

Package ‘era’

July 30, 2026

Type Package

Title Year-Based Time Scales

Version 0.6.0

Description Provides a consistent representation of year-based time scales as a numeric vector with an associated 'era'. There are built-in era definitions for many year numbering systems used in contemporary and historic calendars (e.g. Common Era, Islamic 'Hijri' years); year-based time scales used in archaeology, astronomy, geology, and other palaeosciences (e.g. Before Present, SI-prefixed 'annus'); and support for arbitrary user-defined eras. Years can be converted from any one era to another using a generalised transformation function. Methods are also provided for robust casting and coercion between years and other numeric types, type-stable arithmetic with years, and pretty-printing in tables.

Language en-GB

License MIT + file LICENSE

URL <https://era.joeroe.io>, <https://github.com/joeroe/era>

BugReports <https://github.com/joeroe/era/issues>

Encoding UTF-8

Depends R (>= 2.10)

Imports vctrs (>= 0.3.0), methods, rlang, pillar, zeallot

Suggests tibble, testthat, covr, knitr, rmarkdown, dplyr, spelling, purrr, tidyr, lubridate

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Joe Roe [aut, cre] (ORCID: <<https://orcid.org/0000-0002-1011-1244>>)

Maintainer Joe Roe <joeroe@archaeo.dev>

Repository CRAN

Date/Publication 2026-07-30 16:50:13 UTC

Contents

era	2
eras	3
era_parameters	4
era_year	5
era_year_parameters	6
is_era	6
is_era_year	7
is_yr	8
this_year	9
yr	10
yr_difference	10
yr_earlier_than	11
yr_era	13
yr_extremes	14
yr_sort	15
yr_transform	16
Index	18

era	<i>Create an era object</i>
-----	-----------------------------

Description

An era object defines the time scale associated with a vector of years (see [yr\(\)](#)). `era()` returns an era object, either by looking up label in the standard eras defined in [eras\(\)](#) or, if more than one argument is given, constructing a new definition with the specified parameters.

Usage

```
era(  
  label = character(),  
  epoch = NULL,  
  name = label,  
  unit = era_year("Gregorian"),  
  scale = 1,  
  direction = -1  
)
```

Arguments

label	Character. If only one argument is given to <code>era()</code> , the abbreviated label of a standard era defined in eras() . Otherwise, the label to give to the era constructed using the following arguments.
epoch	Numeric. Epoch year from which years are counted in Gregorian astronomical years (i.e. there is a "year zero").

name	Character. Full name of the era. Defaults to the value of label.
unit	An era_year() object describing the name of the year unit and its average length in solar days. Defaults to a Gregorian year (365.2425 days).
scale	Integer. Number of years represented by one unit, e.g. 1000 for ka. Default: 1.
direction	Are years counted backwards (-1) (the default) or forwards (1) from epoch?

Value

An object of class era.

See Also

Other era definition functions: [eras\(\)](#)

Other era helper functions: [era_parameters](#), [era_year\(\)](#), [era_year_parameters](#), [is_era\(\)](#), [is_era_year\(\)](#), [is_yr\(\)](#), [this_year\(\)](#)

Examples

```
era("cal BP")

era("T.A.", epoch = -9021, name = "Third Age", direction = 1)
```

eras	<i>Standard era definitions</i>
------	---------------------------------

Description

Definitions of common eras and time scales.

`eras()` lists all available era definitions. `eras(label)` looks up a specific era by its unique, abbreviated name (e.g. "cal BP").

Usage

```
eras(label = NA)
```

Arguments

label (Optional) Abbreviated names(s) of eras to look up.

Details

Looking up eras by label uses partial matching.

Value

A table of era definitions. This can be passed to [era\(\)](#) to construct an era object.

See Also

Other era definition functions: [era\(\)](#)

Examples

```
# List all available eras
print(eras(), n = Inf)

# Look up a specific era by label
eras("cal BP")

# With partial matching
eras("cal")
```

era_parameters	<i>Get parameters of an era</i>
----------------	---------------------------------

Description

Extracts a specific parameter from an era object.

Usage

```
era_label(x)

era_epoch(x)

era_name(x)

era_unit(x)

era_scale(x)

era_direction(x)
```

Arguments

x An era object.

Details

The available parameters are:

- **label** – unique, abbreviated label of the era, e.g. "cal BP"
- **epoch** – year of origin of the era, e.g. 1950 for Before Present
- **name** – full name of the era, e.g. "calendar years Before Present"
- **unit** – unit of years used, an [era_year\(\)](#) object
- **scale** – multiple of years used, e.g. 1000 for ka/kiloannum
- **direction** – whether years are counted "backwards" or "forwards" from the epoch #

Value

Value of the parameter.

