Package 'tsibble'

June 3, 2020

Type Package

Title Tidy Temporal Data Frames and Tools

Version 0.9.0

Description Provides a 'tbl_ts' class (the 'tsibble') for
   temporal data in an data- and model-oriented format. The 'tsibble'
   provides tools to easily manipulate and analyse temporal data, such as
   filling in time gaps and aggregating over calendar periods.

License GPL-3

URL https://tsibble.tidyverts.org

BugReports https://github.com/tidyverts/tsibble/issues

Depends R (>= 3.2.0)

Imports anytime (>= 0.3.1),
   dplyr (>= 1.0.0),
   ellipsis (>= 0.3.0),
   lifecycle,
   lubridate (>= 1.7.0),
   purrr (>= 0.2.3),
   rlang (>= 0.4.6),
   tibble (>= 3.0.0),
   tidyselect (>= 1.0.0),
   vctrs (>= 0.3.0)

Suggests covr,
   furrr,
   ggplot2 (>= 3.3.0),
   hms,
   knitr,
   nanotime,
   nycflights13 (>= 1.0.0),
   rmarkdown,
   scales (>= 1.1.0),
   spelling,
   testthat (>= 2.3.2),
   tidyr (>= 1.1.0),
   timeDate

VignetteBuilder knitr

RdMacros lifecycle

ByteCompile true

Encoding UTF-8

Language en-GB

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.0

R **topics documented:**

tsibble-package	*tsibble: tidy temporal data frames and tools*

Description

The **tsibble** package provides a data class of tbl_ts to represent tidy temporal data. A tsibble consists of a time index, key, and other measured variables in a data-centric format, which is built on top of the tibble.

Index

An extensive range of indices are supported by tsibble: native time classes in R (such as Date, POSIXct, and difftime) and tsibble's new additions (such as yearweek, yearmonth, and year-quarter). Some commonly-used classes have built-in support too, including ordered, hms::hms, zoo::yearmon, zoo::yearqtr, and nanotime.

For a tbl_ts of regular interval, a choice of index representation has to be made. For example, a monthly data should correspond to time index created by yearmonth or zoo::yearmon, instead of Date or POSIXct. Because months in a year ensures the regularity, 12 months every year. However, if using Date, a month containing days ranges from 28 to 31 days, which results in irregular time space. This is also applicable to year-week and year-quarter.

Tsibble supports arbitrary index classes, as long as they can be ordered from past to future. To support a custom class, you need to define index_valid() for the class and calculate the interval through interval_pull().

Key

Key variable(s) together with the index uniquely identifies each record:

- Empty: an implicit variable. NULL resulting in a univariate time series.
- A single variable: For example, data(pedestrian) use the bare Sensor as the key.
- Multiple variables: For example, Declare key = c(Region, State, Purpose) for data(tourism). Key can be created in conjunction with tidy selectors like starts_with().

Interval

The interval function returns the interval associated with the tsibble.

- Regular: the value and its time unit including "nanosecond", "microsecond", "millisecond", "second", "minute", "hour", "day", "week", "month", "quarter", "year". An unrecognisable time interval is labelled as "unit".
- Irregular: as_tsibble(regular = FALSE) gives the irregular tsibble. It is marked with !.
- Unknown: Not determined (?), if it's an empty tsibble, or one entry for each key variable.

An interval is obtained based on the corresponding index representation:

- integerish numerics between 1582 and 2499: "year" (Y). Note the year of 1582 saw the beginning of the Gregorian Calendar switch.
- yearquarter/yearqtr: "quarter" (Q)
- yearmonth/yearmon: "month" (M)
- yearweek: "week" (W)

- `Date`: "day" (D)
- `difftime`: "week" (W), "day" (D), "hour" (h), "minute" (m), "second" (s)
- `POSIXt/hms`: "hour" (h), "minute" (m), "second" (s), "millisecond" (us), "microsecond" (ms)
- `nanotime`: "nanosecond" (ns)
- other numerics &`ordered` (ordered factor): "unit" When the interval cannot be obtained due to the mismatched index format, an error is issued.

The interval is invariant to subsetting, such as `filter()`, `slice()`, and [.tbl_ts. But if the result is an empty tsibble, the interval is always unknown. When joining a tsibble with other data sources and aggregating to different time scales, the interval gets re-calculated.

Time zone

Time zone corresponding to index will be displayed if index is `POSIXct`. ? means that the obtained time zone is a zero-length character "".

Print options

The tsibble package fully utilises the `print` method from the tibble. Please refer to tibble::tibble-package to change display options.

Author(s)

Maintainer: Earo Wang <earo.wang@gmail.com> (ORCID)

Authors:

- Di Cook (ORCID) [thesis advisor]
- Rob Hyndman (ORCID) [thesis advisor]
- Mitchell O'Hara-Wild (ORCID)

See Also

Useful links:

- https://tsibble.tidyverts.org
- Report bugs at https://github.com/tidyverts/tsibble/issues

Examples

```
# create a tsibble w/o a key ----
tsibble(
  date = as.Date("2017-01-01") + 0:9,
  value = rnorm(10)
)

# create a tsibble with one key ----
tsibble(
  qtr = rep(yearquarter("2010-01") + 0:9, 3),
  group = rep(c("x", "y", "z"), each = 10),
  value = rnorm(30),
  key = group
)
```

---

as.ts.tbl_ts   *Coerce a tsibble to a time series*

Description

Stable

Usage

```
## S3 method for class 'tbl_ts'
as.ts(x, value, frequency = NULL, fill = NA_real_, ...)
```

Arguments

x	A tbl_ts object.
value	A measured variable of interest to be spread over columns, if multiple measures.
frequency	A smart frequency with the default NULL. If set, the preferred frequency is passed to ts().
fill	A value to replace missing values.
...	Ignored for the function.

Value

A ts object.

Examples

```
# a monthly series
x1 <- as_tsibble(AirPassengers)
as.ts(x1)
```

---

as_tibble.tbl_ts   *Coerce to a tibble or data frame*

Description

Coerce to a tibble or data frame

Usage

```
## S3 method for class 'tbl_ts'
as_tibble(x, ...)
```

Arguments

x	A tbl_ts.
...	Ignored.

