Package 'mixtime'

May 19, 2026

Title Mixed Temporal Vectors and Operations

Version 0.1.0

Description Flexible time classes for time series analysis and forecasting with mixed temporal granularities. Supports linear and cyclical time representations in discrete and continuous forms, with timezone support, across multiple calendar systems including Gregorian and ISO week date calendars. Time points are stored numerically relative to a chronon; an atomic time granule defined by time units of a calendar. Calendrical arithmetic enables conversion between time granules (e.g. days to months) and calendar systems. Multi-unit arithmetic allows for temporal analysis with other granules of common calendars (e.g. fortnights are 2-week units). Time vectors of different granularities (e.g. monthly and quarterly) can be combined in a single vector, making 'mixtime' ideal for data that changes observation frequency over time or requires temporal reconciliation across scales. The package is extensible, allowing users to define custom calendars that build upon civil and astronomical time systems.

License MIT + file LICENSE

URL https://pkg.mitchelloharawild.com/mixtime/, https://github.com/mitchelloharawild/mixtime

BugReports https://github.com/mitchelloharawild/mixtime/issues

Encoding UTF-8

Language en-GB

Depends R (>= 3.0.2)

Imports lifecycle, vctrs, rlang, cli, S7, vecvec (> 0.2.1), tzdb, methods

Suggests stats, tsibble, testthat, pillar, knitr, rmarkdown

LinkingTo cpp11 (>= 0.5.2), tzdb (>= 0.5.0)

RdMacros lifecycle

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Mitchell O'Hara-Wild [aut, cre] (ORCID:
<https://orcid.org/0000-0001-6729-7695>)

Maintainer Mitchell O'Hara-Wild <mail@mitchelloharawild.com>

Repository CRAN

Date/Publication 2026-05-19 07:30:14 UTC

Contents

`as_mixtime` *Convert a time class into a mixtime*

Description

Coerces a time object (e.g. `Date`, `POSIXct`, `yearmonth`) to a `mixtime` vector using `vctrs::vec_cast()`. The chronon and cycle are inferred from `x` via `time_chronon()` and `time_cycle()`.

Usage

```
as_mixtime(x, ...)
```

Arguments

`x`	A time value to convert to a `mixtime`. Any time class with a defined `time_chronon()` method can be converted (e.g. `Date`, `POSIXct`, `yearmonth`, etc.).
`...`	Additional arguments passed to the underlying `vec_cast()` method.

Value

A `mixtime` object corresponding to `x`.

See Also

`mixtime()` for constructing a `mixtime` directly from data, `is_mixtime()` for testing if an object is a `mixtime`.

Examples

```
as_mixtime(Sys.Date())
as_mixtime(Sys.time())
```

`calendar_gregorian` *Gregorian time unit classes*

Description

Time unit constructors for the Gregorian calendar system. These units can be used with `linear_time()` to create custom time representations.

Usage

```
cal_gregorian
```

Format

A civil-based calendar containing Gregorian time units.

Details

The following time units are available in the Gregorian calendar (`cal_gregorian$`).

- `year()`: Year unit
- `quarter()`: Quarter (3-month period) unit
- `month()`: Month unit
- `day()`: Day unit
- `hour()`: Hour unit
- `minute()`: Minute unit
- `second()`: Second unit
- `millisecond()`: Millisecond unit

These units form a hierarchy where conversions between adjacent units follow the Gregorian calendar rules. For units that don't have a fixed relationship (e.g., months to days), the conversion requires a time context.

Value

An S3 list of class `c("cal_gregorian", "mt_calendar")` containing the named time unit classes of the Gregorian calendar. Each unit is accessible via `$` notation and calling it with a step size produces a time granule (e.g., 1 month granule as `cal_gregorian$month(1L)`).

See Also

`linear_time()` for creating linear time points.

Examples

```
# Create a custom time representation using Gregorian units
linear_time(
  Sys.time(),
  chronon = hour(1L)
)
```

`calendar_isoweek` *ISO 8601 time unit classes*

Description

Time unit constructors for the ISO 8601 calendar system. These units can be used with `linear_time()` to create custom time representations.

Usage

```
cal_isoweek
```

Format

A civil-based calendar containing ISO 8601 time units.

Details

The following time units are available in the ISO week date calendar:

- `year()`: ISO year unit (years start on the week containing the first Thursday)
- `week()`: Week unit (7-day periods)
- `day()`: Day unit
- `hour()`: Hour unit
- `minute()`: Minute unit
- `second()`: Second unit
- `millisecond()`: Millisecond unit

ISO 8601 weeks always start on Monday and the first week of a year is the week containing the first Thursday of that year. This means that some days in early January may belong to the last week of the previous ISO year, and some days in late December may belong to the first week of the next ISO year.

Value

An S3 list of class `c("cal_isoweek", "mt_calendar")` containing the named time unit classes of the ISO 8601 week calendar. Each unit is accessible via `$` notation and calling it with a step size produces a time granule (e.g., 1 week granule as `cal_isoweek$week(1L)`).

See Also

`linear_time()` for creating custom time representations, `yearweek()` for a pre-defined ISO 8601 year-week representation

---

`calendar_sym454` *Symmetry454 time unit classes*

Description

Time unit constructors for the Symmetry454 calendar system. These units can be used with `linear_time()` to create custom time representations.

Usage

```
cal_sym454
```

Format

A civil-based calendar containing Symmetry454 time units.

Details

The Symmetry454 calendar (Sym454) is a perennial solar calendar proposed by Dr. Irv Bromberg. It preserves the traditional 12-month structure and 7-day week, with months arranged in a symmetrical 4-5-4 week pattern per quarter. Every month starts on Monday and has a whole number of weeks, meaning no month ever contains a partial week.

The following time units are available in the Symmetry454 calendar (`cal_sym454$`).

- `year()`: Year unit
- `month()`: Month unit
- `week()`: Week unit
- `day()`: Day unit
- `hour()`: Hour unit
- `minute()`: Minute unit
- `second()`: Second unit
- `millisecond()`: Millisecond unit

Leap years:

Rather than intercalary days, Symmetry454 uses a **leap week** appended to December once every 5 or 6 years, making December a 5-week month in leap years. A year is a leap year if `(52 * year + 146) %% 293 < 52`.

This yields a mean year of 365 + 71/293 days (approx 365 days, 5 hours, 48 minutes, 56.5 seconds), intentionally slightly shorter than the mean northward equinoctial year.

Value

An S3 list of class `c("cal_sym454", "mt_calendar")` containing the named time unit classes of the Symmetry454 calendar. Each unit is accessible via `$` notation and calling it with a step size produces a time granule (e.g., 1 week granule as `cal_sym454$week(1L)`).

See Also

`linear_time()` for creating linear time points, and https://en.wikipedia.org/wiki/Symmetry454 for more calendar details.

Examples

```
# Create a custom time representation using Symmetry454 units
linear_time(
  Sys.time(),
  chronon = cal_sym454$week(1L)
)
```

`calendar_time_civil` *Civil time unit classes*

Description

Time unit constructors for the civil time system where the boundary of each day is at midnight on the 24 hour clock. This calendar is intended to be built on by other calendars (e.g. `[cal_time_civil]` and `[cal_isoweek]`) to add common time components. These units can be used with `linear_time()` to create custom time representations.

Usage

```
cal_time_civil
```

Details

The following time units are available (`cal_time_civil$`).

- `day()`: Day unit
- `hour()`: Hour unit
- `minute()`: Minute unit
- `second()`: Second unit
- `millisecond()`: Millisecond unit

Value

A time granule object for the civil time system.

See Also

cal_time_civil, cal_isoweek

Examples

```
# Create a custom time representation using civil time granules
hms <- new_cyclical_time_fn(
  chronon = second(1L),
  cycle = hour(1L)
)
```

`calendar_time_lunar` *Lunar time unit classes*

Description

Time unit constructors for the lunar time system where the boundary of each day is at sunrise, sunset, or noon. This calendar is intended to be built on by other calendars to add common time components.

