time intervals.

## Examples

```
td <- today()
# from today till the day after tomorrow
td %--% (td + 2)
# from today till the end of next year
td %--% (as.year(td) + 1)
# from the beginning of the year till today
as.year(td) %--% td
# 9 to 5
as.time("9am") %--% as.time("5pm")
# 7 to 9 and 4 to 6 via constructor...
tinterval(as.time(c("7am", "4pm")), as.time(c("9am", "6pm")))
# ... or more naturally via concatenation
c(as.time("7am") %--% as.time("9am"), as.time("4pm") %--% as.time("6pm"))
# automatic parsing
as.tinterval(c("2023-01 -- 2024-06", "2024-12 -- 2025-03"))
# empty time interval
as.tinterval(c("2024-01 -- 2023-06"))
# open time interval
"2024-01" %--% NULL
"2024-01" %--% as.month(NA)
as.tinterval(c("2024-01 -- ..."))
# +/- operators
(x <- tinterval(td, td + 2))
x + c(0, 7, 14)
x %+w% 0:2
# indexing
(x <- "2023-01" %--% "2024-06")
(x <- x %+y% c(0, 2, 4))
x[2:3]
x[-1]
# beginnings and ends of intervals
x$start
x$end
# conversion from interval represented by months to dates
(x <- "2025-07" %--% "2025-08")
as.tinterval(x, "d")
# conversion from interval represented by dates to date-time (see Details)
(x <- "2025-08-02" %--% "2025-08-03")
as.tinterval(x, "t")
```

---

# ti_type *Get Time Index Type*

---

## Description

`ti_type` method returns time index type as a character value, either in short form (single letter, code used internally) or long form (name).

`is.instant` returns `TRUE` for continuous time indices representing points in time (date-time, time of day, and numeric indices) and `FALSE` for time discrete indices that represent periods of time, for example, days representing (usually) 24 hours, weeks, months, quarter, and years (as well as integer indices).

## Usage

```
ti_type(x, long = TRUE, valid = FALSE)

## S3 method for class 'tind'
ti_type(x, long = TRUE, valid = FALSE)

## S3 method for class 'Date'
ti_type(x, long = TRUE, valid = FALSE)

## S3 method for class 'POSIXt'
ti_type(x, long = TRUE, valid = FALSE)

is.instant(x)
```

## Arguments

| | |
|---|---|
| `x` | an object of `tind` class or an object coercible to it. |
| `long` | a logical value, if `FALSE`, internal single-character code of index type is returned; if `TRUE`, long (human-readable) name is returned (`TRUE` by default). |
| `valid` | a logical value, if `TRUE`, syntactically valid names will be returned (`FALSE` by default). |

## Value

A character value for `ti_type`, a logical value for `is.instant`.

## Note

Behaviour of `is.instant` differs from that of identically named function in **lubridate** package, which returns `TRUE` for all time classes including dates.

## See Also

`t_unit`.

## Examples

```
ti_type(as.tind(1999))
ti_type(as.tind(1999), FALSE)
ti_type(as.tind("2001q3"))
ti_type(as.tind("2001q3"), FALSE)
ti_type(as.tind("2003-11"))
ti_type(as.tind("2003-11"), FALSE)
ti_type(as.tind("2004-W53"))
```

```
ti_type(as.tind("2004-W53"), FALSE)
ti_type(as.tind("2020-02-29"))
ti_type(as.tind("2020-02-29"), FALSE)
ti_type(today())
ti_type(today(), FALSE)
is.instant(today())
ti_type(now())
ti_type(now(), FALSE)
is.instant(now())
ti_type(Sys.Date())
ti_type(Sys.Date(), FALSE)
is.instant(Sys.Date())
ti_type(Sys.time())
ti_type(Sys.time(), FALSE)
is.instant(Sys.time())
```

---

tspan *Determine Time Span of Time Indices and Time Intervals*

---

## Description

tspan method determines the time span of time indices and time intervals.

## Usage

```
tspan(x, ...)

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

## S3 method for class 'tind'
tspan(x, na.rm = FALSE, ...)

## S3 method for class 'tinterval'
tspan(x, ...)
```

## Arguments

| | |
|---|---|
| x | an object of tind class (or of other time index class supported by **tind** package) or an object of tinterval class. |
| ... | further arguments passed to or from other methods. |
| na.rm | a logical value indicating whether missing values should be removed. |

## Details

Given tind argument, tspan returns a single tdiff (or a single number for types "i" and "n") giving time span of indices.

tspan for tinterval argument returns a tdiff of the same length as the argument giving spans of time intervals.