Examples

```
as_tibble(pedestrian)
```

---

as_tsibble   *Coerce to a tsibble object*

Description

Stable

Usage

```
as_tsibble(
  x,
  key = NULL,
  index,
  regular = TRUE,
  validate = TRUE,
  .drop = TRUE,
  ...
)

## S3 method for class 'ts'
as_tsibble(x, ..., tz = "UTC")

## S3 method for class 'mts'
as_tsibble(x, ..., tz = "UTC", pivot_longer = TRUE)
```

Arguments

x	Other objects to be coerced to a tsibble (tbl_ts).
key	Unquoted variable(s) that uniquely determine time indices. NULL for empty key, and c() for multiple variables. It works with tidy selector (e.g. dplyr::starts_with()).
index	A bare (or unquoted) variable to specify the time index variable.
regular	Regular time interval (TRUE) or irregular (FALSE). The interval is determined by the greatest common divisor of index column, if TRUE.
validate	TRUE suggests to verify that each key or each combination of key variables leads to unique time indices (i.e. a valid tsibble). If you are sure that it's a valid input, specify FALSE to skip the checks.
.drop	If TRUE, empty key groups are dropped.
...	Other arguments passed on to individual methods.
tz	Time zone. May be useful when a ts object is more frequent than daily.
pivot_longer	TRUE gives a "longer" form of the data, otherwise as is.

Details

A tsibble is sorted by its key first and index.

Value

A tsibble object.

Index

An extensive range of indices are supported by tsibble: native time classes in R (such as Date, POSIXct, and difftime) and tsibble's new additions (such as [yearweek](), [yearmonth](), and [year-quarter]()). Some commonly-used classes have built-in support too, including ordered, hms::hms, zoo::yearmon, zoo::yearqtr, and nanotime.

For a tbl_ts of regular interval, a choice of index representation has to be made. For example, a monthly data should correspond to time index created by [yearmonth]() or zoo::yearmon, instead of Date or POSIXct. Because months in a year ensures the regularity, 12 months every year. However, if using Date, a month containing days ranges from 28 to 31 days, which results in irregular time space. This is also applicable to year-week and year-quarter.

Tsibble supports arbitrary index classes, as long as they can be ordered from past to future. To support a custom class, you need to define [index_valid()]() for the class and calculate the interval through [interval_pull()]().

Key

Key variable(s) together with the index uniquely identifies each record:

- Empty: an implicit variable. NULL resulting in a univariate time series.
- A single variable: For example, data(pedestrian) use the bare Sensor as the key.
- Multiple variables: For example, Declare key = c(Region, State, Purpose) for data(tourism). Key can be created in conjunction with tidy selectors like starts_with().

Interval

The [interval]() function returns the interval associated with the tsibble.

- Regular: the value and its time unit including "nanosecond", "microsecond", "millisecond", "second", "minute", "hour", "day", "week", "month", "quarter", "year". An unrecognisable time interval is labelled as "unit".
- Irregular: as_tsibble(regular = FALSE) gives the irregular tsibble. It is marked with !.
- Unknown: Not determined (?), if it's an empty tsibble, or one entry for each key variable.

An interval is obtained based on the corresponding index representation:

- integerish numerics between 1582 and 2499: "year" (Y). Note the year of 1582 saw the beginning of the Gregorian Calendar switch.
- yearquarter/yearqtr: "quarter" (Q)
- yearmonth/yearmon: "month" (M)
- yearweek: "week" (W)
- Date: "day" (D)
- difftime: "week" (W), "day" (D), "hour" (h), "minute" (m), "second" (s)
- POSIXt/hms: "hour" (h), "minute" (m), "second" (s), "millisecond" (us), "microsecond" (ms)
- nanotime: "nanosecond" (ns)
- other numerics &ordered (ordered factor): "unit" When the interval cannot be obtained due to the mismatched index format, an error is issued.

The interval is invariant to subsetting, such as filter(), slice(), and [.tbl_ts. But if the result is an empty tsibble, the interval is always unknown. When joining a tsibble with other data sources and aggregating to different time scales, the interval gets re-calculated.

See Also

tsibble

Examples

```
# coerce tibble to tsibble w/o a key
tbl1 <- tibble(
  date = as.Date("2017-01-01") + 0:9,
  value = rnorm(10)
)
as_tsibble(tbl1)
# supply the index to suppress the message
as_tsibble(tbl1, index = date)

# coerce tibble to tsibble with a single variable for key
# use `yearquarter()` to represent quarterly data
tbl2 <- tibble(
  qtr = rep(yearquarter("2010 Q1") + 0:9, 3),
  group = rep(c("x", "y", "z"), each = 10),
  value = rnorm(30)
)
# "qtr" is automatically considered as the index var
as_tsibble(tbl2, key = group)
as_tsibble(tbl2, key = group, index = qtr)

# create a tsibble with multiple variables for key
# use `yearmonth()` to represent monthly data
tbl3 <- tibble(
  mth = rep(yearmonth("2010 Jan") + 0:8, each = 3),
  xyz = rep(c("x", "y", "z"), each = 9),
  abc = rep(letters[1:3], times = 9),
  value = rnorm(27)
)
as_tsibble(tbl3, key = c(xyz, abc))
# coerce ts to tsibble
as_tsibble(AirPassengers)
as_tsibble(sunspot.year)
as_tsibble(sunspot.month)
as_tsibble(austres)
# coerce mts to tsibble
z <- ts(matrix(rnorm(300), 100, 3), start = c(1961, 1), frequency = 12)
as_tsibble(z)
as_tsibble(z, pivot_longer = FALSE)
```

---

build_tsibble	*Low-level constructor for a tsibble object*

Description

`build_tsibble()` creates a `tbl_ts` object with more controls. It is useful for creating a `tbl_ts` internally inside a function, and it allows developers to determine if the time needs ordering and the interval needs calculating.

Usage

```
build_tsibble(
  x,
  key = NULL,
  key_data = NULL,
  index,
  index2 = index,
  ordered = NULL,
  interval = TRUE,
  validate = TRUE,
  .drop = key_drop_default(x)
)
```

Arguments

x	A data.frame, tbl_df, tbl_ts, or other tabular objects.
key	Unquoted variable(s) that uniquely determine time indices. NULL for empty key, and c() for multiple variables. It works with tidy selector (e.g. dplyr::starts_with()).
key_data	A data frame containing key variables and .rows. When a data frame is supplied, the argument key will be ignored.
index	A bare (or unquoted) variable to specify the time index variable.
index2	A candidate of index to update the index to a new one when index_by. By default, it's identical to index.
ordered	The default of NULL arranges the key variable(s) first and then index from past to future. TRUE suggests to skip the ordering as x in the correct order. FALSE checks the ordering and may give a warning.
interval	TRUE automatically calculates the interval, and FALSE for irregular interval. Use the specified interval via new_interval() as is.
validate	TRUE suggests to verify that each key or each combination of key variables leads to unique time indices (i.e. a valid tsibble). If you are sure that it's a valid input, specify FALSE to skip the checks.
.drop	If TRUE, empty key groups are dropped.