See Also

Other era helper functions: [era\(\)](#), [era_year\(\)](#), [era_year_parameters](#), [is_era\(\)](#), [is_era_year\(\)](#), [is_yr\(\)](#), [this_year\(\)](#)

Examples

```
x <- era("cal BP")
era_name(x)
```

era_year	<i>Year units</i>
----------	-------------------

Description

era_year objects describe the unit used for a year as its length in days. This value is used in an era definition ([era\(\)](#)) to enable conversions between eras that use different units (with [yr_transform\(\)](#)).

Usage

```
era_year(label = character(), days = 365.2425)
```

Arguments

label	Character. Name of the year unit.
days	Numeric. Average length of the year in solar days. Defaults to a Gregorian year (365.2425 days).

Value

S3 vector of class era_year.

See Also

Other era helper functions: [era\(\)](#), [era_parameters](#), [era_year_parameters](#), [is_era\(\)](#), [is_era_year\(\)](#), [is_yr\(\)](#), [this_year\(\)](#)

Examples

```
era_year("Julian", 365.25)
```

era_year_parameters	<i>Get the parameters of an era_year object.</i>
---------------------	--

Description

Extracts a specific parameter from a year unit object constructed by [era_year\(\)](#).

Usage

```
era_year_label(x)
```

```
era_year_days(x)
```

Arguments

`x` An object of class `era_year`.

Value

Value of the parameter.

See Also

Other era helper functions: [era\(\)](#), [era_parameters](#), [era_year\(\)](#), [is_era\(\)](#), [is_era_year\(\)](#), [is_yr\(\)](#), [this_year\(\)](#)

Examples

```
julian <- era_year("Julian", 365.25)
era_year_label(julian)
era_year_days(julian)
```

is_era	<i>Validation functions for era objects</i>
--------	---

Description

Tests whether an object is an era definition (an era object). `is_era()` tests whether the object inherits from the S3 class `era_yr`. `is_valid_era()` performs additional checks to determine whether the object is well-formed (see details). `validate_era()` throws an informative error message for invalid yrs.

Usage

```
is_era(x)
```

```
validate_era(x)
```

```
is_valid_era(x)
```

Arguments

x Object to test.

Details

Valid era objects:

- Must have all parameters set and not NA
- Must have a character label parameter
- Must have a numeric epoch parameter
- Must have a character name parameter
- Must have a character unit parameter that is one of the defined units
- Must have a positive, integer scale parameter
- Must have a direction parameter that is -1 (backwards) or 1 (forwards)

Value

is_era() and is_valid_era() return TRUE or FALSE. validate_era() returns x invisibly, and is used for its side-effect of throwing an informative error for invalid objects.

See Also

Other era helper functions: [era\(\)](#), [era_parameters](#), [era_year\(\)](#), [era_year_parameters](#), [is_era_year\(\)](#), [is_yr\(\)](#), [this_year\(\)](#)

is_era_year

Validation functions for era_year objects

Description

Tests whether an object is of class era_year (constructed by [era_year\(\)](#)).

Usage

```
is_era_year(x)
```

Arguments

x Object to test.

Value

TRUE or FALSE.

See Also

Other era helper functions: [era\(\)](#), [era_parameters](#), [era_year\(\)](#), [era_year_parameters](#), [is_era\(\)](#), [is_yr\(\)](#), [this_year\(\)](#)

Examples

```
is_era_year(era_year("Julian", 365.25))
```

is_yr

Validation functions for yr objects

Description

Tests whether an object is a vector of years with an era (a yr object). `is_yr()` tests whether the object inherits from the S3 class `era_yr`. `is_valid_yr()` performs additional checks to determine whether the object is well-formed (see details). `validate_yr()` throws an informative error message for invalid yrs.

Usage

```
is_yr(x)
```

```
validate_yr(x)
```

```
is_valid_yr(x)
```

Arguments

`x` Object to test.

Details

Valid yr objects:

- Must contain numeric data (NAs are allowed)
- Must have the era attribute set and not NA
- Must not have more than one era
- Must have an era attribute that is a valid era object (see `validate_era()`)

Value

`is_yr()` and `is_valid_yr()` return TRUE or FALSE. `validate_yr()` returns `x` invisibly, and is used for its side-effect of throwing an informative error for invalid objects.

