

# Package ‘tsibble’

October 9, 2022

**Type** Package

**Title** Tidy Temporal Data Frames and Tools

**Version** 1.1.3

**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),  
generics,  
lifecycle,  
lubridate (>= 1.7.0),  
methods,  
rlang (>= 0.4.6),  
tibble (>= 3.0.0),  
tidyselect (>= 1.0.0),  
vctrs (>= 0.3.1)

**Suggests** covr,  
ggplot2 (>= 3.3.0),  
hms,  
knitr,  
nanotime,  
nycflights13 (>= 1.0.0),  
rmarkdown,  
scales (>= 1.1.0),  
spelling,  
testthat (>= 3.0.0),  
tidyr (>= 1.1.0),  
timeDate

**VignetteBuilder** knitr

**RdMacros** lifecycle

**ByteCompile** true  
**Config/testthat/edition** 3  
**Encoding** UTF-8  
**Language** en-GB  
**LazyData** true  
**Roxygen** list(markdown = TRUE)  
**RoxygenNote** 7.2.1

## R topics documented:

tsibble-package	3
as.ts.tbl_ts	5
as_tibble.tbl_ts	6
as_tsibble	6
build_tsibble	9
count_gaps	10
difference	11
fill_gaps	12
filter_index	13
group_by_key	15
guess_frequency	15
has_gaps	16
holiday_aus	17
index	17
index_by	18
index_valid	19
interval	20
interval_pull	21
is_duplicated	21
is_tsibble	22
key	23
key_data	23
measures	24
new_data	24
new_interval	25
new_tsibble	26
pedestrian	27
scan_gaps	27
time_in	28
tourism	29
tsibble	30
tsibble-scales	32
tsibble-tidyverse	33
update_tsibble	34
yearmonth	35
yearquarter	36
yearweek	37

## 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`)
- tsibble's new additions (such as `yearweek`, `yearmonth`, and `yearquarter`).
- other commonly-used classes: `ordered`, `hms::hms`, `lubridate::period`, and `nanotime::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`, 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)` uses `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: "quarter" (Q)

- yearmonth: "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)
- period: "year" (Y), "month" (M), "day" (D), "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]`. However, 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](#))

Other contributors:

- Tyler Smith [contributor]
- Wil Davis <william.davis@worthingtonindustries.com> [contributor]

### 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	<i>Coerce to a tibble or data frame</i>
------------------	---

---

**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	<i>Coerce to a tsibble object</i>
------------	-----------------------------------

---

**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	Variable(s) that uniquely determine time indices. NULL for empty key, and c() for multiple variables. It works with tidy selector (e.g. <code>dplyr::starts_with()</code> ).
index	A 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`)
- tsibble's new additions (such as `yearweek`, `yearmonth`, and `yearquarter`).
- other commonly-used classes: `ordered`, `hms::hms`, `lubridate::period`, and `nanotime::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`, 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)` uses `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: "quarter" (Q)
- yearmonth: "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)
- period: "year" (Y), "month" (M), "day" (D), "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]`. However, 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	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 variable to specify the time index variable.
index2	A candidate of index to update the index to a new one when <code>index_by</code> . 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 <code>new_interval()</code> 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"),
  .start = NULL,
  .end = NULL
)
```

### Arguments

.data	A tsibble.
.full	<ul style="list-style-type: none"> <li>• FALSE inserts NA for each keyed unit within its own period.</li> <li>• TRUE fills NA over the entire time span of the data (a.k.a. fully balanced panel).</li> <li>• <code>start()</code> pad NA to the same starting point (i.e. <code>min(&lt;index&gt;)</code>) across units.</li> <li>• <code>end()</code> pad NA to the same ending point (i.e. <code>max(&lt;index&gt;)</code>) across units.</li> </ul>
.name	Strings to name new columns.
.start, .end	Set custom starting/ending time that allows to expand the existing time spans.

**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, .start = NULL, .end = NULL)
```

**Arguments**

<code>.data</code>	A tsibble.
<code>...</code>	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. <ul style="list-style-type: none"> <li>empty: filled with default NA.</li> <li>filled by values or functions.</li> </ul>
<code>.full</code>	<ul style="list-style-type: none"> <li>FALSE inserts NA for each keyed unit within its own period.</li> <li>TRUE fills NA over the entire time span of the data (a.k.a. fully balanced panel).</li> <li><code>start()</code> pad NA to the same starting point (i.e. <code>min(&lt;index&gt;)</code>) across units.</li> <li><code>end()</code> pad NA to the same ending point (i.e. <code>max(&lt;index&gt;)</code>) across units.</li> </ul>
<code>.start, .end</code>	Set custom starting/ending time that allows to expand the existing time spans.

