Package: CFtime (via r-universe)

September 15, 2024

Title Using CF-Compliant Calendars with Climate Projection Data

Version 1.4.0.9000

Description Support for all calendars as specified in the Climate and Forecast (CF) Metadata Conventions for climate and forecasting data. The CF Metadata Conventions is widely used for distributing files with climate observations or projections, including the Coupled Model Intercomparison Project (CMIP) data used by climate change scientists and the Intergovernmental Panel on Climate Change (IPCC). This package specifically allows the user to work with any of the CF-compliant calendars (many of which are not compliant with POSIXt). The CF time coordinate is formally defined in the CF Metadata Conventions document available at <https://cfconventions.org/Data/cf-conventions/cf-conventions-1.11/cf-conventions.html#time-coordinate>.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports methods, utils

Suggests knitr, rmarkdown, ncdf4, RNetCDF, testthat (>= 3.0.0), stringr

URL https://github.com/pvanlaake/CFtime

BugReports https://github.com/pvanlaake/CFtime/issues

VignetteBuilder knitr

Config/testthat/edition 3

Repository https://pvanlaake.r-universe.dev

RemoteUrl https://github.com/pvanlaake/cftime

RemoteRef HEAD

RemoteSha dba18f12347312721a4c21b2920af9ab44a5db74

Contents

`+,CFtime,CFtime-method`

Merge two CFtime objects

Description

Two `CFtime` instances can be merged into one with this operator, provided that the units and calendars of the datums of the two instances are equivalent.

Usage

```
## S4 method for signature 'CFtime,CFtime'
e1 + e2
```

Arguments

`e1, e2`	CFtime. Instances of the `CFtime` class.

Details

If the origins of the two datums are not identical, the earlier origin is preserved and the offsets of the later origin are updated in the resulting CFtime instance.

The order of the two parameters is indirectly significant. The resulting `CFtime` instance will have the offsets of both instances in the order that they are specified. There is no reordering or removal of duplicates. This is because the time series are usually associated with a data set and the correspondence between the data in the files and the CFtime instance is thus preserved. When merging the data sets described by this time series, the order must be identical to the merging here.

Any bounds that were set will be removed. Use `bounds()` to retrieve the bounds of the individual `CFtime` instances and then set them again after merging the two instances.

Value

A `CFtime` object with a set of offsets composed of the offsets of the instances of `CFtime` that the operator operates on. If the datum units or calendars of the `CFtime` instances are not equivalent, an error is thrown.

Examples

```
e1 <- CFtime("days since 1850-01-01", "gregorian", 0:364)
e2 <- CFtime("days since 1850-01-01 00:00:00", "standard", 365:729)
e1 + e2
```

+,CFtime,numeric-method

Extend a CFtime object with additional offsets

Description

A `CFtime` instance can be extended by adding additional offsets using this operator.

Usage

```
## S4 method for signature 'CFtime,numeric'
e1 + e2
```

Arguments

`e1`	CFtime. Instance of the `CFtime` class.
`e2`	numeric. Vector of offsets to be added to the `CFtime` instance.

Details

The resulting CFtime instance will have its offsets in the order that they are added, meaning that the offsets from the CFtime instance come first and those from the numeric vector follow. There is no reordering or removal of duplicates. This is because the time series are usually associated with a data set and the correspondence between the two is thus preserved, if and only if the data sets are merged in the same order.

Note that when adding multiple vectors of offsets to a CFtime instance, it is more efficient to first concatenate the vectors and then do a final addition to the CFtime instance. So avoid `CFtime(definition, calendar, e1) + CFtime(definition, calendar, e2) + CFtime(definition, calendar, e3) + ...` but rather do `CFtime(definition, calendar) + c(e1, e2, e3, ...)`. It is the responsibility of the operator to ensure that the offsets of the different data sets are in reference to the same datum.

Note also that RNetCDF and ncdf4 packages both return the values of the "time" dimension as a 1-dimensional array. You have to `dim(time_values) <- NULL` to de-class the array to a vector before adding offsets to an existing CFtime instance.

Negative offsets will generate an error.

Any bounds that were set will be removed. Use bounds() to retrieve the bounds of the individual CFtime instances and then set them again after merging the two instances.

Value

A CFtime object with offsets composed of the CFtime instance and the numeric vector.

Examples

```
e1 <- CFtime("days since 1850-01-01", "gregorian", 0:364)
e2 <- 365:729
e1 + e2
```

==,CFtime,CFtime-method

Equivalence of CFtime objects

Description

This operator can be used to test if two CFtime objects represent the same CF-convention time coordinates. Two CFtime objects are considered equivalent if they have an equivalent datum and the same offsets.

Usage

```
## S4 method for signature 'CFtime,CFtime'
e1 == e2
```

Arguments

e1, e2	CFtime. Instances of the CFtime class.