Usage

```
cal_time_lunar
```

Format

A location-based calendar containing lunar time units.

Details

The following time units are available in the lunar calendar systems.

- `month()`: Synodic month unit
- `phase()`: Synodic phase unit

Value

An S3 list of class `c("cal_time_lunar", "mt_calendar")` containing the named time unit classes of the lunar calendar. Each unit is accessible via `$` notation and calling it with a step size and location produces a time granule (e.g., 1 synodic month granule as `cal_time_lunar$month(1L, lat = 0, lon = 0)`). Because lunar phases depend on the observer's position, each unit constructor requires `lat` and `lon` arguments.

See Also

cal_time_civil

Examples

```
# Find the time of a new moon in the Gregorian calendar
t <- linear_time(Sys.Date(), cal_time_lunar$month(1L, lat = -37.8136, lon = 144.9631))
datetime(t, tz = "Australia/Melbourne")
```

`calendar_time_solar` *Solar time unit classes*

Description

This time calendar contains solar time units, where the boundary of each day is at apparent solar midnight. Solar events define the `ampm` (midnight and noon) and `illumination` (dawn, sunrise, sunset, dusk) units.

Usage

```
cal_time_solar
```

Format

A location-based calendar containing solar time units.

Details

The following time units are available in the solar calendar systems.

- `day()`: Day unit
- `ampm()`: Half-day units (AM = before solar noon, PM = after solar noon)
- `hour()`: Hour units within the solar day
- `minute()`: Minute units within the solar hour
- `second()`: Second units within the solar minute
- `degree()`: Solar angle units within the day
- `arcminute()`: Arcminute units within the solar degree
- `arcsecond()`: Arcsecond units within the solar arcminute
- `illumination()`: Illumination phases (night, astronomical/nautical/civil dawn, day, civil/nautical/astronomical dusk)

AM/PM half-days:

The `ampm` unit divides each solar day into two halves between solar noon and solar midnight:

Half	Period	Description
AM	Solar midnight to noon	Morning half; before solar transit
PM	Solar noon to midnight	Afternoon half; after solar transit

Solar illumination phases:

Phases describe the illumination state of the sky and correspond to standard twilight definitions used in astronomy and navigation. Each phase is bounded by a pair of solar altitude thresholds:

Phase	Solar altitude range	Description
Night	< -18°	Sky fully dark; from last dusk to first dawn (spans noon)
Astronomical dawn	-18° to -12°	Astronomical twilight before sunrise; faint objects obscured
Nautical dawn	-12° to -6°	Nautical twilight before sunrise; horizon visible at sea
Civil dawn	-6° to -0.833°	Civil twilight before sunrise; sky brightening in the east
Day	> -0.833°	Sun above the horizon; spans solar noon
Civil dusk	-0.833° to -6°	Civil twilight after sunset; sky fading in the west
Nautical dusk	-6° to -12°	Nautical twilight after sunset; horizon visible at sea
Astronomical dusk	-12° to -18°	Astronomical twilight after sunset; faint objects obscured

The -0.833° threshold for sunrise and sunset accounts for the mean angular radius of the solar disc (0.267°) plus the standard atmospheric refraction at the horizon (0.566°). Noon and midnight are derived from the equation of time rather than a fixed altitude. Locations that experience polar day or polar night (civil days where sunrise does not occur) are not currently supported, it is recommended to use an alternative reference location.

Value

An S3 list of class `c("cal_time_solar", "mt_calendar")` containing the named time unit classes of the solar calendar. Each unit is accessible via `$` notation and calling it with a step size and location produces a time granule (e.g., 1 solar day granule as `cal_time_solar$day(1L, lat = 0, lon = 0)`). Because solar day boundaries depend on the observer's position, each unit constructor requires `lat` and `lon` arguments.

See Also

`cal_time_civil`

Examples

```
# Find the current solar time in Melbourne
datetime(Sys.time(), calendar = cal_time_solar, lat = -37.8136, lon = 144.9631)
```

`chronon_cardinality` *Cardinality between time granules*

Description

This S7 generic function defines the calendrical relationships between two chronons, and is one of the building block for defining calendars in mixtime. It calculates how many `x` chronons fit into the `y` chronon. Some chronon sizes are context-dependent (such as the number of days in a month), and so an optional time point defined in terms of `y` chronons can be provided with `at`.

Usage

```
chronon_cardinality(x, y, ...)
```

Arguments

`x`	The finer time granule (e.g. `cal_gregorian$month(1L)`)
`y`	The coarser time granule (e.g. `cal_gregorian$year(1L)`)
`...`	Additional arguments for methods.

Details

The methods are dispatched based on the shortest path along defined methods. This allows for defining only the direct relationships between adjacent time units, and relying on graph traversal to find how to convert between more distant units. For example the number of seconds in an hour can be calculated from the number of seconds in a minute and then number of minutes in an hour.

If a method is defined for converting between time units of different calendar systems (e.g., Gregorian calendar days to Chinese calendar days), then that method can be used to convert times at any granularity between the two systems.

Value

Numeric describing how many `x` time granules fit into `y` at time `at`.

Examples

```
# There are 12 months in a year
with(cal_gregorian, chronon_cardinality(month(1L), year(1L)))

# There are 7 days in a week
with(cal_isoweek, chronon_cardinality(day(1L), week(1L)))

# There are 3600 seconds in an hour
with(cal_gregorian, chronon_cardinality(second(1L), hour(1L)))

# There are 18 "2 months" in 3 years
with(cal_gregorian, chronon_cardinality(month(2L), year(3L)))

# There are 365 days in 2025 (a common year)
chronon_cardinality(
  cal_gregorian$day(1L), cal_gregorian$year(1L),
  at = year(as.Date("2025-01-01"))
)
```

```
# There are 366 days in 2024 (a leap year)
chronon_cardinality(
  cal_gregorian$day(1L), cal_gregorian$year(1L),
  at = mixtime::year(as.Date("2024-01-01"))
)

# There are 29 days in February 2024 (a leap year)
chronon_cardinality(
  cal_gregorian$day(1L), cal_gregorian$month(1L),
  at = yearmonth(as.Date("2024-02-01"))
)
```

`chronon_common` *Find the common chronon of a time object*

Description

This utility function takes a set of chronons and identifies a common chronon of the finest granularity that can represent all input chronons without loss of information. This is useful for operations that require a shared time granule, such as combining or comparing different time measured at different precisions.

The result is obtained by finding the greatest lower bound (GLB) of the input chronons using the ordered relationships defined by `chronon_cardinality()` methods. The GLB represents the finest chronon that can represent all input chronons without loss of information.

Usage

```
chronon_common(x, ...)

chronon_common.mixtime(x, .ptype = NULL, ...)
```

Arguments

`x`	A time object (typically a `mixtime`).
`...`	Additional arguments for methods.
`.ptype`	If NULL, the default, the output returns the common chronon across all chronons of `x`. Alternatively, a prototype chronon can be supplied to `.ptype` to demand a specific chronon is used. If the supplied `.ptype` cannot represent all input chronons without loss of information, an error is raised.

Value

A time granule object representing the common chronon.

Examples

```
# The common chronon between a year-month and a day is a day
chronon_common(c(yearmonth(Sys.Date()), date(Sys.Date())))

# The common chronon between a Gregorian month and an ISO week is a day
chronon_common(c(yearmonth(Sys.Date()), yearweek(Sys.Date())))

# The common chronon between a ISO week and an hour is an hour
chronon_common(c(yearweek(Sys.Date()), linear_time(Sys.time(), hour(1L))))
```

`chronon_divmod` *Convert between chronons of different time granules*

Description

This function converts between chronons measured in different time granules. It is used internally for converting between different continuous time types, and is particularly useful for efficiently converting between irregular time granules. The default method uses `chronon_cardinality()` to cast between time granules, which is efficient for regular time granules.