Examples

```
# Prepare `pedestrian` to use a new index `Date` ----
pedestrian %>%
  build_tsibble(
    key = !!key_vars(.), index = !!index(.), index2 = Date,
    interval = interval(.)
  )
```

---

count_gaps	*Count implicit gaps*

Description

Count implicit gaps

Usage

```
count_gaps(.data, .full = FALSE, .name = c(".from", ".to", ".n"))
```

Arguments

`.data`	A tsibble.
`.full`	• FALSE inserts NA for each keyed unit within its own period.
	• TRUE fills NA over the entire time span of the data (a.k.a. fully balanced panel).
	• `start()` pad NA to the same starting point (i.e. min(<index>)) across units.
	• `end()` pad NA to the same ending point (i.e. max(<index>)) across units.
`.name`	Strings to name new columns.

Value

A tibble contains:

- the "key" of the `tbl_ts`
- ".from": the starting time point of the gap
- ".to": the ending time point of the gap
- ".n": the number of implicit missing observations during the time period

See Also

Other implicit gaps handling: fill_gaps(), has_gaps(), scan_gaps()

Examples

```
ped_gaps <- pedestrian %>%
  count_gaps(.full = TRUE)
ped_gaps
if (!requireNamespace("ggplot2", quietly = TRUE)) {
  stop("Please install the ggplot2 package to run these following examples.")
}
library(ggplot2)
ggplot(ped_gaps, aes(x = Sensor, colour = Sensor)) +
  geom_linerange(aes(ymin = .from, ymax = .to)) +
  geom_point(aes(y = .from)) +
  geom_point(aes(y = .to)) +
  coord_flip() +
  theme(legend.position = "bottom")
```

---

difference	*Lagged differences*

Description

Stable

Usage

```
difference(x, lag = 1, differences = 1, default = NA, order_by = NULL)
```

Arguments

x	Vector of values
lag	A positive integer indicating which lag to use.
differences	A positive integer indicating the order of the difference.
default	Value used for non-existent rows. Defaults to NA.
order_by	Override the default ordering to use another vector or column

Value

A numeric vector of the same length as x.

See Also

dplyr::lead and dplyr::lag

Examples

```
# examples from base
difference(1:10, 2)
difference(1:10, 2, 2)
x <- cumsum(cumsum(1:10))
difference(x, lag = 2)
difference(x, differences = 2)
# Use order_by if data not already ordered (example from dplyr)
library(dplyr, warn.conflicts = FALSE)
tsbl <- tsibble(year = 2000:2005, value = (0:5)^2, index = year)
scrambled <- tsbl %>% slice(sample(nrow(tsbl)))

wrong <- mutate(scrambled, diff = difference(value))
arrange(wrong, year)

right <- mutate(scrambled, diff = difference(value, order_by = year))
arrange(right, year)
```

---

fill_gaps	*Turn implicit missing values into explicit missing values*

Description

Stable

Usage

```
fill_gaps(.data, ..., .full = FALSE)
```

Arguments

`.data`	A tsibble.
`...`	A set of name-value pairs. The values provided will only replace missing values that were marked as "implicit", and will leave previously existing NA untouched.

- empty: filled with default NA.
- filled by values or functions.

`.full`	

- FALSE inserts NA for each keyed unit within its own period.
- TRUE fills NA over the entire time span of the data (a.k.a. fully balanced panel).
- start() pad NA to the same starting point (i.e. min(<index>)) across units.
- end() pad NA to the same ending point (i.e. max(<index>)) across units.

See Also

tidyr::fill, tidyr::replace_na for handling missing values NA.

Other implicit gaps handling: count_gaps(), has_gaps(), scan_gaps()

Examples

```
harvest <- tsibble(
  year = c(2010, 2011, 2013, 2011, 2012, 2014),
  fruit = rep(c("kiwi", "cherry"), each = 3),
  kilo = sample(1:10, size = 6),
  key = fruit, index = year
)

# gaps as default `NA`
fill_gaps(harvest, .full = TRUE)
fill_gaps(harvest, .full = start())
fill_gaps(harvest, .full = end())
full_harvest <- fill_gaps(harvest, .full = FALSE)
full_harvest

# replace gaps with a specific value
harvest %>%
  fill_gaps(kilo = 0L)

# replace gaps using a function by variable
harvest %>%
  fill_gaps(kilo = sum(kilo))

# replace gaps using a function for each group
harvest %>%
  group_by_key() %>%
  fill_gaps(kilo = sum(kilo))

# leaves existing `NA` untouched
harvest[2, 3] <- NA
harvest %>%
  group_by_key() %>%
  fill_gaps(kilo = sum(kilo, na.rm = TRUE))

# replace NA
```

```
pedestrian %>%
  group_by_key() %>%
  fill_gaps(Count = as.integer(median(Count)))

if (!requireNamespace("tidyr", quietly = TRUE)) {
  stop("Please install the 'tidyr' package to run these following examples.")
}
# use fill() to fill `NA` by previous/next entry
pedestrian %>%
  group_by_key() %>%
  fill_gaps() %>%
  tidyr::fill(Count, .direction = "down")
```

---

filter_index	*A shorthand for filtering time index for a tsibble*

Description

This shorthand respects time zones and encourages compact expressions.

Usage

```
filter_index(.data, ..., .preserve = FALSE)
```

Arguments

`.data`	A tsibble.
`...`	Formulas that specify start and end periods (inclusive), or strings.

- `~ end` or `. ~ end`: from the very beginning to a specified ending period.
- `start ~ end`: from specified beginning to ending periods.
- `start ~ .`: from a specified beginning to the very end of the data. Supported index type: POSIXct (to seconds), Date, yearweek, yearmonth/yearmon, yearquarter/yearqtr, hms/difftime & numeric.

`.preserve`	Relevant when the `.data` input is grouped. If `.preserve = FALSE` (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.

System Time Zone ("Europe/London")

There is a known issue of an extra hour gained for a machine setting time zone to "Europe/London", regardless of the time zone associated with the POSIXct inputs. It relates to *anytime* and *Boost*. Use `Sys.timezone()` to check if the system time zone is "Europe/London". It would be recommended to change the global environment "TZ" to other equivalent names: GB, GB-Eire, Europe/Belfast, Europe/Guernsey, Europe/Isle_of_Man and Europe/Jersey as documented in `?Sys.timezone()`, using `Sys.setenv(TZ = "GB")` for example.