See Also

Other era helper functions: [era\(\)](#), [era_parameters](#), [era_year\(\)](#), [era_year_parameters](#), [is_era\(\)](#), [is_era_year\(\)](#), [this_year\(\)](#)

Examples

```
x <- yr(5000:5050, era("cal BP"))
is_yr(x)
is_valid_yr(x)
validate_yr(x)
```

this_year

Current year

Description

Returns the current year as a year vector, in the era system specified by era.

Usage

```
this_year(era = "CE")
```

Arguments

era An era object or label understood by [era\(\)](#). Defaults to the Common Era ([era\("CE"\)](#)).

Value

A yr vector with the current year.

See Also

Other era helper functions: [era\(\)](#), [era_parameters](#), [era_year\(\)](#), [era_year_parameters](#), [is_era\(\)](#), [is_era_year\(\)](#), [is_yr\(\)](#)

Examples

```
# This year in the Common Era
this_year()
# This year in the Holocene Epoch
this_year("HE")
```

yr	Create a vector of years with era
----	-----------------------------------

Description

A yr object represents years with an associated calendar era or time scale.

Usage

```
yr(x = numeric(), era = character())
```

Arguments

- | | |
|-----|--|
| x | A numeric vector of years. |
| era | The calendar era used by x. Either: <ul style="list-style-type: none"> • A string matching one of the standard era labels defined in eras() • An era object constructed with era() |

Value

A yr (era_yr) object.

See Also

Other years with era functions: [yr_era\(\)](#), [yr_transform\(\)](#)

Examples

```
# The R Age
yr(1993:2020, "CE")

# A bad movie
yr(10000, "BC")
```

yr_difference	Chronological difference between year vectors
---------------	---

Description

yr_difference(x, y) calculates how much later x is than y, taking into

Usage

```
yr_difference(x, y)
```

Arguments

x	A yr vector with era.
y	A yr vector or numeric value. If yr, and in a different era than x, it is transformed to the era of x. Recycled to the common length of x and y.

Details

This function performs chronological subtraction, not numeric subtraction. Both inputs are converted to a common chronological scale (accounting for era direction and epoch), and the difference is computed as $x - y$ on that scale.

For forward-counting eras (e.g. CE), this is equivalent to numeric subtraction. For backward-counting eras (e.g. BCE, BP), the era direction is accounted for, so `yr_difference(yr(1900, "BCE"), yr(2000, "BCE"))` returns 100 (1900 BCE is 100 years later than 2000 BCE), even though $1900 - 2000 = -100$.

Value

Numeric vector of signed differences. Positive values indicate x is chronologically later than y; negative values indicate x is earlier.

See Also

Other functions for chronological ordering and extremes: [yr_earlier_than\(\)](#), [yr_extremes](#), [yr_sort\(\)](#)

Examples

```
# Forward-counting era (CE):
yr_difference(yr(200, "CE"), yr(100, "CE")) # 100
yr_difference(yr(100, "CE"), yr(200, "CE")) # -100

# Backward-counting era (BCE):
yr_difference(yr(1900, "BCE"), yr(2000, "BCE")) # 100
yr_difference(yr(2000, "BCE"), yr(1900, "BCE")) # -100

# Cross-era (accounts for year zero):
yr_difference(yr(10, "CE"), yr(10, "BCE")) # 19

# Scaled eras (returns difference in era's native units):
yr_difference(yr(2, "ka"), yr(1, "ka")) # 1
```

yr_earlier_than

Chronological comparison of years

Description

Tests whether years are chronologically earlier or later than each other, accounting for era direction.

Usage

```
yr_earlier_than(x, y)
```

```
yr_later_than(x, y)
```

```
yr_not_earlier_than(x, y)
```

```
yr_not_later_than(x, y)
```

Arguments

`x, y` [yr](#) vectors with era. If in different eras, `y` is transformed to the era of `x`. Recycled to their common length.

Details

For forward-counting eras (e.g. CE), `yr_earlier_than(x, y)` is equivalent to $x < y$ and `yr_later_than(x, y)` is equivalent to $x > y$. For backward-counting eras (e.g. BP, BCE), the comparisons are reversed.

`yr_not_earlier_than()` and `yr_not_later_than()` are the negations of `yr_earlier_than()` and `yr_later_than()`, equivalent to $\geq$ and $\leq$ respectively for forward-counting eras.

These are implemented as functions rather than S3 methods for $<$, $>$, $\leq$, and $\geq$ to avoid surprises when numerical (i.e. not chronological) comparison is expected.