Time spans are determined differently for time indices representing periods of time (for example days) and for time indices that represent points in time as determined by is.instant function. A sequence of 3 consecutive dates has span of 3 days and a sequence of 5 consecutive integers has span 5. On the other hand, a sequence of 3 consecutive hours will have span of 2 hours, the difference between the last and the first hour, see Examples.

## Value

A tdiff or a numeric vector (for integer and numeric indices).

## See Also

tinterval class, is.instant function, resolution_t method for determining the resolution of time indices, is.regular method for checking if time indices form a regular sequence.

## Examples

```
# 4 consecutive days
(x <- today() + 0:3)
tspan(x)
# 10 consecutive integers
(x <- as.tind(1:10, "i"))
tspan(x)
# 4 consecutive hours
(x <- tind(H = 12:15))
# span is 3 hours (not 4)
tspan(x)
# the same
tind(H = 15) - tind(H = 12)
```

---

# tzone *Read or Set Time Zone for Date-Time Indices*

---

## Description

Date-time indices (objects of tind class of type "t") always have the time zone attribute set. The time zone setting affects how time (measured relative to the Unix epoch in UTC) is translated to local time. Objects of base POSIXct and POSIXlt classes also have an optional time zone attribute. tzone method is also implemented for some other classes supported by the **tind** package.

List of time zones supported by a particular R installation can be obtained via a call to OlsonNames function, see Examples.

## Usage

```
tzone(x)

tzone(x) <- value

## S3 method for class 'tind'
tzone(x)

## S3 replacement method for class 'tind'
tzone(x) <- value

## S3 method for class 'tinterval'
tzone(x)

## S3 replacement method for class 'tinterval'
tzone(x) <- value

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

## S3 replacement method for class 'POSIXct'
tzone(x) <- value

## S3 method for class 'POSIXlt'
tzone(x)

## S3 replacement method for class 'POSIXlt'
tzone(x) <- value
```

## Arguments

| | |
|---|---|
| x | an object of tind class or of POSIXct/POSIXlt classes (or of other class for which the method was implemented). |
| value | a character value, the new time zone attribute. |

## Details

If the provided name is not in the list of supported time zones, an attempt is made to identify it via approximate match. If the result is a single time zone, it is accepted with a warning.

Unambiguous city names are automatically recognised, see Examples.

An attempt to set time zone attribute of a time index of different type than date-time ("t") will lead to an error. In case of POSIXct/POSIXlt objects with no time zone attribute, the extractor returns the system time zone.

## Value

For the extractor, the time zone as a character value (or NULL for classes without time zone attribute). For the replacement, the argument with the modified time zone.

## See Also

as.tzone.

## Examples

```
# check time in the system time zone
(nw <- now())
# get time zone
tzone(nw)
# set time zone to UTC
tzone(nw) <- "UTC"
nw
tzone(nw)
# check time in different time zones
if ("Asia/Tokyo" %in% OlsonNames()) {
tzone(nw) <- "Asia/Tokyo"
nw
}
if ("Europe/Warsaw" %in% OlsonNames()) {
tzone(nw) <- "Europe/Warsaw"
nw
}
if ("America/New_York" %in% OlsonNames()) {
tzone(nw) <- "America/New_York"
nw
}
# try invalid time zone => error
try(
tzone(nw) <- "Hasdfg/Qwerty"
)
# unambiguous city names are automatically recognised
tzone(nw) <- "Tokyo"
nw
tzone(nw) <- "Warsaw"
nw
tzone(nw) <- "New York"
nw
# incomplete names and approximate matches are also recognised with a warning
if ("Europe/Warsaw" %in% OlsonNames()) try({
tzone(nw) <- "Warsa"
nw
})
if ("America/New_York" %in% OlsonNames()) try({
tzone(nw) <- "NewYork"
nw
})

# list first 6 supported time zones using base::OlsonNames
head(OlsonNames())
# list first 6 supported time zones with string "Europe"
head(grep("Europe", OlsonNames(), value = TRUE))
# list first 6 supported time zones with string "Asia"
```

```
head(grep("Asia", OlsonNames(), value = TRUE))
# list first 6 supported time zones with string "Africa"
head(grep("Africa", OlsonNames(), value = TRUE))
# list first 6 supported time zones with string "America"
head(grep("America", OlsonNames(), value = TRUE))
```

---

t_unit *Get Time Unit*

---

## Description

`t_unit` method returns time unit of time difference object as a character value, either in short form (most often single letter, code used internally) or long form (name).