See Also

[time_in](#) for a vector of time index

Examples

```
# from the starting time to the end of Feb, 2015
pedestrian %>%
  filter_index(~ "2015-02")

# entire Feb 2015, & from the beginning of Aug 2016 to the end
pedestrian %>%
  filter_index("2015-02", "2016-08" ~ .)

# multiple time windows
pedestrian %>%
  filter_index(~"2015-02", "2015-08" ~ "2015-09", "2015-12" ~ "2016-02")

# entire 2015
pedestrian %>%
  filter_index("2015")

# specific
pedestrian %>%
  filter_index("2015-03-23" ~ "2015-10")
pedestrian %>%
  filter_index("2015-03-23" ~ "2015-10-31")
pedestrian %>%
  filter_index("2015-03-23 10" ~ "2015-10-31 12")
```

---

group_by_key   *Group by key variables*

Description

Stable

Usage

```
group_by_key(.data, ..., .drop = key_drop_default(.data))
```

Arguments

.data	A `tbl_ts` object.
...	Ignored.
.drop	When .drop = TRUE, empty groups are dropped. See [group_by_drop_default()](#) for what the default value is for this argument.

Examples

```
tourism %>%
  group_by_key()
```

---

guess_frequency | *Guess a time frequency from other index objects*

Description

Stable

A possible frequency passed to the `ts()` function

Usage

```
guess_frequency(x)
```

Arguments

x   An index object including "yearmonth", "yearquarter", "Date" and others.

Details

If a series of observations are collected more frequently than weekly, it is more likely to have multiple seasonalities. This function returns a frequency value at its smallest. For example, hourly data would have daily, weekly and annual frequencies of 24, 168 and 8766 respectively, and hence it gives 24.

References

<https://robjhyndman.com/hyndsight/seasonal-periods/>

Examples

```
guess_frequency(yearquarter("2016 Q1") + 0:7)
guess_frequency(yearmonth("2016 Jan") + 0:23)
guess_frequency(seq(as.Date("2017-01-01"), as.Date("2017-01-31"), by = 1))
guess_frequency(seq(
  as.POSIXct("2017-01-01 00:00"), as.POSIXct("2017-01-10 23:00"),
  by = "1 hour"
))
```

---

has_gaps | *Does a tsibble have implicit gaps in time?*

Description

Does a tsibble have implicit gaps in time?

Usage

```
has_gaps(.data, .full = FALSE, .name = ".gaps")
```

Arguments

`.data`	A tsibble.
`.full`	• `FALSE` inserts NA for each keyed unit within its own period.
	• `TRUE` fills NA over the entire time span of the data (a.k.a. fully balanced panel).
	• `start()` pad NA to the same starting point (i.e. min(<index>)) across units.
	• `end()` pad NA to the same ending point (i.e. max(<index>)) across units.
`.name`	Strings to name new columns.

Value

A tibble contains "key" variables and new column `.gaps` of TRUE/FALSE.

See Also

Other implicit gaps handling: `count_gaps()`, `fill_gaps()`, `scan_gaps()`

Examples

```
harvest <- tsibble(
  year = c(2010, 2011, 2013, 2011, 2012, 2013),
  fruit = rep(c("kiwi", "cherry"), each = 3),
  kilo = sample(1:10, size = 6),
  key = fruit, index = year
)
has_gaps(harvest)
has_gaps(harvest, .full = TRUE)
has_gaps(harvest, .full = start())
has_gaps(harvest, .full = end())
```

---

holiday_aus   *Australian national and state-based public holiday*

Description

Australian national and state-based public holiday

Usage

```
holiday_aus(year, state = "national")
```

Arguments

`year`	A vector of integer(s) indicating year(s).
`state`	A state in Australia including "ACT", "NSW", "NT", "QLD", "SA", "TAS", "VIC", "WA", as well as "national".

Details

Not documented public holidays:

- AFL public holidays for Victoria
- Queen's Birthday for Western Australia
- Royal Queensland Show for Queensland, which is for Brisbane only

This function requires "timeDate" to be installed.

Value

A tibble consisting of `holiday` labels and their associated dates in the year(s).

References

[Public holidays]{style="color:red"}

Examples

```
holiday_aus(2016, state = "VIC")
holiday_aus(2013:2016, state = "ACT")
```

---

index   *Return index variable from a tsibble*

Description

Return index variable from a tsibble

Usage

```
index(x)

index_var(x)

index2(x)

index2_var(x)
```

Arguments

x   A tsibble object.

Examples

```
index(pedestrian)
index_var(pedestrian)
```

---

index_by   *Group by time index and collapse with* summarise()

Description

Stable

index_by() is the counterpart of group_by() in temporal context, but it only groups the time index.
The following operation is applied to each partition of the index, similar to group_by() but dealing
with index only. index_by() + summarise() will update the grouping index variable to be the new
index. Use ungroup() to remove the index grouping vars.

Usage

```
index_by(.data, ...)
```

Arguments

.data	A tbl_ts.
...	If empty, grouping the current index. If not empty, a single expression is required for either an existing variable or a name-value pair. A lambda expression is supported, for example ~ as.Date(.) where . refers to the index variable. The index functions that can be used, but not limited:

- lubridate::year: yearly aggregation
- yearquarter: quarterly aggregation
- yearmonth: monthly aggregation
- yearweek: weekly aggregation
- as.Date or lubridate::as_date: daily aggregation
- lubridate::ceiling_date, lubridate::floor_date, or lubridate::round_date: fine-resolution aggregation
- Extract time components functions, such as lubridate::hour() & lubridate::day()
- other index functions from other packages or self-defined functions

Details

- A index_by()-ed tsibble is indicated by @ in the "Groups" when displaying on the screen.

Examples

```
pedestrian %>% index_by()
# Monthly counts across sensors
library(dplyr, warn.conflicts = FALSE)
monthly_ped <- pedestrian %>%
  group_by_key() %>%
  index_by(Year_Month = ~ yearmonth(.)) %>%
  summarise(
    Max_Count = max(Count),
    Min_Count = min(Count)
  )
monthly_ped
index(monthly_ped)
```

```
# Using existing variable
pedestrian %>%
  group_by_key() %>%
  index_by(Date) %>%
  summarise(
    Max_Count = max(Count),
    Min_Count = min(Count)
  )

# Attempt to aggregate to 4-hour interval, with the effects of DST
pedestrian %>%
  group_by_key() %>%
  index_by(Date_Time4 = ~ lubridate::floor_date(., "4 hour")) %>%
  summarise(Total_Count = sum(Count))

library(lubridate, warn.conflicts = FALSE)
# Annual trips by Region and State
tourism %>%
  index_by(Year = ~ year(.)) %>%
  group_by(Region, State) %>%
  summarise(Total = sum(Trips))

# Rouding to financial year, using a custom function
financial_year <- function(date) {
  year <- year(date)
  ifelse(quarter(date) <= 2, year, year + 1)
}
tourism %>%
  index_by(Year = ~ financial_year(.)) %>%
  summarise(Total = sum(Trips))
```

---

index_valid   *Add custom index support for a tsibble*

Description

Stable

S3 method to add an index type support for a tsibble.