Value

TRUE if the CFtime objects are equivalent, FALSE otherwise.

Examples

```
e1 <- CFtime("days since 1850-01-01", "gregorian", 0:364)
e2 <- CFtime("days since 1850-01-01 00:00:00", "standard", 0:364)
e1 == e2
```

---

as.character,CFtime-method

Return the timestamps contained in the CFtime instance.

Description

Return the timestamps contained in the CFtime instance.

Usage

```
## S4 method for signature 'CFtime'
as.character(x)
```

Arguments

x	The CFtime instance whose timestamps will be returned

Value

The timestamps in the specified CFtime instance.

Examples

```
cf <- CFtime("days since 1850-01-01", "julian", 0:364)
as.character(cf)
```

as_timestamp *Create a vector that represents CF timestamps*

Description

This function generates a vector of character strings or POSIXcts that represent the date and time in a selectable combination for each offset.

Usage

```
as_timestamp(cf, format = NULL, asPOSIX = FALSE)
```

Arguments

cf	CFtime. The CFtime instance that contains the offsets to use.
format	character. A character string with either of the values "date" or "timestamp". If the argument is not specified, the format used is "timestamp" if there is time information, "date" otherwise.
asPOSIX	logical. If TRUE, for "standard", "gregorian" and "proleptic_gregorian" calendars the output is a vector of POSIXct - for other calendars the result is NULL. Default value is FALSE.

Details

The character strings use the format YYYY-MM-DDThh:mm:ss±hhmm, depending on the format specifier. The date in the string is not necessarily compatible with POSIXt - in the 360_day calendar 2017-02-30 is valid and 2017-03-31 is not.

For the "standard", "gregorian" and "proleptic_gregorian" calendars the output can also be generated as a vector of POSIXct values by specifying asPOSIX = TRUE.

Value

A character vector where each element represents a moment in time according to the format specifier.

See Also

The format() function gives greater flexibility through the use of strptime-like format specifiers.

Examples

```
cf <- CFtime("hours since 2020-01-01", "standard", seq(0, 24, by = 0.25))
as_timestamp(cf, "timestamp")

cf2 <- CFtime("days since 2002-01-21", "standard", 0:20)
tail(as_timestamp(cf2, asPOSIX = TRUE))
```

```
tail(as_timestamp(cf2))

tail(as_timestamp(cf2 + 1.5))
```

---

bounds *Bounds of the time offsets*

Description

CF-compliant NetCDF files store time information as a single offset value for each step along the dimension, typically centered on the valid interval of the data (e.g. 12-noon for day data). Optionally, the lower and upper values of the valid interval are stored in a so-called "bounds" variable, as an array with two rows (lower and higher value) and a column for each offset. With function `bounds()<-` those bounds can be set for a CFtime instance. The bounds can be retrieved with the `bounds()` function.

Usage

```
bounds(x, format)

## S4 method for signature 'CFtime'
bounds(x, format)

bounds(x) <- value

## S4 replacement method for signature 'CFtime'
bounds(x) <- value
```

Arguments

`x`	A `CFtime` instance
`format`	Optional. A single string with format specifiers, see `format()` for details.
`value`	A `matrix` (or `array`) with dimensions (2, length(offsets)) giving the lower (first row) and higher (second row) bounds of each offset (this is the format that the CF Metadata Conventions uses for storage in NetCDF files). Use `FALSE` to unset any previously set bounds, `TRUE` to set regular bounds at mid-points between the offsets (which must be regular as well).

Value

If bounds have been set, an array of bounds values with dimensions (2, length(offsets)). The first row gives the lower bound, the second row the upper bound, with each column representing an offset of `x`. If the `format` argument is specified, the bounds values are returned as strings according to the format. `NULL` when no bounds have been set.

Examples

```
cf <- CFtime("days since 2024-01-01", "standard", seq(0.5, by = 1, length.out = 366))
as_timestamp(cf)[1:3]
bounds(cf) <- rbind(0:365, 1:366)
bounds(cf)[, 1:3]
bounds(cf, "%d-%b-%Y")[, 1:3]
```

`CFfactor` *Create a factor from the offsets in an CFtime instance*

Description

With this function a factor can be generated for the time series, or a part thereof, contained in the `CFtime` instance. This is specifically interesting for creating factors from the date part of the time series that aggregate the time series into longer time periods (such as month) that can then be used to process daily CF data sets using, for instance, `tapply()`.

Usage

```
CFfactor(cf, period = "month", epoch = NULL)
```

Arguments

`cf`	CFtime. An instance of the `CFtime` class whose offsets will be used to construct the factor.
`period`	character. A character string with one of the values "year", "season", "quarter", "month" (the default), "dekad" or "day".
`epoch`	numeric or list, optional. Vector of years for which to construct the factor, or a list whose elements are each a vector of years. If `epoch` is not specified, the factor will use the entire time series for the factor.