**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())
fill_gaps(harvest, .start = 2009, .end = 2016)
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")

```

**Description**

This shorthand respects time zones and encourages compact expressions.

**Usage**

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

**Arguments**

<code>.data</code>	A tibble.
<code>...</code>	Formulas that specify start and end periods (inclusive), or strings. <ul style="list-style-type: none"> <li>• <code>~ end</code> or <code>. ~ end</code>: from the very beginning to a specified ending period.</li> <li>• <code>start ~ end</code>: from specified beginning to ending periods.</li> <li>• <code>start ~ .</code>: 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 &amp; numeric.</li> </ul>
<code>.preserve</code>	Relevant when the <code>.data</code> input is grouped. If <code>.preserve = FALSE</code> (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	<i>Group by key variables</i>
--------------	-------------------------------

---

**Description****[Stable]****Usage**

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

**Arguments**

.data	A tibble object.
...	Ignored.
.drop	Drop groups formed by factor levels that don't appear in the data? The default is TRUE except when .data has been previously grouped with .drop = FALSE. See <a href="#">group_by_drop_default()</a> for details.

**Examples**

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

---

guess_frequency	<i>Guess a time frequency from other index objects</i>
-----------------	--

---

**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 tibble have implicit gaps in time?***Description**

Does a tibble have implicit gaps in time?

**Usage**

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

**Arguments**

.data	A tibble.
.full	<ul style="list-style-type: none"> <li>FALSE inserts NA for each keyed unit within its own period.</li> <li>TRUE fills NA over the entire time span of the data (a.k.a. fully balanced panel).</li> <li>start() pad NA to the same starting point (i.e. min(&lt;index&gt;)) across units.</li> <li>end() pad NA to the same ending point (i.e. max(&lt;index&gt;)) across units.</li> </ul>
.name	Strings to name new columns.
.start, .end	Set custom starting/ending time that allows to expand the existing time spans.

**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 <- tibble(
  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_au	<i>Australian national and state-based public holiday</i>
------------	---

---

**Description**

Australian national and state-based public holiday

**Usage**

```
holiday_au(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).

**Examples**

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

---

index	<i>Return index variable from a tsibble</i>
-------	---

---

**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	<i>Group by time index and collapse with summarise()</i>
----------	--

---

**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))

# Rounding 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 supported by tsibble.

**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	<i>Pull time interval from a vector</i>
---------------	---

---

**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 index-like class.

**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	<i>Test duplicated observations determined by key and index variables</i>
---------------	---

---

**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	Variable(s) that uniquely determine time indices. NULL for empty key, and c() for multiple variables. It works with tidy selector (e.g. <code>dplyr::starts_with()</code> ).
index	A 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	<i>If the object is a tsibble</i>
------------	-----------------------------------

---

**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	<i>Return key variables</i>
-----	-----------------------------

---

**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)
```

---

key_data	<i>Key metadata</i>
----------	---------------------

---

**Description**

Key metadata

**Usage**

```
key_data(.data)
```

```
key_rows(.data)
```

```
key_size(x)
```

```
n_keys(x)
```

**Arguments**

.data, x            A tsibble

**See Also**

[dplyr::group\\_data](#)

**Examples**

```
key_data(pedestrian)
```

---

measures	<i>Return measured variables</i>
----------	----------------------------------

---

**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	<i>New tsibble data and append new observations to a tsibble</i>
----------	--

---

**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 <ul style="list-style-type: none"> <li>• <math>n &gt; 0</math>, future observations</li> <li>• <math>n &lt; 0</math>, past observations</li> </ul>
...	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	<i>Interval constructor for a tsibble</i>
--------------	---

---

**Description****[Stable]**

- new\_interval() creates an interval object.
- gcd\_interval() computes the greatest common divisor for the difference of numerics.
- is\_regular\_interval() checks if the interval is regular.

**Usage**