Usage

```
index_valid(x)
```

Arguments

x   An object of index type that the tsibble supports.

Details

This method is primarily used for adding an index type support in as_tsibble.

Value

TRUE/FALSE or NA (unsure)

See Also

interval_pull for obtaining interval for regularly spaced time.

Examples

```
index_valid(seq(as.Date("2017-01-01"), as.Date("2017-01-10"), by = 1))
```

---

interval   *Meta-information of a tsibble*

Description

- interval() returns an interval of a tsibble.
- is_regular checks if a tsibble is spaced at regular time or not.
- is_ordered checks if a tsibble is ordered by key and index.

Usage

```
interval(x)

is_regular(x)

is_ordered(x)
```

Arguments

x   A tsibble object.

Examples

```
interval(pedestrian)
is_regular(pedestrian)
is_ordered(pedestrian)
```

---

interval_pull   *Pull time interval from a vector*

Description

Stable

Assuming regularly spaced time, the interval_pull() returns a list of time components as the "interval" class.

Usage

```
interval_pull(x)
```

Arguments

x	A vector of `POSIXct`, `Date`, `yearweek`, `yearmonth`, `yearquarter`, `difftime/hms`, `ordered`, `integer`, `numeric`, and `nanotime`.

Details

Extend tsibble to support custom time indexes by defining S3 generics `index_valid()` and `interval_pull()` for them.

Value

an "interval" class (a list) includes "year", "quarter", "month", "week", "day", "hour", "minute", "second", "millisecond", "microsecond", "nanosecond", "unit".

Examples

```
x <- seq(as.Date("2017-10-01"), as.Date("2017-10-31"), by = 3)
interval_pull(x)
```

---

is_duplicated	*Test duplicated observations determined by key and index variables*

Description

Stable

- `is_duplicated()`: a logical scalar if the data exist duplicated observations.
- `are_duplicated()`: a logical vector, the same length as the row number of `data`.
- `duplicates()`: identical key-index data entries.

Usage

```
is_duplicated(data, key = NULL, index)

are_duplicated(data, key = NULL, index, from_last = FALSE)

duplicates(data, key = NULL, index)
```

Arguments

data	A data frame for creating a tsibble.
key	Unquoted variable(s) that uniquely determine time indices. NULL for empty key, and `c()` for multiple variables. It works with tidy selector (e.g. [dplyr::starts_with()](#)).
index	A bare (or unquoted) variable to specify the time index variable.
from_last	TRUE does the duplication check from the last of identical elements.

Examples

```
harvest <- tibble(
  year = c(2010, 2011, 2013, 2011, 2012, 2014, 2014),
  fruit = c(rep(c("kiwi", "cherry"), each = 3), "cherry"),
  kilo = sample(1:10, size = 7)
)
is_duplicated(harvest, key = fruit, index = year)
are_duplicated(harvest, key = fruit, index = year)
are_duplicated(harvest, key = fruit, index = year, from_last = TRUE)
duplicates(harvest, key = fruit, index = year)
```

---

is_tsibble   *If the object is a tsibble*

Description

Stable

Usage

```
is_tsibble(x)

is_grouped_ts(x)
```

Arguments

x   An object.

Value

TRUE if the object inherits from the tbl_ts class.

Examples

```
# A tibble is not a tsibble ----
tbl <- tibble(
  date = seq(as.Date("2017-10-01"), as.Date("2017-10-31"), by = 1),
  value = rnorm(31)
)
is_tsibble(tbl)

# A tsibble ----
tsbl <- as_tsibble(tbl, index = date)
is_tsibble(tsbl)
```

---

key *Return key variables*

Description

key() returns a list of symbols; key_vars() gives a character vector.

Usage

```
key(x)

key_vars(x)
```

Arguments

x   A tsibble.

Examples

```
key(pedestrian)
key_vars(pedestrian)

key(tourism)
key_vars(tourism)
```

---

measures *Return measured variables*

Description

Return measured variables

Usage

```
measures(x)

measured_vars(x)
```

Arguments

x   A tbl_ts.

Examples

```
measures(pedestrian)
measures(tourism)

measured_vars(pedestrian)
measured_vars(tourism)
```

---

new_data   *New tsibble data and append new observations to a tsibble*

Description

Stable

append_row(): add new rows to the start/end of a tsibble by filling a key-index pair and NA for measured variables.

append_case() is an alias of append_row().

Usage

```
new_data(.data, n = 1L, ...)

## S3 method for class 'tbl_ts'
new_data(.data, n = 1L, keep_all = FALSE, ...)

append_row(.data, n = 1L, ...)
```

Arguments

.data	A tbl_ts.
n	An integer indicates the number of key-index pair to append. If
	• n > 0, future observations
	• n < 0, past observations
...	Passed to individual S3 method.
keep_all	If TRUE keep all the measured variables as well as index and key, otherwise only index and key.

Examples

```
new_data(pedestrian)
new_data(pedestrian, keep_all = TRUE)
new_data(pedestrian, n = 3)
new_data(pedestrian, n = -2)

tsbl <- tsibble(
  date = rep(as.Date("2017-01-01") + 0:2, each = 2),
  group = rep(letters[1:2], 3),
  value = rnorm(6),
  key = group
)
append_row(tsbl)
append_row(tsbl, n = 2)
append_row(tsbl, n = -2)
```

---

new_interval | *Interval constructor for a tsibble*

Description

Stable

- new_interval() creates an interval object.
- is_regular_interval() checks if the interval is regular.

Usage

```
new_interval(..., .regular = TRUE, .others = list())

is_regular_interval(x)
```

Arguments

...	A set of name-value pairs to specify default interval units: "year", "quarter", "month", "week", "day", "hour", "minute", "second", "millisecond", "microsecond", "nanosecond", "unit".
.regular	Logical. FALSE gives an irregular interval, and will ignore the ... argument.
.others	A list name-value pairs that are not included in the ..., to allow custom interval.
x	An interval.