Value

A logical vector.

See Also

Other functions for chronological ordering and extremes: [yr_difference\(\)](#), [yr_extremes](#), [yr_sort\(\)](#)

Examples

```
# Forward-counting era:
x <- yr(c(100, 200, 300), "CE")
yr_earlier_than(x, yr(200, "CE"))
yr_later_than(x, yr(200, "CE"))
yr_not_earlier_than(x, yr(200, "CE"))
yr_not_later_than(x, yr(200, "CE"))

# Backward-counting era:
y <- yr(c(100, 200, 300), "BCE")
yr_earlier_than(y, yr(200, "BCE"))
yr_later_than(y, yr(200, "BCE"))
yr_not_earlier_than(y, yr(200, "BCE"))
yr_not_later_than(y, yr(200, "BCE"))
```

yr_era	<i>Get or set the era of a vector of years</i>
--------	--

Description

Functions for extracting or assigning the era of a vector of years. This function does not alter the underlying values of `x`. Use [yr_transform\(\)](#) to *convert* the values of a `yr` vector to a new era.

Usage

```
yr_era(x)

yr_set_era(x, era)

yr_era(x) <- value
```

Arguments

<code>x</code>	A vector of years.
<code>value, era</code>	An era object (see era()) to be assigned to <code>x</code> .

Value

`yr_era(x)` returns the existing era associated with `x`.

`yr_set_era(x, era)` and `yr_era(x) <- era` return `x` with the new era assigned. If `x` is not already a `yr` vector, it will attempt to coerce it into one.

See Also

Other years with era functions: [yr\(\)](#), [yr_transform\(\)](#)

Examples

```
x <- 5000:5050
yr_era(x) <- era("cal BP")
yr_era(x)
```

yr_extremes

*Chronological minima and maxima***Description**

Returns the chronologically earliest and/or latest value in a vector of years, i.e. era-aware version [min\(\)](#), [max\(\)](#), and [range\(\)](#).

Usage

```
yr_earliest(x, na.rm = FALSE)
```

```
yr_latest(x, na.rm = FALSE)
```

```
yr_range(x, na.rm = FALSE)
```

Arguments

x	A yr vector with era
na.rm	a logical indicating whether missing values should be removed

Details

These are implemented as prefixed functions rather than S3 [min\(\)](#), [max\(\)](#), and [range\(\)](#) methods for [yrs](#) to avoid surprises when numerical (i.e. not chronological) extremes are expected.

Value

For [yr_earliest\(\)](#) and [yr_latest\(\)](#), a [yr](#) vector of length 1 with the earliest or latest value.

For [yr_range\(\)](#), a [yr](#) vector of length 2 with the earliest and latest value (in that order).

If x contains NA values and na.rm = FALSE (the default), only NAs will be returned.

See Also

Other functions for chronological ordering and extremes: [yr_difference\(\)](#), [yr_earlier_than\(\)](#), [yr_sort\(\)](#)

Examples

```
# Forward-counting era:
x <- yr(c(200, 100, 300), "CE")
yr_earliest(x)
yr_latest(x)
yr_range(x)

# Backward-counting era:
y <- yr(c(200, 100, 300), "BCE")
yr_earliest(y)
```

```
yr_latest(y)
yr_range(x)
```

 yr_sort

Chronological ordering of year vectors

Description

Sorts a vector of years into earliest-to-latest or latest-to-earliest chronological order based on its era.

Usage

```
yr_sort(x, reverse = FALSE, ...)
```

Arguments

x	yr vector with era
reverse	Set FALSE (the default) for chronological order (earliest to latest) or TRUE for reverse chronological order (latest to earliest).
...	Other arguments passed to sort() ; in particular use <code>na.last</code> to control NA handling.

Details

This is implemented as a prefixed function rather than an S3 [sort\(\)](#) method for [yrs](#) to avoid surprises when numerical (i.e. not chronological) sorting is expected.

Value

Sorted [yr](#) vector

See Also

Other functions for chronological ordering and extremes: [yr_difference\(\)](#), [yr_earlier_than\(\)](#), [yr_extremes](#)

Examples

```
# Forward-counting era:
x <- yr(c(200, 100, 300), "CE")
yr_sort(x)
yr_sort(x, reverse = TRUE)

# Backward-counting era:
y <- yr(c(200, 100, 300), "BCE")
yr_sort(y)
yr_sort(y, reverse = TRUE)
```

 yr_transform

 Convert years from one era to another

Description

Transform a vector of years from one era to another.