```

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

is_regular_interval(x)

gcd_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)
gcd_interval(c(1, 3, 5, 6))
```

---

new_tsibble	<i>Create a subclass of a tsibble</i>
-------------	---------------------------------------

---

**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 tibble 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 tibble for implicit missing observations*

---

### Description

Scan a tibble for implicit missing observations

### Usage

```
scan_gaps(.data, .full = FALSE, .start = NULL, .end = NULL)
```

**Arguments**

<code>.data</code>	A tibble.
<code>.full</code>	<ul style="list-style-type: none"> <li>• FALSE inserts NA for each keyed unit within its own period.</li> <li>• TRUE fills NA over the entire time span of the data (a.k.a. fully balanced panel).</li> <li>• <code>start()</code> pad NA to the same starting point (i.e. <code>min(&lt;index&gt;)</code>) across units.</li> <li>• <code>end()</code> pad NA to the same ending point (i.e. <code>max(&lt;index&gt;)</code>) across units.</li> </ul>
<code>.start, .end</code>	Set custom starting/ending time that allows to expand the existing time spans.

**See Also**

Other implicit gaps handling: [count\\_gaps\(\)](#), [fill\\_gaps\(\)](#), [has\\_gaps\(\)](#)

**Examples**

```
scan_gaps(pedestrian)
```

---

<code>time_in</code>	<i>If time falls in the ranges using compact expressions</i>
----------------------	--

---

**Description**

This function respects time zone and encourages compact expressions.

**Usage**

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

**Arguments**

<code>x</code>	A vector of time index, such as classes <code>POSIXct</code> , <code>Date</code> , <code>yearweek</code> , <code>yearmonth</code> , <code>yearquarter</code> , <code>hms/difftime</code> , and <code>numeric</code> .
<code>...</code>	Formulas that specify start and end periods (inclusive), or strings. <ul style="list-style-type: none"> <li>• <code>~ end</code> or <code>. ~ end</code>: from the very beginning to a specified ending period.</li> <li>• <code>start ~ end</code>: from specified beginning to ending periods.</li> <li>• <code>start ~ .:</code> from a specified beginning to the very end of the data. Supported index type: <code>POSIXct</code> (to seconds), <code>Date</code>, <code>yearweek</code>, <code>yearmonth/yearmon</code>, <code>yearquarter/yearqtr</code>, <code>hms/difftime</code> &amp; <code>numeric</code>.</li> </ul>

**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)
lg1 <- time_in(x, ~"2015-02", "2015-08" ~ "2015-09", "2015-12" ~ "2016-02")
lg1[1:10]
# more specific
lg2 <- time_in(x, "2015-03-23 10" ~ "2015-10-31 12")
lg2[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	Variable(s) that uniquely determine time indices. NULL for empty key, and <code>c()</code> for multiple variables. It works with tidy selector (e.g. <code>dplyr::starts_with()</code> ).
index	A 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`)
- tsibble's new additions (such as `yearweek`, `yearmonth`, and `yearquarter`).
- other commonly-used classes: `ordered`, `hms::hms`, `lubridate::period`, and `nanotime::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`, 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)` uses `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: "quarter" (Q)
- yearmonth: "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)
- period: "year" (Y), "month" (M), "day" (D), "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]`. However, 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-scales

*tsibble scales for ggplot2*


---

## Description

Defines ggplot2 scales for tsibble custom index: [yearweek](#), [yearmonth](#), and [yearquarter](#).

## Usage

```

scale_x_yearquarter(...)

scale_y_yearquarter(...)

scale_x_yearmonth(...)

scale_y_yearmonth(...)

scale_x_yearweek(...)

scale_y_yearweek(...)

```

## Arguments

... Arguments passed to [ggplot2::scale\\_x\\_date\(\)](#).

## Value

A ggproto object inheriting from Scale



## 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)
)
(stocks_m <- stocks %>%
  pivot_longer(-time, names_to = "stock", values_to = "price"))
stocks_m %>%
  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	Variable(s) that uniquely determine time indices. NULL for empty key, and c() for multiple variables. It works with tidy selector (e.g. <code>dplyr::starts_with()</code> ).
index	A 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	<i>Represent year-month</i>
-----------	-----------------------------

---

**Description****[Stable]**Create or coerce using `yearmonth()`.**Usage**

```
yearmonth(x, ...)

make_yearmonth(year = 1970L, month = 1L)