Value

an "interval" class

Examples

```
(x <- new_interval(hour = 1, minute = 30))
(y <- new_interval(.regular = FALSE)) # irregular interval
new_interval() # unknown interval
new_interval(.others = list(semester = 1)) # custom interval
is_regular_interval(x)
is_regular_interval(y)
```

---

new_tsibble | *Create a subclass of a tsibble*

Description

Create a subclass of a tsibble

Usage

```
new_tsibble(x, ..., class = NULL)
```

Arguments

`x`	A `tbl_ts`, required.
`...`	Name-value pairs defining new attributes other than a tsibble.
`class`	Subclasses to assign to the new object, default: none.

---

pedestrian	*Pedestrian counts in the city of Melbourne*

Description

A dataset containing the hourly pedestrian counts from 2015-01-01 to 2016-12-31 at 4 sensors in the city of Melbourne.

Usage

```
pedestrian
```

Format

A tsibble with 66,071 rows and 5 variables:

- **Sensor**: Sensor names (key)
- **Date_Time**: Date time when the pedestrian counts are recorded (index)
- **Date**: Date when the pedestrian counts are recorded
- **Time**: Hour associated with Date_Time
- **Counts**: Hourly pedestrian counts

References

[Melbourne Open Data Portal](#)

Examples

```
library(dplyr)
data(pedestrian)
# make implicit missingness to be explicit ----
pedestrian %>% fill_gaps()
# compute daily maximum counts across sensors ----
pedestrian %>%
  group_by_key() %>%
  index_by(Date) %>% # group by Date and use it as new index
  summarise(MaxC = max(Count))
```

---

scan_gaps   *Scan a tsibble for implicit missing observations*

Description

Scan a tsibble for implicit missing observations

Usage

```
scan_gaps(.data, .full = FALSE)
```

Arguments

.data   A tsibble.

.full
- FALSE inserts NA for each keyed unit within its own period.
- TRUE fills NA over the entire time span of the data (a.k.a. fully balanced panel).
- start() pad NA to the same starting point (i.e. min(<index>)) across units.
- end() pad NA to the same ending point (i.e. max(<index>)) across units.

See Also

Other implicit gaps handling: [count_gaps()](), [fill_gaps()](), [has_gaps()]()

Examples

```
scan_gaps(pedestrian)
```

---

slide_tsibble   *Perform sliding windows on a tsibble by row*

Description

Questioning

Usage

```
slide_tsibble(.x, .size = 1, .step = 1, .id = ".id")
```

Arguments

.x   A tsibble.

.size   A positive integer for window size.

.step   A positive integer for calculating at every specified step instead of every single step.

.id   A character naming the new column .id containing the partition.

Rolling tsibble

slide_tsibble(), tile_tsibble(), and stretch_tsibble() provide fast and shorthand for rolling over a tsibble by observations. That said, if the supplied tsibble has time gaps, these rolling helpers will ignore those gaps and proceed.

They are useful for preparing the tsibble for time series cross validation. They all return a tsibble including a new column .id as part of the key. The output dimension will increase considerably with slide_tsibble() and stretch_tsibble(), which is likely to run out of memory when the data is large.

See Also

Other rolling tsibble: stretch_tsibble(), tile_tsibble()

Examples

```
harvest <- tsibble(
  year = rep(2010:2012, 2),
  fruit = rep(c("kiwi", "cherry"), each = 3),
  kilo = sample(1:10, size = 6),
  key = fruit, index = year
)
harvest %>%
  slide_tsibble(.size = 2)
```

---

stretch_tsibble   *Perform stretching windows on a tsibble by row*

Description

Questioning

Usage

```
stretch_tsibble(.x, .step = 1, .init = 1, .id = ".id")
```

Arguments

.x	A tsibble.
.step	A positive integer for incremental step.
.init	A positive integer for an initial window size.
.id	A character naming the new column .id containing the partition.

Rolling tsibble

slide_tsibble(), tile_tsibble(), and stretch_tsibble() provide fast and shorthand for rolling over a tsibble by observations. That said, if the supplied tsibble has time gaps, these rolling helpers will ignore those gaps and proceed.

They are useful for preparing the tsibble for time series cross validation. They all return a tsibble including a new column .id as part of the key. The output dimension will increase considerably with slide_tsibble() and stretch_tsibble(), which is likely to run out of memory when the data is large.

See Also

Other rolling tsibble: slide_tsibble(), tile_tsibble()

Examples

```
harvest <- tsibble(
  year = rep(2010:2012, 2),
  fruit = rep(c("kiwi", "cherry"), each = 3),
  kilo = sample(1:10, size = 6),
  key = fruit, index = year
)
harvest %>%
  stretch_tsibble()
```

---

tile_tsibble   *Perform tiling windows on a tsibble by row*

Description

Questioning

Usage

```
tile_tsibble(.x, .size = 1, .id = ".id")
```

Arguments

.x	A tsibble.
.size	A positive integer for window size.
.id	A character naming the new column .id containing the partition.

Rolling tsibble

slide_tsibble(), tile_tsibble(), and stretch_tsibble() provide fast and shorthand for rolling over a tsibble by observations. That said, if the supplied tsibble has time gaps, these rolling helpers will ignore those gaps and proceed.

They are useful for preparing the tsibble for time series cross validation. They all return a tsibble including a new column .id as part of the key. The output dimension will increase considerably with slide_tsibble() and stretch_tsibble(), which is likely to run out of memory when the data is large.

See Also

Other rolling tsibble: slide_tsibble(), stretch_tsibble()

Examples

```
harvest <- tsibble(
  year = rep(2010:2012, 2),
  fruit = rep(c("kiwi", "cherry"), each = 3),
  kilo = sample(1:10, size = 6),
  key = fruit, index = year
)
harvest %>%
  tile_tsibble(.size = 2)
```

---

time_in   *If time falls in the ranges using compact expressions*

Description

This function respects time zone and encourages compact expressions.

Usage

```
time_in(x, ...)
```

Arguments

x   A vector of time index, such as classes POSIXct, Date, yearweek, yearmonth,
                yearquarter, hms/difftime, and numeric.

...   Formulas that specify start and end periods (inclusive), or strings.

                • ~ end or . ~ end: from the very beginning to a specified ending period.
                • start ~ end: from specified beginning to ending periods.
                • start ~ .: from a specified beginning to the very end of the data. Sup-
                  ported index type: POSIXct (to seconds), Date, yearweek, yearmonth/yearmon,
                  yearquarter/yearqtr, hms/difftime & numeric.

Value

logical vector

System Time Zone ("Europe/London")

There is a known issue of an extra hour gained for a machine setting time zone to "Europe/London",
regardless of the time zone associated with the POSIXct inputs. It relates to *anytime* and *Boost*. Use
Sys.timezone() to check if the system time zone is "Europe/London". It would be recommended
to change the global environment "TZ" to other equivalent names: GB, GB-Eire, Europe/Belfast,
Europe/Guernsey, Europe/Isle_of_Man and Europe/Jersey as documented in ?Sys.timezone(),
using Sys.setenv(TZ = "GB") for example.