Usage

```
chronon_divmod(from, to, ...)
```

Arguments

`from`	The time granule that `x` is measured in (e.g., `day(1L)`).
`to`	The time granule to convert `x` into (e.g., `week(1L)`).
`...`	Additional arguments for methods.

Value

An list of two elements:

- `div`: integer vector of chronons measured in the `to` time granule.
- `mod`: integer vector of the remainder (in `from` time granule) after converting to the `to` time granule.

Examples

```
# Convert day 16 after epoch (1970-01-01) into weeks since epoch (and remainder days)
with(cal_isoweek, chronon_divmod(day(1L), week(1L), 16L))

# Convert week 4 after epoch (1970-W1) into days since epoch
with(cal_isoweek, chronon_divmod(week(1L), day(1L), 4L))
```

chronon_epoch *Epoch offset for chronons*

Description

Returns the epoch offset for a given chronon (time unit). The epoch defines the starting point of the chronon's linear numbering, used when converting from internal representations to common displays (e.g. applying an epoch of 1970-01-01).

Usage

```
chronon_epoch(x, ...)
```

Arguments

x	A chronon (time unit) object.
...	Additional arguments for methods.

Value

A numeric value representing the epoch for the chronon.

Examples

```
# The epoch for year linear time displays is 1970
chronon_epoch(cal_gregorian$year(1L))
```

chronon_format_attr *Default formatting strings for chronon attributes*

Description

Provides suffixes for default formatting strings for a given chronon (time granule). This provides useful information such as timezones or locations in the string.

Usage

```
chronon_format_attr(x, ...)
```

Arguments

x	A chronon (time granule) object.
...	Additional arguments for methods.

Value

A character string containing the default format suffix for the chronon.

chronon_format_linear *Default formatting strings for chronons*

Description

Provides default linear time formatting strings for a given chronon (finest time granule). The format strings use placeholders like {lin(year(1L))}, {cyc(month(1L), year(1L)} and {cyc(day(1L), month(1L)}, which are evaluated in the context of the data's time_calendar().

Usage

```
chronon_format_linear(x, cal = time_calendar(x), ...)

chronon_format_cyclical(x, y, ...)
```

Arguments

x	A time granule for the chronon.
cal	The calendar of the chronon, used to disambiguate suitable format strings for time units that are shared across calendars (e.g. cal_gregorian$day and cal_isoweek$day).
...	Additional arguments for methods.
y	A time granule for the cycle

Value

A character string containing the default format template for the chronon.

Examples

```
chronon_format_linear(cal_gregorian$year(1L))
chronon_format_linear(cal_gregorian$month(1L))
chronon_format_linear(cal_gregorian$day(1L))
chronon_format_linear(cal_isoweek$day(1L))

chronon_format_cyclical(cal_gregorian$month(1L), cal_gregorian$year(1L))
chronon_format_cyclical(cal_gregorian$day(1L), cal_gregorian$month(1L))
chronon_format_cyclical(cal_isoweek$day(1L), cal_isoweek$week(1L))
chronon_format_cyclical(cal_isoweek$week(1L), cal_isoweek$year(1L))
```

circsum *Compute circular rolling sums*

Description

Calculates rolling sums of length k for all contiguous subsequences around a circular vector. Returns sums for each valid k-element window that wraps around the vector as if arranged in a circle.

Usage

```
circsum(x, size, step = size)
```

Arguments

x	A numeric vector to compute circular sums over.
size	Integer; the window size (number of consecutive elements to sum).
step	Integer; the step size (the increment in starting index for each sum).

Value

A numeric vector containing the sum of each contiguous subsequence around the circle. The length of the resulting vector is the number of combinations until the pattern between x and step repeats

Examples

```
# Simple circular sum with window of 2
circsum(c(1, 2, 3, 4), 2)
# Returns: 3 7 (1+2, 3+4)

# Window of 3 elements
circsum(c(1, 2, 3, 4, 5), 3)
# Returns: 6 10 9 8 12 (1+2+3, 4+5+1, 2+3+4, 5+1+2, 3+4+5)
```

class_mixtime *Base S7 class for mixtime vector objects*

Description

class_mixtime is the base S7 class for all mixtime vector objects, inheriting from vecvec::class_vecvec. While not intended to be used directly, this S7 class is suitable to use when defining S7 methods for mixtime vectors. S3 methods can be defined using the mixtime::mixtime class.

Usage

```
class_mixtime(x = list(), i = seq_len(sum(lengths(x))))
```

Arguments

x	A list of "mt_time" vectors, see new_time() for details.
i	A vector of integers specifying the location of each element in x as if they were combined in order. The values in i must be between 1 and the total number of elements across all vectors in x, and can contain duplicates. If not provided, it defaults to a sequence from 1 to the total number of elements across all vectors in x.

Value

When used as a class definition (e.g., in S7::method(generic, class_mixtime)), an S7 class object representing the mixtime class, inheriting from vecvec::class_vecvec. When called as a constructor (class_mixtime(list(...))), a mixtime vector of S7 class mixtime (also inheriting the S3 class "mixtime"), containing the supplied list of time vectors as a vecvec::class_vecvec structure. End users should prefer mixtime() or new_mixtime() for construction.

See Also

mixtime() for creating mixtime vectors, and new_mixtime() for the low-level constructor function of this S7 class.

---

cyclical_labels *Friendly labels for cyclical relationships*

Description

This S7 generic function provides the labels for cyclical relationships between time granules. These functions should return locale specific labels.

Usage

```
cyclical_labels(granule, cycle, ...)
```

Arguments

granule	A time granule object representing the granule (e.g., month(1L))
cycle	A time granule object representing the cycle (e.g., year(1L))
...	Additional arguments for methods.

Value

Character vector of labels for the time point within the cycle.

Examples

```
# Labels for months in a year
with(cal_gregorian, cyclical_labels(month(1L), year(1L), 1:12))

# Labels for days in a week
with(cal_isoweek, cyclical_labels(day(1L), week(1L), 1:7))

# Labels for weeks in a year, defaulted from time_unit_abbr()
with(cal_isoweek, cyclical_labels(week(1L), year(1L), 1:52))
```

`cyclical_time` *Cyclical time points*

Description

`cyclical_time()` creates a vector of cyclical time points representing positions within repeating cycles. This function is useful for creating custom cyclical time representations that aren't covered by the convenience functions like `day_of_week()` or `month_of_year()`.

Usage

```
cyclical_time(
  data,
  chronon = time_chronon(data),
  cycle = time_cycle(data),
  discrete = TRUE,
  calendar = time_calendar(data)
)
```

Arguments

`data`	Input data to convert to cyclical time. Can be: • Numeric values (interpreted as chronons, 1-indexed) • Character strings (parsed as dates/times) • Date or POSIXct objects • Other time objects
`chronon`	A time granule representing the chronon (finest indivisible time granule), evaluated in the context of `calendar`. Use unquoted expressions like `day(1L)` or `month(1L)`. Chronons from a specific calendar can also be used (e.g. `cal_isoweek$day(1L)`).
`cycle`	A time granule representing the cycle (coarser time granule that defines the period), evaluated in the context of `calendar`. Use unquoted expressions like `week(1L)` or `year(1L)`.
`discrete`	Logical. If `TRUE` (default), returns integer positions within the cycle (discrete time model). If `FALSE`, returns fractional positions allowing representation of fractional time chronons (continuous time model).
`calendar`	Calendar system used to evaluate `chronon` and `cycle`. Defaults to `time_calendar(data)` for existing time objects. Common options include cal_gregorian and cal_isoweek.