## S3 method for class 'character'
yearmonth(x, format = NULL, ...)

is_yearmonth(x)
```

**Arguments**

<code>x</code>	Other object.
<code>...</code>	Further arguments to methods.
<code>year, month</code>	A vector of numerics give years and months.
<code>format</code>	A vector of strings to specify additional formats of <code>x</code> (e.g. <code>%Y%m</code> ), if a warning or an error occurs.

**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**

[`scale\_x\_yearmonth`](#) and others for `ggplot2` scales  
 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"))

# 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)

make_yearmonth(year = 2021, month = 10:11)
make_yearmonth(year = 2020:2021, month = 10:11)
```

---

yearquarter

*Represent year-quarter*


---

**Description****[Stable]**

Create or coerce using yearquarter().

**Usage**

```
yearquarter(x, fiscal_start = 1)

make_yearquarter(year = 1970L, quarter = 1L, fiscal_start = 1)

is_yearquarter(x)

fiscal_year(x)
```

**Arguments**

x                    Other object.

fiscal\_start        numeric indicating the starting month of a fiscal year.

year, quarter        A vector of numerics give years and quarters.

**Value**

year-quarter (yearquarter) objects.

**Display**

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

**See Also**

[scale\\_x\\_yearquarter](#) and others for ggplot2 scales

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)
yearquarter(x, fiscal_start = 6)

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

# 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")

make_yearquarter(year = 2021, quarter = 2:3)
make_yearquarter(year = 2020:2021, quarter = 2:3)

# `fiscal_year()` helps to extract fiscal year
y <- yearquarter(as.Date("2020-06-01"), fiscal_start = 6)
fiscal_year(y)
lubridate::year(y) # calendar years
```

---

yearweek

*Represent year-week based on the ISO 8601 standard (with flexible start day)*

---

**Description**

**[Stable]**

Create or coerce using `yearweek()`.

**Usage**

```
yearweek(x, week_start = getOption("lubridate.week.start", 1))

make_yearweek(
  year = 1970L,
  week = 1L,
  week_start = getOption("lubridate.week.start", 1)
```

```
)
is_yearweek(x)
is_53weeks(year, week_start = getOption("lubridate.week.start", 1))
```

### Arguments

x	Other object.
week_start	An integer between 1 (Monday) and 7 (Sunday) to specify the day on which week starts following ISO conventions. Default to 1 (Monday). Use options(lubridate.week.start = 7) to set this parameter globally.
year, week	A vector of numerics give years and weeks.

### 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

[scale\\_x\\_yearweek](#) and others for ggplot2 scales  
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)
yearweek(x, week_start = 7)

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

# 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")

make_yearweek(year = 2021, week = 10:11)
make_yearweek(year = 2020:2021, week = 10:11)

is_53weeks(2015:2016)
is_53weeks(1969)
is_53weeks(1969, week_start = 7)
```

# Index

- \* **datasets**
  - pedestrian, 27
  - tourism, 29
- \* **implicit gaps handling**
  - count\_gaps, 10
  - fill\_gaps, 12
  - has\_gaps, 16
  - scan\_gaps, 27
- \* **index functions**
  - yearmonth, 35
  - yearquarter, 36
  - yearweek, 37

append\_case (new\_data), 24  
append\_row (new\_data), 24  
are\_duplicated (is\_duplicated), 21  
as.Date, 18  
as.ts.tbl\_ts, 5  
as\_tibble.tbl\_ts, 6  
as\_tsibble, 6, 20  
  
build\_tsibble, 9, 31  
  
count\_gaps, 10, 12, 16, 28  
  
difference, 11  
dplyr::anti\_join(), 33  
dplyr::arrange(), 33  
dplyr::bind\_cols(), 33  
dplyr::bind\_rows(), 33  
dplyr::filter(), 33  
dplyr::full\_join(), 33  
dplyr::group\_by(), 33  
dplyr::group\_data, 23  
dplyr::inner\_join(), 33  
dplyr::lag, 12  
dplyr::lead, 12  
dplyr::left\_join(), 33  
dplyr::mutate(), 33  
dplyr::nest\_join(), 33  
dplyr::relocate(), 33  
dplyr::right\_join(), 33  
dplyr::select(), 33  
dplyr::semi\_join(), 33  
  
dplyr::slice(), 33  
dplyr::starts\_with(), 7, 9, 22, 30, 34  
dplyr::summarise(), 33  
dplyr::transmute(), 33  
duplicates (is\_duplicated), 21  
  