Usage

```
yr_transform(x, era, ...)

## S3 method for class 'era_yr'
yr_transform(x, era = yr_era(x), precision = NA, ...)

## Default S3 method:
yr_transform(x, era, ...)
```

Arguments

x	A yr object or numeric vector. If numeric, era must be specified.
era	era object describing the target era, see era() .
...	Additional arguments passed to methods.
precision	Desired precision of the transformation, i.e. the transformed values are rounded to the nearest precision. If NA (the default), no rounding is performed and the exact transformed value is returned.

Details

Transformation between eras uses the scale, epoch, direction and unit parameters of the era definition. NA values for any of these parameters in the source or destination era will cause an error. This is most often encountered when either are measured in 'radiocarbon years', which cannot be related to a calendar time scale without **calibration** or un-calibration.

The transformation function is exact and treats years as a real number scale. This means that transformations between eras with different year units (e.g. Gregorian to Julian) and/or epochs not aligned to 1 January in the Gregorian calendar (e.g. Common Era to Islamic calendars) will likely return non-integer values. The precision argument provides a convenient way to round the result if you do not need this level of precision. It is also useful for working around the ambiguous definition of 'present' in various geological time-scales.

Value

A yr object in the era specified by era.

See Also

Other years with era functions: [yr\(\)](#), [yr_era\(\)](#)

Examples

```
x <- yr(10010:10001, "cal BP")  
yr_transform(x, era("BCE"))  
  
yr_transform(x, era("ka"), precision = 1)
```

Index

* era definition functions

era, [2](#)
eras, [3](#)

* era helper functions

era, [2](#)
era_parameters, [4](#)
era_year, [5](#)
era_year_parameters, [6](#)
is_era, [6](#)
is_era_year, [7](#)
is_yr, [8](#)
this_year, [9](#)

* functions for chronological ordering and extremes

yr_difference, [10](#)
yr_earlier_than, [11](#)
yr_extremes, [14](#)
yr_sort, [15](#)

* years with era functions

yr, [10](#)
yr_era, [13](#)
yr_transform, [16](#)

era, [2](#)
era(), [3–10](#), [13](#), [16](#)
era_direction(era_parameters), [4](#)
era_epoch(era_parameters), [4](#)
era_label(era_parameters), [4](#)
era_name(era_parameters), [4](#)
era_parameters, [3](#), [4](#), [5–9](#)
era_scale(era_parameters), [4](#)
era_unit(era_parameters), [4](#)
era_year, [5](#)
era_year(), [3–9](#)
era_year_days(era_year_parameters), [6](#)
era_year_label(era_year_parameters), [6](#)
era_year_parameters, [3](#), [5](#), [6](#), [7–9](#)
eras, [3](#)
eras(), [2](#), [3](#), [10](#)

is_era, [6](#)
is_era(), [3](#), [5](#), [6](#), [8](#), [9](#)
is_era_year, [7](#)
is_era_year(), [3](#), [5–7](#), [9](#)
is_valid_era(is_era), [6](#)
is_valid_yr(is_yr), [8](#)
is_yr, [8](#)
is_yr(), [3](#), [5–9](#)

max(), [14](#)
min(), [14](#)

range(), [14](#)

sort(), [15](#)

this_year, [9](#)
this_year(), [3](#), [5–9](#)

validate_era(is_era), [6](#)
validate_yr(is_yr), [8](#)

yr, [10](#), [11](#), [12](#), [14](#), [15](#)
yr(), [2](#), [13](#), [16](#)
yr_difference, [10](#)
yr_difference(), [12](#), [14](#), [15](#)
yr_earlier_than, [11](#)
yr_earlier_than(), [11](#), [14](#), [15](#)
yr_earliest(yr_extremes), [14](#)
yr_era, [13](#)
yr_era(), [10](#), [16](#)
yr_era<-(yr_era), [13](#)
yr_extremes, [11](#), [12](#), [14](#), [15](#)
yr_later_than(yr_earlier_than), [11](#)
yr_latest(yr_extremes), [14](#)
yr_not_earlier_than(yr_earlier_than),
[11](#)
yr_not_later_than(yr_earlier_than), [11](#)
yr_range(yr_extremes), [14](#)
yr_set_era(yr_era), [13](#)
yr_sort, [15](#)

`yr_sort()`, [11](#), [12](#), [14](#)

`yr_transform`, [16](#)

`yr_transform()`, [5](#), [10](#), [13](#)