Value

A `mixtime` time vector containing an `mt_cyclical` vector.

See Also

- `new_cyclical_time_fn()` for creating reusable cyclical time functions
- `day_of_week()`, `day_of_month()`, `day_of_year()` for common cyclical representations
- `month_of_year()`, `week_of_year()` for other cyclical time helpers
- cal_gregorian, cal_isoweek for calendar systems

Examples

```
# Day of week (1-7, Monday = 1)
cyclical_time(
  Sys.Date(),
  chronon = day(1L),
  cycle = week(1L),
  calendar = cal_isoweek
)

# Month of year (1-12)
cyclical_time(
  Sys.Date(),
  chronon = month(1L),
  cycle = year(1L)
)

# Continuous time (discrete = FALSE) for fractional month of year
cyclical_time(
  Sys.Date(),
  chronon = month(1L),
  cycle = year(1L),
  discrete = FALSE
)

# Day of month with Gregorian calendar
cyclical_time(
  Sys.Date(),
  chronon = day(1L),
  cycle = month(1L),
  calendar = cal_gregorian
)

# Hours, minutes, and seconds
cyclical_time(
  Sys.time(),
  chronon = second(1L),
  cycle = day(1L)
)
```

cyclical_time_helpers *Cyclical time helpers*

Description

Helper functions for creating cyclical time representations. These functions create time objects that repeat within a larger time cycle, useful for identifying seasonal patterns or positions within a calendar period.

Usage

```
month_of_year(data, discrete = TRUE, calendar = time_calendar(data), ...)

day_of_year(data, discrete = TRUE, calendar = time_calendar(data), ...)

day_of_month(data, discrete = TRUE, calendar = time_calendar(data), ...)

time_of_day(data, discrete = TRUE, calendar = time_calendar(data), ...)

day_of_week(data, discrete = TRUE, calendar = time_calendar(data), ...)

week_of_year(data, discrete = TRUE, calendar = time_calendar(data), ...)
```

Arguments

data	Another object to be coerced into the specified cyclical time.
discrete	If TRUE, the position within the cycle that data falls into is returned as an integer. If FALSE, a fractional position is returned (analagous to time using a continuous time model).
calendar	A calendar object specifying the calendar system to use.
...	Additional arguments for cyclical_time(), such as tz for timezones.

Value

A mixtime time vector containing an mt_cyclical vector with chronon and cycle matching the function used.

Cyclical time representations

- day_of_week(): Represents the day position within a week (1-7) using the ISO 8601 standard where weeks start on Monday.
- day_of_month(): Represents the day position within a month (1-28, 1-29, 1-30, or 1-31 depending on the month). The chronon is one day, cycling within a month.
- day_of_year(): Represents the day position within a year (1-365 or 1-366 for leap years). The chronon is one day, cycling within a year.

- `week_of_year()`: Represents the week position within a year (1-52 or 1-53) using the ISO 8601 week numbering system.
- `month_of_year()`: Represents the month position within a year (1-12). The chronon is one month, cycling within a year.

Custom cyclical time representations

You can create custom cyclical time representations using `cyclical_time()` with any of the supported time units (see calendar_gregorian and calendar_isoweek).

For example, to create a representation for day of the month:

```
day_of_month <- new_cyclical_time_fn(
  chronon = day(1L), cycle = month(1L),
  default_calendar = cal_gregorian
)
```

See Also

`cyclical_time()` for creating cyclical time vectors, `new_cyclical_time_fn()` for creating cyclical time helper functions

Examples

```
month_of_year(Sys.Date())
day_of_year(Sys.Date())
day_of_week(Sys.Date())
day_of_week(as.Date("2025-12-15") + 0:6)
```

`duration` *Duration vectors*

Description

`duration()` creates a vector of durations with a specified chronon. Durations represent a fixed span of time measured in a given time granule (e.g., 3 months, 5 days), without reference to a specific point in time.

Usage

```
duration(data, chronon = time_chronon(data), calendar = time_calendar(data))
```

Arguments

data	A time vector of duration magnitudes.
chronon	A time granule expression representing the chronon, evaluated in the context of calendar. Use unquoted expressions like month(1L) or day(1L). Chronons from a specific calendar can also be used (e.g. cal_gregorian$month(1L)). Defaults to the time chronon of the input data (time_chronon(data)).
calendar	Calendar system used to evaluate chronon. Defaults to time_calendar(data) for existing time objects. Common options include cal_gregorian and cal_isoweek.

Value

A mixtime vector containing an mt_duration vector.

See Also

- new_duration_fn() for creating reusable duration functions
- cal_gregorian, cal_isoweek for calendar systems

Examples

```
# A duration of 3 months
duration(3L, cal_gregorian$month(1L))

# A vector of durations in days
duration(1:7, cal_gregorian$day(1L))
```

---

duration_helpers *Duration helper functions*

Description

Convenience functions for creating duration vectors of common time units. Each function wraps new_duration_fn() for its respective chronon.

Usage

```
years(data, calendar = time_calendar(data), ...)

quarters(data, calendar = time_calendar(data), ...)

months(data, calendar = time_calendar(data), ...)

weeks(data, calendar = time_calendar(data), ...)

days(data, calendar = time_calendar(data), ...)
```

```
hours(data, calendar = time_calendar(data), ...)

minutes(data, calendar = time_calendar(data), ...)

seconds(data, calendar = time_calendar(data), ...)

milliseconds(data, calendar = time_calendar(data), ...)
```

Arguments

`data`	A time vector of duration magnitudes.
`calendar`	Calendar system used to evaluate `chronon`. Defaults to `time_calendar(data)` for existing time objects. Common options include cal_gregorian and cal_isoweek.
`...`	Additional arguments passed to the chronon (e.g. `tz` for timezones).

Value

A `mixtime` vector containing an `mt_duration` vector.

See Also

- `new_duration_fn()` for creating custom duration functions
- `duration()` for creating duration vectors directly
- cal_gregorian, cal_isoweek for calendar systems

Examples

```
years(3L)
quarters(2L)
months(6L)
weeks(4L)
days(7L)
hours(12L)
minutes(30L)
seconds(45L)
milliseconds(500L)
```

`is_mixtime` *Check if an object is a mixtime*

Description

Tests whether `x` inherits from the `mixtime` class.

Usage

```
is_mixtime(x)
```

Arguments

x	An object to test.

Value

A scalar logical: TRUE if x is a mixtime vector, FALSE otherwise.

See Also

as_mixtime() to coerce objects to mixtime, mixtime() to construct a mixtime.

Examples

```
is_mixtime(Sys.Date())
is_mixtime(mixtime(Sys.Date()))
```

---

is_time *Check the time type of values*

Description

Test whether elements of a mixtime vector are linear, cyclical, or durations.

Usage

```
is_time_linear(x, ...)

is_time_cyclical(x, ...)

is_time_duration(x, ...)
```

Arguments

x	A time object (typically a mixtime vector).
...	Additional arguments for methods.

Details

These helpers return a logical vector the same length as x identifying the type of time represented by each element.

Value

A logical vector the same length as x.

Examples

```
t <- c(yearmonth(0), month_of_year(0), months(0L))
is_time_linear(t)
is_time_cyclical(t)
is_time_duration(t)
```

---

linear_labels *Friendly labels for linear relationships*

Description

This S7 generic function provides the labels for linear (non-repeating) positions of a time granule. These functions should return locale specific labels.

Usage

```
linear_labels(granule, ...)
```

Arguments

granule	A time granule object representing the granule (e.g., year(1L))
...	Additional arguments for methods.

Value

Character vector of labels for the time point.

Examples

```
# Labels for years on a linear axis
with(cal_gregorian, linear_labels(year(1L), 2020:2025))
```

---

linear_time *Linear time points*

Description

linear_time() creates a vector of linear time points with a specified chronon (smallest time granule). This function is useful for creating custom time representations that aren't covered by the convenience functions like yearmonth() or yearweek().