fill\_gaps, 11, 12, 16, 28  
filter\_index, 13, 29  
fiscal\_year (yearquarter), 36  
  
gcd\_interval (new\_interval), 25  
ggplot2::scale\_x\_date(), 32  
group\_by\_drop\_default(), 15  
group\_by\_key, 15  
guess\_frequency, 15  
  
has\_gaps, 11, 12, 16, 28  
holiday\_aus, 17  
  
index, 17  
index2 (index), 17  
index2\_var (index), 17  
index\_by, 10, 18  
index\_valid, 19  
index\_valid(), 3, 7, 30  
index\_var (index), 17  
interval, 3, 8, 20, 31  
interval\_pull, 20, 21  
interval\_pull(), 3, 7, 30  
is\_53weeks (yearweek), 37  
is\_duplicated, 21  
is\_grouped\_ts (is\_tsibble), 22  
is\_ordered (interval), 20  
is\_regular (interval), 20  
is\_regular\_interval (new\_interval), 25  
is\_tsibble, 22  
is\_yearmonth (yearmonth), 35  
is\_yearquarter (yearquarter), 36  
is\_yearweek (yearweek), 37  
  
key, 23  
key\_data, 23  
key\_rows (key\_data), 23  
key\_size (key\_data), 23  
key\_vars (key), 23

- lubridate::as\_date, 18
- lubridate::ceiling\_date, 18
- lubridate::day(), 18
- lubridate::floor\_date, 18
- lubridate::hour(), 18
- lubridate::round\_date, 18
- lubridate::year, 18
  
- make\_yearmonth (yearmonth), 35
- make\_yearquarter (yearquarter), 36
- make\_yearweek (yearweek), 37
- measured\_vars (measures), 24
- measures, 24
  
- n\_keys (key\_data), 23
- new\_data, 24
- new\_interval, 25
- new\_interval(), 10
- new\_tsibble, 26
  
- pedestrian, 27
  
- scale\_alpha\_yearmonth (tsibble-scales), 32
- scale\_alpha\_yearquarter (tsibble-scales), 32
- scale\_alpha\_yearweek (tsibble-scales), 32
- scale\_color\_yearmonth (tsibble-scales), 32
- scale\_color\_yearquarter (tsibble-scales), 32
- scale\_color\_yearweek (tsibble-scales), 32
- scale\_colour\_yearmonth (tsibble-scales), 32
- scale\_colour\_yearquarter (tsibble-scales), 32
- scale\_colour\_yearweek (tsibble-scales), 32
- scale\_fill\_yearmonth (tsibble-scales), 32
- scale\_fill\_yearquarter (tsibble-scales), 32
- scale\_fill\_yearweek (tsibble-scales), 32
- scale\_size\_yearmonth (tsibble-scales), 32
- scale\_size\_yearquarter (tsibble-scales), 32
- scale\_size\_yearweek (tsibble-scales), 32
- scale\_x\_yearmonth, 35
- scale\_x\_yearmonth (tsibble-scales), 32
- scale\_x\_yearquarter, 37
- scale\_x\_yearquarter (tsibble-scales), 32
- scale\_x\_yearweek, 38
- scale\_x\_yearweek (tsibble-scales), 32
- scale\_y\_yearmonth (tsibble-scales), 32
- scale\_y\_yearquarter (tsibble-scales), 32
- scale\_y\_yearweek (tsibble-scales), 32
- scan\_gaps, 11, 12, 16, 27
- strptime(), 35, 37, 38
  
- tibble::tibble-package, 4
- tidyr::drop\_na(), 33
- tidyr::fill, 12
- tidyr::fill(), 33
- tidyr::gather(), 33
- tidyr::nest(), 33
- tidyr::pivot\_longer(), 33
- tidyr::pivot\_wider(), 33
- tidyr::replace\_na, 12
- tidyr::spread(), 33
- time\_in, 14, 28
- tourism, 29
- tsibble, 8, 30
- tsibble-package, 3
- tsibble-scales, 32
- tsibble-tidyverse, 33
  
- update\_tsibble, 34
  
- yearmonth, 3, 7, 18, 30, 32, 35, 37, 38
- yearquarter, 3, 7, 18, 30, 32, 35, 36, 38
- yearweek, 3, 7, 18, 30, 32, 35, 37, 37