`units` method for `tdiff` objects is equivalent to `t_unit` with `x` argument only.

## Usage

```
t_unit(x, long = TRUE, valid = FALSE)

## S3 method for class 'tdiff'
t_unit(x, long = TRUE, valid = FALSE)

## S3 method for class 'difftime'
t_unit(x, long = TRUE, valid = FALSE)

## S3 method for class 'tdiff'
units(x)
```

## Arguments

| | |
|---|---|
| `x` | an object of `tdiff` class or an object coercible to it. |
| `long` | a logical value, if `FALSE`, internal code of time unit is returned; if `TRUE`, long (human-readable) name is returned (`TRUE` by default). |
| `valid` | a logical value, if `TRUE`, syntactically valid names will be returned (`FALSE` by default). Currently, has no impact as all unit names are syntactically valid. |

## Details

Time differences (for example differences between date-time indices) are internally represented as number of seconds. However, the returned time unit is automatically determined based on the resolution of the argument. If all time differences are full hours or full minutes, appropriate unit is returned, see Examples. For time differences that contain zeros and missing values only, returned unit is a second.

## Value

A character value.

## See Also

tdiff, ti_type.

## Examples

```
(x <- as.tdiff(1, "y"))
t_unit(x)
t_unit(x, FALSE)
(x <- as.tdiff(1, "q"))
t_unit(x)
t_unit(x, FALSE)
(x <- as.tdiff(1, "m"))
t_unit(x)
t_unit(x, FALSE)
(x <- as.tdiff(1, "w"))
t_unit(x)
t_unit(x, FALSE)
(x <- as.tdiff(1, "d"))
t_unit(x)
t_unit(x, FALSE)
(x <- as.tdiff(1, "h"))
t_unit(x)
t_unit(x, FALSE)
(x <- as.tdiff(1, "min"))
t_unit(x)
t_unit(x, FALSE)
(x <- as.tdiff(1, "s"))
t_unit(x)
t_unit(x, FALSE)
# automatic unit determination
(x <- as.tdiff(600, "s")) # ten minutes
t_unit(x)
t_unit(x, FALSE)
(x <- as.tdiff(7200, "s")) # two hours
t_unit(x)
t_unit(x, FALSE)
```

---

# year_frac *Converting Time Indices to Year Fractions and Back*

---

## Description

year_frac computes year fraction corresponding to a time index. yf2tind performs the reverse computation.

## Usage

```
year_frac(x)

yf2tind(x, type = NULL, tz = NULL)
```

## Arguments

| | |
|---|---|
| x | an object of tind class or an R object coercible to it for year_frac, a numeric vector for yf2tind. |
| type | a character value determining time index type (y - years, q - quarters, m - months, w - weeks, d - dates, t - date-time). |
| tz | (optional) a character value determining the time zone (the default NULL is interpreted as the system time zone). See tzone documentation for information on time zones. |

## Details

year_fraction returns numeric vector representing time indices in the form year + year fraction. For years this is equivalent to as.numeric. Year fraction is determined based on the index within particular year (minus 1 for all indices except for date-time) and the number of periods within the year. E.g., 2001Q1 gives 2001 (2001 + (1 - 1) / 4), 2001Q3 — 2001.5 (2001 + (3 - 1) / 4), 2010-04 (April 2010) — 2010.25 (2010 + (4 - 1) / 12), 2000-02-29 (60th day in 2000) — 2000.1612 (2000 + (60 - 1) / 366, 2000 was a leap year).

## Value

A numeric vector for year_frac and tind for yf2tind. Result is of the same length as argument.

## Note

year_frac is not to be confused with a similarly called function (YEARFRAC) found in popular spreadsheet software. In order to compute differences between dates as year fractions use daycount_frac function.

## See Also

daycount_frac.

## Examples

```
year_frac(today())
year_frac(now())
yf2tind(2023.5, "y")
yf2tind(2023.5, "q")
yf2tind(2023.5, "m")
yf2tind(2023.5, "w")
yf2tind(2023.5, "d")
yf2tind(2023.5, "t")
```

# Index