Usage

```
linear_time(
  data,
  chronon = time_chronon(data),
  discrete = TRUE,
  calendar = time_calendar(data)
)
```

Arguments

`data`	Input data to convert to linear time. Can be: • Numeric values (interpreted as chronons since Unix epoch) • Character strings (parsed as dates/times) • Date or POSIXct objects • Other time objects
`chronon`	A time granule expression representing the chronon (smallest indivisible time granule), evaluated in the context of `calendar`. Use unquoted expressions like `month(1L)` or `hour(1L)`. Chronons from a specific calendar can also be used (e.g. `cal_isoweek$week(1L)`). Defaults to the time chronon of the input `data` (`time_chronon(data)`).
`discrete`	Logical. If `TRUE` (default), returns integer chronons since Unix epoch (discrete time model). If `FALSE`, returns fractional chronons allowing representation of fractional time granules (continuous time model).
`calendar`	Calendar system used to evaluate `chronon` and `granules`. Defaults to `time_calendar(data)` for existing time objects. Common options include cal_gregorian and cal_isoweek.

Value

A `mixtime` time vector containing an `mt_linear` vector.

See Also

- `new_linear_time_fn()` for creating reusable linear time functions
- `yearmonth()`, `yearquarter()`, `year()` for Gregorian time representations
- `yearweek()` for ISO 8601 week-based time
- cal_gregorian, cal_isoweek for calendar systems

Examples

```
# Hourly time
linear_time(
  Sys.time(),
  chronon = hour(1L)
)

# Monthly time
linear_time(
```

```
  Sys.Date(),
  chronon = month(1L)
)

# Discrete vs continuous time
linear_time(Sys.time(), chronon = day(1L), discrete = TRUE)
linear_time(Sys.time(), chronon = day(1L), discrete = FALSE)

# ISO week calendar with week-day structure
linear_time(
  Sys.Date(),
  chronon = day(1L),
  calendar = cal_isoweek
)
```

`linear_time_helpers` *Linear time helper functions*

Description

Convenience functions for creating common linear time representations. These functions work with different calendar systems and adapt based on the input data's calendar.

Usage

```
year(data, discrete = TRUE, calendar = time_calendar(data), ...)

yearquarter(data, discrete = TRUE, calendar = time_calendar(data), ...)

yearmonth(data, discrete = TRUE, calendar = time_calendar(data), ...)

yearweek(data, discrete = TRUE, calendar = time_calendar(data), ...)

date(data, discrete = TRUE, calendar = time_calendar(data), ...)

datetime(data, discrete = TRUE, calendar = time_calendar(data), ...)
```

Arguments

`data`	A vector of time points (e.g. base::Date, base::POSIXt)
`discrete`	If `TRUE`, the number of chronons since Unix epoch that `data` falls into is returned as an integer. If `FALSE`, a fractional number of chronons is returned (analagous to time using a continuous time model).
`calendar`	A calendar used to evaluate the time units. Defaults to the calendar of the input data. Common options include cal_gregorian and cal_isoweek.
`...`	Additional arguments for `linear_time()`, such as `tz` for timezones.

Details

These functions create linear time representations with different chronons and granules:

- `year()`: Represents time in whole years. The chronon is one year.
- `yearquarter()`: Represents time in quarters, grouped by year. The chronon is one quarter, with years as the granule.
- `yearmonth()`: Represents time in months, grouped by year. The chronon is one month, with years as the granule.
- `yearweek()`: Represents time in weeks, grouped by year. The chronon is one week, with years as the granule. Defaults to ISO 8601 week calendar.

Value

A `mixtime` time vector containing an `mt_linear` vector with chronons matching the function used.

Calendar flexibility

These functions adapt to the calendar system of the input data. For example:

- `year("2025-12-29")` returns a Gregorian year
- `year(yearweek("2025-12-29"))` returns an ISO week-based year

You can also explicitly specify a calendar using the `calendar` argument:

```
year(yearweek("2025-12-29"), calendar = cal_isoweek)
```

Custom linear time representations

For more complex time structures, use `linear_time()` or `new_linear_time_fn()` to create custom representations with any combination of chronons and granules.

See Also

- `linear_time()` for creating custom linear time representations
- `new_linear_time_fn()` for creating reusable linear time functions
- cal_gregorian, cal_isoweek for calendar systems

Examples

```
# Gregorian year
year(Sys.Date())
year(Sys.Date(), discrete = FALSE)

# ISO week-based year
year(yearweek(Sys.Date()))

# Year-quarter
yearquarter(Sys.Date())
yearquarter(Sys.Date(), discrete = FALSE)
```

```
# Year-month
yearmonth(Sys.Date())
yearmonth(Sys.Date(), discrete = FALSE)

# Year-week (ISO 8601)
yearweek(Sys.Date())
yearweek(0:52)
```

`mixtime` *Create a mixtime vector*

Description

A mixtime is a vector which describes a point in time. It uses a calendar definition to translate a vector of numbers into a point in time.

Usage

```
mixtime(
  data,
  chronon = time_chronon(data),
  cycle = time_cycle(data),
  discrete = TRUE
)
```

Arguments

`data`	A vector of time values. This can be a character vector (e.g. "2024-01-01"), a numeric vector (e.g. seconds since epoch), or a time class (e.g. Date, POSIXct, yearmonth, etc.).
`chronon`	A time granule object representing the smallest indivisible time granule (chronon) for the mixtime. This is used to interpret the numeric values in `data` and to define the time resolution of the mixtime. If not provided, it will be inferred from `data`.
`cycle`	An optional time granule object representing the cycle for cyclical time. This is used to define the repeating cycle for cyclical time representations (e.g. day-of-week, month-of-year). If not provided, the mixtime will be treated as linear time.
`discrete`	A logical indicating whether the time values should be treated as discrete (integer) or continuous (fractional). This affects how numeric values are interpreted and how time arithmetic is performed. The default is `TRUE` (discrete).

Value

A `mixtime` object representing the time values in `data` according to the specified `chronon` and `cycle`.

Examples

```
# Create a mixtime for today
mixtime(Sys.Date())

# Create a mixtime for the current date and time
mixtime(Sys.time())

# Convert time from tsibble classes to mixtime
mixtime(tsibble::yearmonth("2024 Jan"))

# Create a mixtime for the time of day (cyclical time)
mixtime(Sys.time(), cycle = cal_gregorian$day(1L))

# Specify a timezone for the chronon
mixtime(Sys.time(), chronon = cal_gregorian$second(1L, tz = Sys.timezone()))
mixtime(Sys.time(), chronon = cal_gregorian$second(1L, tz = "Pacific/Honolulu"))
mixtime(Sys.time(), chronon = cal_gregorian$second(1L, tz = "Australia/Melbourne"))

# Dates (and all granularities) can have timezones
mixtime(Sys.time(), chronon = cal_gregorian$day(1L, tz = Sys.timezone()))
mixtime(Sys.time(), chronon = cal_gregorian$day(1L, tz = "Pacific/Honolulu"))
mixtime(Sys.time(), chronon = cal_gregorian$day(1L, tz = "Australia/Melbourne"))

# Continuous time tracks progress within the chronon
mixtime(Sys.time(), chronon = cal_gregorian$day(1L, tz = Sys.timezone()), discrete = FALSE)

# Mixtime can combine different granularities and timezones in a vector
now <- Sys.time()
c(
  # Datetime (second chronon) in UTC
  mixtime(now),
  # Date (minute chronon) in local timezone
  mixtime(now, chronon = cal_gregorian$minute(1L, tz = Sys.timezone())),
  # Month (month chronon) in UTC
  mixtime(now, chronon = cal_gregorian$month(1L))
)
```

---

mixtime_location *Extract locations from an object*

Description

Generic function to extract the location from objects that have location information.