See Also

[filter_index](#) for filtering tsibble

Examples

```
x <- unique(pedestrian$Date_Time)
lgl <- time_in(x, ~"2015-02", "2015-08" ~ "2015-09", "2015-12" ~ "2016-02")
lgl[1:10]
# more specific
lgl2 <- time_in(x, "2015-03-23 10" ~ "2015-10-31 12")
lgl2[1:10]

library(dplyr)
pedestrian %>%
  filter(time_in(Date_Time, "2015-03-23 10" ~ "2015-10-31 12"))
pedestrian %>%
  filter(time_in(Date_Time, "2015")) %>%
  mutate(Season = ifelse(
    time_in(Date_Time, "2015-03" ~ "2015-08"),
    "Autumn-Winter", "Spring-Summer"
  ))
```

---

tourism   *Australian domestic overnight trips*

Description

A dataset containing the quarterly overnight trips from 1998 Q1 to 2016 Q4 across Australia.

Usage

```
tourism
```

Format

A tsibble with 23,408 rows and 5 variables:

- **Quarter**: Year quarter (index)
- **Region**: The tourism regions are formed through the aggregation of Statistical Local Areas (SLAs) which are defined by the various State and Territory tourism authorities according to their research and marketing needs
- **State**: States and territories of Australia
- **Purpose**: Stopover purpose of visit:
    - "Holiday"
    - "Visiting friends and relatives"
    - "Business"
    - "Other reason"
- **Trips**: Overnight trips in thousands

References

Tourism Research Australia

Examples

```
library(dplyr)
data(tourism)
# Total trips over geographical regions
tourism %>%
  group_by(Region, State) %>%
  summarise(Total_Trips = sum(Trips))
```

---

tsibble   *Create a tsibble object*

Description

Stable

Usage

```
tsibble(..., key = NULL, index, regular = TRUE, .drop = TRUE)
```

Arguments

`...`	A set of name-value pairs.
`key`	Unquoted variable(s) that uniquely determine time indices. `NULL` for empty key, and `c()` for multiple variables. It works with tidy selector (e.g. `dplyr::starts_with()`).
`index`	A bare (or unquoted) variable to specify the time index variable.
`regular`	Regular time interval (`TRUE`) or irregular (`FALSE`). The interval is determined by the greatest common divisor of index column, if `TRUE`.
`.drop`	If `TRUE`, empty key groups are dropped.

Details

A tsibble is sorted by its key first and index.

Value

A tsibble object.

Index

An extensive range of indices are supported by tsibble: native time classes in R (such as `Date`, `POSIXct`, and `difftime`) and tsibble's new additions (such as [yearweek](#), [yearmonth](#), and [year-quarter](#)). Some commonly-used classes have built-in support too, including `ordered`, `hms::hms`, `zoo::yearmon`, `zoo::yearqtr`, and `nanotime`.

For a `tbl_ts` of regular interval, a choice of index representation has to be made. For example, a monthly data should correspond to time index created by [yearmonth](#) or `zoo::yearmon`, instead of `Date` or `POSIXct`. Because months in a year ensures the regularity, 12 months every year. However, if using `Date`, a month containing days ranges from 28 to 31 days, which results in irregular time space. This is also applicable to year-week and year-quarter.

Tsibble supports arbitrary index classes, as long as they can be ordered from past to future. To support a custom class, you need to define [index_valid()](#) for the class and calculate the interval through [interval_pull()](#).

Key

Key variable(s) together with the index uniquely identifies each record:

- Empty: an implicit variable. NULL resulting in a univariate time series.
- A single variable: For example, data(pedestrian) use the bare Sensor as the key.
- Multiple variables: For example, Declare key = c(Region, State, Purpose) for data(tourism). Key can be created in conjunction with tidy selectors like starts_with().

Interval

The [interval](#) function returns the interval associated with the tsibble.

- Regular: the value and its time unit including "nanosecond", "microsecond", "millisecond", "second", "minute", "hour", "day", "week", "month", "quarter", "year". An unrecognisable time interval is labelled as "unit".
- Irregular: as_tsibble(regular = FALSE) gives the irregular tsibble. It is marked with !.
- Unknown: Not determined (?), if it's an empty tsibble, or one entry for each key variable.

An interval is obtained based on the corresponding index representation:

- integerish numerics between 1582 and 2499: "year" (Y). Note the year of 1582 saw the beginning of the Gregorian Calendar switch.
- yearquarter/yearqtr: "quarter" (Q)
- yearmonth/yearmon: "month" (M)
- yearweek: "week" (W)
- Date: "day" (D)
- difftime: "week" (W), "day" (D), "hour" (h), "minute" (m), "second" (s)
- POSIXt/hms: "hour" (h), "minute" (m), "second" (s), "millisecond" (us), "microsecond" (ms)
- nanotime: "nanosecond" (ns)
- other numerics &ordered (ordered factor): "unit" When the interval cannot be obtained due to the mismatched index format, an error is issued.

The interval is invariant to subsetting, such as filter(), slice(), and [.tbl_ts. But if the result is an empty tsibble, the interval is always unknown. When joining a tsibble with other data sources and aggregating to different time scales, the interval gets re-calculated.

See Also

[build_tsibble](#)

Examples

```
# create a tsibble w/o a key
tsibble(
  date = as.Date("2017-01-01") + 0:9,
  value = rnorm(10)
)

# create a tsibble with a single variable for key
tsibble(
  qtr = rep(yearquarter("2010 Q1") + 0:9, 3),
```

```
  group = rep(c("x", "y", "z"), each = 10),
  value = rnorm(30),
  key = group
)

# create a tsibble with multiple variables for key
tsibble(
  mth = rep(yearmonth("2010 Jan") + 0:8, each = 3),
  xyz = rep(c("x", "y", "z"), each = 9),
  abc = rep(letters[1:3], times = 9),
  value = rnorm(27),
  key = c(xyz, abc)
)

# create a tsibble containing "key" and "index" as column names
tsibble(!!!list(
  index = rep(yearquarter("2010 Q1") + 0:9, 3),
  key = rep(c("x", "y", "z"), each = 10),
  value = rnorm(30)),
  key = key, index = index
)
```

---

tsibble-tidyverse   *Tidyverse methods for tsibble*

Description

Current dplyr verbs that tsibble has support for:

- dplyr::filter(), dplyr::slice(), dplyr::arrange(),
- dplyr::select(), dplyr::transmute(), dplyr::mutate(), dplyr::relocate(), dplyr::summarise(), dplyr::group_by(),
- dplyr::left_join(), dplyr::right_join(), dplyr::full_join(), dplyr::inner_join(), dplyr::semi_join(), dplyr::anti_join(), dplyr::nest_join()
- dplyr::bind_rows(), dplyr::bind_cols()

Current tidyr verbs that tsibble has support for:

- tidyr::pivot_longer(), tidyr::pivot_wider(), tidyr::gather(), tidyr::spread(),
- tidyr::nest(), tidyr::fill(), tidyr::drop_na()

Column-wise verbs

- The index variable cannot be dropped for a tsibble object.
- When any key variable is modified, a check on the validity of the resulting tsibble will be performed internally.
- Use as_tibble() to convert tsibble to a general data frame.