Usage

```
loc_latitude(x, ...)

loc_longitude(x, ...)

loc_altitude(x, ...)
```

Arguments

x	An object with location information.
...	Additional arguments passed to methods.

Value

A numeric value representing the location (e.g., longitude, latitude, etc.).

Examples

```
t <- linear_time(
  1:3,
  cal_time_solar$day(1L, lat = -37.8136, lon = 144.9631)
)

loc_longitude(t)
loc_latitude(t)
loc_altitude(t)
```

---

mt_unit *Base S7 class for creating new time units*

Description

This class is the primative class for time units, and should be extended from when creating new time units. A new class is typically created with S7 using: `S7::new_class("tu_***", parent = mt_tz_unit)`

Usage

```
mt_unit(n = 1L)

mt_loc_unit(n = 1L, lat = naive_loc, lon = naive_loc, alt = naive(0))

mt_tz_unit(n = 1L, tz = naive_tz)
```

Arguments

n	The step size of time granule. For example, `n = 2L` is 2 time units - for `cal_isoweek$week(2L)` that would be 2 weeks (a fortnight).
lat	Numeric. Latitude in decimal degrees. Range: -90 to 90. Default: naive location (astronomical calculations require `lat`).
lon	Numeric. Longitude in decimal degrees. Range: -180 to 180. Default: naive location (astronomical calculations require `lon`).
alt	Numeric. Altitude in meters above sea level. Default: 0 (sea level).
tz	The timezone name for the unit (valid units can be found with `[tzdb::tzdb_names()]`)

Details

Time units are the building blocks of calendars in mixtime. Each unit represents a specific temporal component (e.g., day, month, year) and can be combined using `new_calendar()` to create a calendar system.

When creating custom calendars, define time unit classes that inherit from either `mt_unit` (for standard units) or `mt_tz_unit` (for timezone-aware units), then pass them as named arguments to `new_calendar()`. The calendar will use these names to create constructor functions accessible via `$` notation (e.g., `calendar$day(1L)`).

Value

A time granule object of class `mt_unit`

Calendar Algebra Methods

Time units enable calendar arithmetic through two key generic methods that should be implemented for custom time units:

- `chronon_cardinality(x, y, at)` - Returns the number of `x` granule that fit within one `y` granule This can be a fixed value (e.g., 7 days per week) or variable based on `at` (e.g., 28-31 days per month).
- `chronon_divmod(x, from, to)` - Converts time point `x` from granules of `from` to granules of `to`, returning a list with `div` (the quotient) and `mod`. This enables conversions between granules that have variable cardinality (e.g., the date 2020-03-23 to the month 2020-03). All conversions should be based on chronons since epoch (1970-01-01), in the UTC time zone.

These methods work together to enable mixtime to perform calendar-aware arithmetic, understanding that months have variable lengths and handling timezone-aware conversions.

See Also

`new_calendar()` for creating calendars from time units

Examples

```
# Create a timezone-aware unit class

# Use these units to create a calendar
my_calendar <- new_calendar(
  day = S7::new_class("tu_my_day", parent = mt_unit),
  month = S7::new_class("tu_my_month", parent = mt_tz_unit),
  class = "my_calendar"
)

# Access unit constructors from the calendar
my_calendar$day(1L)
my_calendar$month(3L, tz = "America/New_York")
```

`new_calendar` *Create a new calendar*

Description

Define a new calendar as a collection of time units. Calendars are the foundation for representing dates and times in terms of human-readable components like years, months, days, hours, minutes, and seconds. Each calendar is defined by specifying the time units it contains, which determine how time values can be interpreted and manipulated.

Usage

```
new_calendar(..., inherit = NULL, class = character())
```

Arguments

`...`	Named time unit class definitions. Each argument should be a time unit class (typically created with `S7::new_class()`) that inherits from `mt_unit` or `mt_tz_unit`. The names define the calendar's fields and are used to access unit constructors (e.g., `calendar$year()`).
`inherit`	Optional calendar to inherit time units from. Units defined in `...` will override inherited units with the same name.
`class`	Character vector of additional classes for the calendar object.

Details

Time units are typically S7 class definitions that inherit from `mt_unit` for standard units, `mt_tz_unit` for timezone-aware units (civil time), or `mt_loc_unit` for location-aware units (astronomical time). The calendar object provides a namespace for accessing these unit constructors and defines the relationships between them for calendar arithmetic.

Value

A calendar object of class `c(class, "mt_calendar")`, consisting of a named list containing the specified time unit classes.

See Also

`linear_time()`, `cyclical_time()`

Examples

```
# Create a simple calendar with year and month units
# (inheriting from civil time units for day, hour, minute, second, ...)
cal_simple <- new_calendar(
  year = new_time_unit("tu_year", parent = mt_tz_unit),
  month = new_time_unit("tu_month", parent = mt_tz_unit),
  inherit = cal_time_civil,
```

```
  class = "cal_simple"
)

# Create time granules from the calendar
cal_simple$year(1L)
cal_simple$month(1L)
```

`new_cyclical_time_fn` *Cyclical time function factory*

Description

`new_cyclical_time_fn()` creates a cyclical time function for a specified chronon and cycle. The cycle is the larger time granule that defines the time period over which the chronon loops (e.g., a week). The chronon is the smaller time granule that iterates within each cycle (e.g., a day). Combined, these two granules form a cyclical time relationship (e.g., day of the week).

Usage

```
new_cyclical_time_fn(chronon, cycle, default_calendar = cal_gregorian)
```

Arguments

`chronon`	A time granule object representing the chronon (e.g., `day(1L)`)
`cycle`	A time granule object representing the cycle (e.g., `week(1L)`)
`default_calendar`	A default calendar used to find the time units for conversion if they don't exist in the calendar of the input data (e.g., `cal_isoweek`)

Value

A function used to create cyclical time points with a specific chronon and cycle.

Examples

```
day_of_week <- new_cyclical_time_fn(day(1L), week(1L), default_calendar = cal_isoweek)
day_of_week(Sys.Date())

month_of_year <- new_cyclical_time_fn(month(1L), year(1L))
month_of_year(Sys.Date())
```

new_duration_fn *Duration function factory*

Description

new_duration_fn() creates a duration function for a specified chronon. A chronon is the smallest indivisible time unit (e.g., days, months) that defines what the numeric magnitudes in the resulting duration vector represent.

Usage

```
new_duration_fn(chronon, default_calendar = cal_gregorian)
```

Arguments

chronon	A bare call for a time unit object representing the chronon (e.g., month(1L), day(1L)).
default_calendar	A default calendar used to resolve the time units if they don't exist in the calendar of the input data (e.g., cal_gregorian).

Value

A function used to create duration vectors with a specific chronon. The returned function accepts:

data A numeric vector of duration magnitudes.

calendar A calendar system used to evaluate chronon. Defaults to time_calendar(data).

... Additional arguments passed to the chronon (e.g., tz for timezones).

See Also

- duration() for creating duration vectors directly
- cal_gregorian, cal_isoweek for calendar systems

Examples

```
# Create a months duration function
months <- new_duration_fn(month(1L), default_calendar = cal_gregorian)
months(1:6)

# Create a days duration function
days <- new_duration_fn(day(1L), default_calendar = cal_gregorian)
days(1:7)
```

`new_linear_time_fn` *Linear time function factory*

Description

`new_linear_time_fn()` creates a linear time function for a specified chronon. A chronon is the smallest indivisible time granule (e.g., days, hours).

Usage

```
new_linear_time_fn(chronon, default_calendar = cal_gregorian)
```

Arguments

`chronon`	A bare call for a time granule object representing the chronon (e.g., `day(1)`)
`default_calendar`	A default calendar used to find the time units for conversion if they don't exist in the calendar of the input data (e.g., `cal_isoweek` for week chronons to work with gregorian calendar inputs).

Value

A function used to create linear time points with a specific chronon.

Examples

```
# Linear time with 1 month granules as the chronon
ym <- new_linear_time_fn(month(1L))
ym(Sys.Date())

# Linear time with 1 day granules as the chronon
yd <- new_linear_time_fn(day(1L))
yd(Sys.Date())

# Linear time with 1 week granules as the chronon, using the ISO week calendar
yw <- new_linear_time_fn(week(1L), default_calendar = cal_isoweek)
yw(Sys.Date())

# Linear time with 1 hour granules as the chronon
ymd_h <- new_linear_time_fn(hour(1L))
ymd_h(Sys.time())
```

new_mixtime *Constructor for mixtime vectors*

Description

Creates a `mixtime` vector, which can contain time points of different granularities (e.g. monthly and quarterly) in a single vector via `vecvec`.

Usage

```
new_mixtime(x = new_time())
```

Arguments

`x`	A mixtime time vector (created with `new_time()`) to wrap in a mixtime class.

Value

A `mixtime` object, which allows mixed-type time vectors to coexist in a single vector.

new_time *Constructor for mixtime time vectors*

Description

Creates a `mixtime` time vector at a specific time point, with a specified chronon and optional cycle. The `chronon` defines the smallest indivisible time granule for the time vector, while the `cycle` allows for cyclical time representations (e.g. day-of-week, month-of-year).

Usage

```
new_time(x = integer(), chronon = mt_unit(1L), cycle = NULL, class = NULL)
```

Arguments

`x`	A numeric vector of time points, integers for discrete time or doubles for continuous time.
`chronon`	A time granule object representing the smallest indivisible time granule (chronon) for the time vector (e.g. `cal_gregorian$day(1L)`).
`cycle`	An optional time granule object representing the cycle for cyclical time (e.g. `cal_gregorian$week(1L)` for day-of-week). If not provided, the time vector will be treated as linear time.
`class`	An optional character vector of additional S3 classes to assign to the resulting time vector. This allows for further subclassing of `mt_time` for specific time types (e.g. linear, cyclical, durations, etc.).

Value

A `mt_time` vector representing the time points in `x` according to the specified `chronon` and `cycle`.

Examples

```
# Create a continuous mixtime time vector for today
new_time(
  as.double(Sys.Date()),
  chronon = cal_gregorian$day(1L, tz = Sys.timezone()),
  class = "mt_linear"
)

# Create a discrete mixtime time vector for the current date and time
new_time(
  as.integer(Sys.time()),
  chronon = cal_gregorian$second(1L, tz = Sys.timezone()),
  class = "mt_linear"
)

# Create a discrete mixtime time vector for the time of day (cyclical time)
new_time(
  as.integer(Sys.time()),
  chronon = cal_gregorian$second(1L, tz = Sys.timezone()),
  cycle = cal_gregorian$day(1L, tz = Sys.timezone()),
  class = "mt_cyclical"
)
```

---

new_time_unit *Create a new time unit class*

Description

Define a new S7 class representing a time unit for use in a mixtime calendar. Time units are the building blocks of calendars: each unit represents a specific temporal component (e.g., day, month, year) and carries a step size `n`. Units are combined via `new_calendar()` to form a complete calendar system.

Usage

```
new_time_unit(
  name,
  parent = mt_unit,
  package = topNamespaceName(parent.frame()),
  properties = list(),
  abstract = FALSE,
  constructor = NULL,
  validator = NULL
)
```

Arguments

name	A string naming the new S7 class (e.g., "tu_my_year"). By convention, time unit class names are prefixed with tu_.
parent	The parent S7 class. Should be one of mt_unit, mt_tz_unit, or mt_loc_unit, or a class that itself inherits from one of these. Defaults to mt_unit.
package	A string giving the package name that owns the class. Defaults to the name of the calling namespace, so typically does not need to be set explicitly.
properties	A named list of additional S7::new_property() definitions beyond those inherited from parent. Each entry becomes a slot on instances of the new class. Defaults to an empty list.
abstract	Logical. If TRUE the class cannot be instantiated directly; it serves only as a base for further subclassing.
constructor	A function to use as the class constructor, or NULL (default) to generate one automatically. The auto-generated constructor accepts ... (forwarded to the parent constructor) plus one argument per entry in properties, with defaults taken from each property's default field.
validator	A function of one argument (self) that returns NULL when the object is valid, or a character string describing the problem. See S7::new_class() for details.

Details

Choose the parent class based on the type of time the unit represents:

- mt_unit — abstract or calendar-only time (no timezone or location context; e.g., Gregorian year or ISO week).
- mt_tz_unit — civil time with a timezone (e.g., a clock hour that needs tz to resolve wall-clock ambiguity).
- mt_loc_unit — astronomical time with a geographic location (e.g., a solar day tied to observer longitude/latitude).

Value

An S7 class object, as returned by S7::new_class().

See Also

- mt_unit, mt_tz_unit, mt_loc_unit for the base classes to inherit from.
- new_calendar() for assembling time units into a calendar.
- S7::new_class() for the underlying S7 class constructor.

Examples

```
# An abstract unit with no properties
tu_my_seq <- new_time_unit("tu_my_seq", parent = mt_unit)

# A civil-time unit with an extra property
```

```
tu_my_quarter <- new_time_unit(
  "tu_my_quarter",
  parent = mt_tz_unit,
  properties = list(
    fiscal = S7::new_property(S7::class_logical, default = FALSE)
  )
)
```

`seq.mixtime::mixtime` *Generate sequences of mixtime values*

Description

Create regular sequences of time values. This method handles both linear time sequences (dates, date-times) and cyclical time sequences (day of week, month of year).

Usage

```
## S3 method for class '`mixtime::mixtime`'
seq(...)

## S3 method for class 'mt_time'
seq(
  from,
  to,
  by,
  length.out = NULL,
  along.with = NULL,
  on_invalid = c("nearest", "overflow"),
  ...
)
```

Arguments

`...`	Additional arguments passed to the underlying sequence method.
`from`	Starting value of the sequence.
`to`	End value of the sequence (if provided).
`by`	Increment of the sequence. Can be: • A numeric for the number of time chronons • A character string specifying the interval (e.g., "1 day", "2 weeks", "1 month", "1 year") • A time granule object created with time unit functions (e.g., `cal_gregorian$year(1L)`, `cal_gregorian$month(1L)`, `cal_gregorian$day(1L)`) • A time `duration()` object (e.g., `years(1L)`, `months(1L)`, `days(1L)`)
`length.out`	Desired length of the sequence (alternative to `to`).

`along.with` Take the length from this argument (alternative to `length.out`).

`on_invalid` How to handle time points that overflow the cycle when using a by argument with different time granule than the sequence chronon. Options are:

- `"nearest"` (default): Adjust overflowing time points to the nearest valid time point within the cycle
- `"overflow"`: Allow time points to overflow into the next cycle

This is relevant when the starting time point has an offset that doesn't exist in all cycles. For example, starting on day 31 with `by = "1 month"` will overflow in months with fewer than 31 days (e.g., February). With `"nearest"`, these will be adjusted to the last day of the month (e.g., Feb 28/29). With `"overflow"`, the extra days carry into the next month.

If not explicitly specified and overflow occurs, a warning is issued with the default `"nearest"` behavior applied.

Details

For linear time types (Date, POSIXct, yearmonth, etc.), sequences progress forward or backward in time. For cyclical time types (month_of_year, day_of_week, etc.), sequences wrap around cyclically.

Value

A mixtime vector containing the sequence.