Row-wise verbs

A warning is likely to be issued, if observations are not arranged in past-to-future order.

Join verbs

Joining with other data sources triggers the check on the validity of the resulting tsibble.

Examples

```
library(dplyr, warn.conflicts = FALSE)
# `summarise()` a tsibble always aggregates over time
# Sum over sensors
pedestrian %>%
  index_by() %>%
  summarise(Total = sum(Count))
# shortcut
pedestrian %>%
  summarise(Total = sum(Count))
# Back to tibble
pedestrian %>%
  as_tibble() %>%
  summarise(Total = sum(Count))

library(tidyr)
stocks <- tsibble(
  time = as.Date("2009-01-01") + 0:9,
  X = rnorm(10, 0, 1),
  Y = rnorm(10, 0, 2),
  Z = rnorm(10, 0, 4)
)
(stocksm <- stocks %>%
  pivot_longer(-time, names_to = "stock", values_to = "price"))
stocksm %>%
  pivot_wider(names_from = stock, values_from = price)
```

---

update_tsibble   *Update key and index for a tsibble*

Description

Update key and index for a tsibble

Usage

```
update_tsibble(
  x,
  key,
  index,
  regular = is_regular(x),
  validate = TRUE,
  .drop = key_drop_default(x)
)
```

Arguments

x	A tsibble.
key	Unquoted variable(s) that uniquely determine time indices. NULL for empty key, and c() for multiple variables. It works with tidy selector (e.g. dplyr::starts_with()).
index	A bare (or unquoted) variable to specify the time index variable.
regular	Regular time interval (TRUE) or irregular (FALSE). The interval is determined by the greatest common divisor of index column, if TRUE.
validate	TRUE suggests to verify that each key or each combination of key variables leads to unique time indices (i.e. a valid tsibble). If you are sure that it's a valid input, specify FALSE to skip the checks.
.drop	If TRUE, empty key groups are dropped.

Details

Unspecified arguments will inherit the attributes from x.

Examples

```
# update index
library(dplyr)
pedestrian %>%
  group_by_key() %>%
  mutate(Hour_Since = Date_Time - min(Date_Time)) %>%
  update_tsibble(index = Hour_Since)

# update key: drop the variable "State" from the key
tourism %>%
  update_tsibble(key = c(Purpose, Region))
```

---

yearmonth	*Represent year-month*

Description

Stable

Create or coerce using yearmonth().

Usage

```
yearmonth(x)

is_yearmonth(x)
```

Arguments

x	Other object.

Value

year-month (yearmonth) objects.

Display

Use format() to display yearweek, yearmonth, and yearquarter objects in required formats. Please see strptime() details for supported conversion specifications.

See Also

Other index functions: yearquarter(), yearweek()

Examples

```
# coerce POSIXct/Dates to yearmonth
x <- seq(as.Date("2016-01-01"), as.Date("2016-12-31"), by = "1 month")
yearmonth(x)

# parse characters
yearmonth(c("2018 Jan", "2018-01", "2018 January"))

# creat an empty yearmonth container
yearmonth()

# seq() and arithmetic
mth <- yearmonth("2017-11")
seq(mth, length.out = 10, by = 1) # by 1 month
mth + 0:9

# display formats
format(mth, format = "%y %m")

# units since 1970 Jan
as.double(yearmonth("1969 Jan") + 0:24)
```

---

yearquarter   *Represent year-quarter*

Description

Stable

Create or coerce using yearquarter().

Usage

```
yearquarter(x)

is_yearquarter(x)
```

Arguments

x   Other object.

Value

year-quarter (yearquarter) objects.

Display

Use `format()` to display yearweek, yearmonth, and yearquarter objects in required formats. Please see `strptime()` details for supported conversion specifications.

See Also

Other index functions: `yearmonth()`, `yearweek()`

Examples

```
# coerce POSIXct/Dates to yearquarter
x <- seq(as.Date("2016-01-01"), as.Date("2016-12-31"), by = "1 quarter")
yearquarter(x)

# parse characters
yearquarter(c("2018 Q1", "2018 Qtr1", "2018 Quarter 1"))

# creat an empty yearquarter container
yearquarter()

# seq() and arithmetic
qtr <- yearquarter("2017 Q1")
seq(qtr, length.out = 10, by = 1) # by 1 quarter
qtr + 0:9

# display formats
format(qtr, format = "%y Qtr%q")

# units since 1970 Q1
as.double(yearquarter("1969 Q1") + 0:8)
```

---

yearweek   *Represent year-week (ISO) starting on Monday*

Description

Stable

Create or coerce using yearweek().

Usage

```
yearweek(x)

is_yearweek(x)

is_53weeks(year)
```

Arguments

x	Other object.
year	A vector of years.

Value

year-week (yearweek) objects.

TRUE/FALSE if the year has 53 ISO weeks.

Display

Use `format()` to display yearweek, yearmonth, and yearquarter objects in required formats.
Please see `strptime()` details for supported conversion specifications.

See Also

Other index functions: `yearmonth()`, `yearquarter()`

Examples

```
# coerce POSIXct/Dates to yearweek
x <- seq(as.Date("2016-01-01"), as.Date("2016-12-31"), by = "1 week")
yearweek(x)

# parse characters
yearweek(c("2018 W01", "2018 Wk01", "2018 Week 1"))

# creat an empty yearweek container
yearweek()

# seq() and arithmetic
wk1 <- yearweek("2017 W50")
wk2 <- yearweek("2018 W12")
seq(from = wk1, to = wk2, by = 2)
wk1 + 0:9

# display formats
format(c(wk1, wk2), format = "%V/%Y")

# units since 1969-12-29
as.double(yearweek("1969 W41") + 0:24)
is_53weeks(2015:2016)
```

Index