Examples

```
# Linear time sequences with integer by
seq(yearmonth("2020-01-01"), yearmonth("2020-12-01"))
seq(yearquarter("2020-01-01"), length.out = 5, by = 3)

# Linear time sequences with string intervals
seq(date("2020-01-01"), date("2020-12-31"), by = "1 month")
seq(yearmonth("2020-01-01"), yearmonth("2025-01-01"), by = "1 year")
seq(date("2020-01-01"), length.out = 10, by = "2 weeks")

# Linear time sequences incrementing by time granules
seq(yearmonth("2020-01-01"), yearmonth("2020-12-01"), by = cal_gregorian$month(2L))
seq(date("2020-01-01"), length.out = 5, by = cal_gregorian$year(1L))
seq(date("2020-01-01"), date("2020-01-31"), by = cal_gregorian$day(7L))

# Handling invalid dates with on_invalid
seq(date("2020-01-31"), length.out = 3, by = "1 month")  # warns, uses nearest
seq(date("2020-01-31"), length.out = 3, by = "1 month", on_invalid = "nearest")
seq(date("2020-01-31"), length.out = 3, by = "1 month", on_invalid = "overflow")

# Cyclical time sequences
seq(month_of_year(0L), month_of_year(11L))
seq(day_of_week(0L), day_of_week(6L), by = 1)
```

time_calendar *Obtain the calendar of a time object*

Description

This S7 generic function extracts the calendar system from a time object. The calendar defines the collection of time units (years, months, days, etc.) used to interpret the time representation.

Usage

```
time_calendar(x, ...)
```

Arguments

x	A time object (e.g., base::Date, base::POSIXct, `linear_time()`, etc.)
...	Additional arguments for methods.

Value

A calendar object (e.g., `cal_gregorian`, `cal_isoweek`)

Examples

```
# The calendar of a Date object is the Gregorian calendar
time_calendar(Sys.Date())

# The calendar of a POSIXct object is also Gregorian
time_calendar(Sys.time())

# The calendar of a yearweek object is the ISO week calendar
time_calendar(yearweek(Sys.Date()))

# A mixed time object returns a list of calendars
time_calendar(c(yearmonth(Sys.Date()), Sys.Date()))
```

time_chronon *Obtain the chronon of a time object*

Description

This S7 generic function extracts the chronon (the smallest time granule) from a time object, such as continuous time or cyclical time representations.

Usage

```
time_chronon(x, ...)
```

Arguments

x	A time object (e.g., base::Date, base::POSIXct, `linear_time()`, etc.)
...	Additional arguments for methods.

Value

A time `duration()` vector representing the chronon of each value (e.g., `days(1L)`).

Examples

```
# The chronon of a Date object is 1 day
time_chronon(Sys.Date())

# The chronon of a POSIXct object is 1 second
time_chronon(Sys.time())

# The chronon of a continuous time year and month is 1 month
time_chronon(yearmonth(Sys.Date()))

# The common chronon of a mixed time object is the finest chronon
time_chronon(c(yearmonth(Sys.Date()), Sys.Date()))
```

`time_cycle` *Obtain the cycle of a time object*

Description

This S7 generic function extracts the cycle (the cyclical time granule) from a time object, such as cyclical time representations.

Usage

```
time_cycle(x, ...)
```

Arguments

x	A time object (e.g., base::Date, base::POSIXct, `linear_time()`, etc.)
...	Additional arguments for methods.

Value

A time `duration()` object representing the cycle of each value (e.g. `weeks(1L)`), or `NA` if the object has no cyclical component.

Examples

```
# Non-cyclical objects return NA
time_cycle(Sys.Date())

# The cycle of a cyclical time object
time_cycle(month_of_year(Sys.Date()))
```

`time_round` *Round, floor and ceiling transformations for time objects*

Description

A family of helpers to round date/time objects to a specified time granule such as second, minute, hour, or day. These functions preserve the input time class, as rounded by the attributes of the `granule`.

Usage

```
time_round(x, granule, ...)

time_ceiling(x, granule, ...)

time_floor(x, granule, ...)
```

Arguments

`x`	A date/time object to be rounded. Accepted types include Date, POSIXct, POSIXlt and other objects that inherit from POSIXt. The returned object will be of the same class as the input.
`granule`	A time granule (or object coercible to a time granule, e.g. "day").
`...`	Additional arguments passed to specific implementations.

Value

An object of the same class as x with its time components adjusted to the requested granule.

See Also

base::round, lubridate::round_date

Examples

```
# Round POSIXct to the nearest minute (preserving tz)
t <- as.POSIXct("2020-01-01 12:34:56", tz = "UTC")
time_round(t, granule = cal_gregorian$minute(1L))

# Floor to the nearest hour
time_floor(t, granule = cal_gregorian$hour(1L))

# Ceiling a Date (treated as midnight-of-day rounding)
d <- as.Date("2020-01-01")
time_ceiling(d, granule = cal_gregorian$month(1L))
```

---

time_unit_full *Time units as a string*

Description

These S7 generic functions provide the full and abbreviated names for time units. `time_unit_full()` is used in messages and durations (e.g., "2 months"). `time_unit_abbr()` is used for tsibble index interval displays (e.g., "1M").

Usage

```
time_unit_full(x, ...)

time_unit_abbr(x, ...)
```

Arguments

`x`	A time granule object (e.g., `cal_gregorian$month(1L)`)
`...`	Additional arguments for methods.

Value

A string representing the time unit

Examples

```
time_unit_full(cal_gregorian$year(1L))
time_unit_abbr(cal_gregorian$year(1L))
```

tz_abbreviation *Get timezone abbreviation*

Description

Returns the timezone abbreviation (e.g., "EST", "PDT") for a given datetime in its specified timezone.

Usage

```
tz_abbreviation(x, tz = tz_name(x))
```

Arguments

x	A POSIXct datetime object or something coercible to POSIXct. The timezone is extracted from this object.
tz	A character vector of timezones to abbreviate at time point x.

Value

A character vector of timezone abbreviations.

Examples

```
tz_abbreviation(Sys.time())
tz_abbreviation(as.POSIXct("2024-01-15 12:00:00", tz = "America/New_York"))
```

tz_name *Extract timezone from an object*

Description

Generic function to extract the timezone from objects that have timezone information.

Usage

```
tz_name(x, ...)
```

Arguments

x	An object with timezone information.
...	Additional arguments passed to methods.

Value

A character vector representing the timezone of each time point (e.g., "America/New_York", "UTC").

Examples

```
tz_name(Sys.time())
tz_name(as.POSIXct("2024-06-15 12:00:00", tz = "America/New_York"))
```

`tz_offset` *Get timezone offset*

Description

Returns the UTC offset for a given datetime in its specified timezone.

Usage

```
tz_offset(x, ...)
```

Arguments

`x`	A time class coercible to POSIXt with an associated time zone.
`...`	Additional arguments passed to methods.

Value

A numeric vector of offsets from UTC in the same chronon (e.g. seconds for POSIXt, days for dates, etc.)

Examples

```
tz_offset(as.POSIXct(Sys.time(), tz = Sys.timezone()))
tz_offset(as.POSIXct("2024-06-15 12:00:00", tz = "America/New_York"))
```

`tz_transitions` *Get timezone transitions*

Description

Returns all timezone transitions (e.g., daylight saving time changes) that occur between two datetimes. The timezone is taken from the start datetime.

Usage

```
tz_transitions(start, end)
```

Arguments

start	A POSIXct datetime object or something coercible to POSIXct, representing the start of the time range. The timezone is extracted from this object.
end	A POSIXct datetime object or something coercible to POSIXct, representing the end of the time range.

Value

A data frame containing information about timezone transitions in the specified range.

Examples

```
# Get all DST transitions in 2024 for New York
tz_transitions(
  as.POSIXct("2024-01-01", tz = "America/New_York"),
  as.POSIXct("2024-12-31", tz = "America/New_York")
)
```

Index