Details

The factor will respect the calendar of the datum that the time series is built on. For `period`s longer than a day this will result in a factor where the calendar is no longer relevant (because calendars impacts days, not dekads, months, quarters, seasons or years).

The factor will be generated in the order of the offsets of the `CFtime` instance. While typical CF-compliant data sources use ordered time series there is, however, no guarantee that the factor is ordered as multiple `CFtime` objects may have been merged out of order.

If the `epoch` parameter is specified, either as a vector of years to include in the factor, or as a list of such vectors, the factor will only consider those values in the time series that fall within the list of years, inclusive of boundary values. Other values in the factor will be set to `NA`. The years need not be contiguous, within a single vector or among the list items, or in order.

The following periods are supported by this function:

- `year`, the year of each offset is returned as "YYYY".

- season, the meteorological season of each offset is returned as "Sx", with x being 1-4, preceeded by "YYYY" if no epoch is specified. Note that December dates are labeled as belonging to the subsequent year, so the date "2020-12-01" yields "2021S1". This implies that for standard CMIP files having one or more full years of data the first season will have data for the first two months (January and February), while the final season will have only a single month of data (December).
- quarter, the calendar quarter of each offset is returned as "Qx", with x being 1-4, preceeded by "YYYY" if no epoch is specified.
- month, the month of each offset is returned as "01" to "12", preceeded by "YYYY-" if no epoch is specified. This is the default period.
- dekad, ten-day periods are returned as "Dxx", where xx runs from "01" to "36", preceeded by "YYYY" if no epoch is specified. Each month is subdivided in dekads as follows: 1- days 01 - 10; 2- days 11 - 20; 3- remainder of the month.
- day, the month and day of each offset are returned as "MM-DD", preceeded by "YYYY-" if no epoch is specified.

It is not possible to create a factor for a period that is shorter than the temporal resolution of the source data set from which the cf argument derives. As an example, if the source data set has monthly data, a dekad or day factor cannot be created.

Creating factors for other periods is not supported by this function. Factors based on the timestamp information and not dependent on the calendar can trivially be constructed from the output of the as_timestamp() function.

For non-epoch factors the attribute 'CFtime' of the result contains a CFtime instance that is valid for the result of applying the factor to a data set that the cf argument is associated with. In other words, if CFtime instance 'Acf' describes the temporal dimension of data set 'A' and a factor 'Af' is generated from 'Acf', then attr(Af, "CFtime") describes the temporal dimension of the result of, say, apply(A, 1:2, tapply, Af, FUN). The 'CFtime' attribute is NULL for epoch factors.

Value

If epoch is a single vector or not specified, a factor with a length equal to the number of offsets in cf. If epoch is a list, a list with the same number of elements and names as epoch, each containing a factor. Elements in the factor will be set to NA for time series values outside of the range of specified years.

The factor, or factors in the list, have attributes 'period', 'epoch' and 'CFtime'. Attribute 'period' holds the value of the period argument. Attribute 'epoch' indicates the number of years that are included in the epoch, or -1 if no epoch is provided. Attribute 'CFtime' holds an instance of CFtime that has the same definition as cf, but with offsets corresponding to the mid-point of non-epoch factor levels; if the epoch argument is specified, attribute 'CFtime' is NULL.

See Also

cut() creates a non-epoch factor for arbitrary cut points.

Examples

```
cf <- CFtime("days since 1949-12-01", "360_day", 19830:54029)
```

```
# Create a dekad factor for the whole time series
f <- CFfactor(cf, "dekad")

# Create three monthly factors for early, mid and late 21st century epochs
ep <- CFfactor(cf, epoch = list(early = 2021:2040, mid = 2041:2060, late = 2061:2080))
```

`CFfactor_coverage` *Coverage of time elements for each factor level*

Description

This function calculates the number of time elements, or the relative coverage, in each level of a factor generated by `CFfactor()`.

Usage

```
CFfactor_coverage(cf, f, coverage = "absolute")
```

Arguments

`cf`	CFtime. An instance of CFtime.
`f`	factor or list. A factor or a list of factors derived from the parameter `cf`. The factor or list thereof should generally be generated by the function `CFfactor()`.
`coverage`	"absolute" or "relative".

Value

If `f` is a factor, a numeric vector with a length equal to the number of levels in the factor, indicating the number of units from the time series in `cf` contained in each level of the factor when `coverage = "absolute"` or the proportion of units present relative to the maximum number when `coverage = "relative"`. If `f` is a list of factors, a list with each element a numeric vector as above.

Examples

```
cf <- CFtime("days since 2001-01-01", "365_day", 0:364)
f <- CFfactor(cf, "dekad")
CFfactor_coverage(cf, f, "absolute")
```

CFfactor_units *Number of base time units in each factor level*

Description

Given a factor as returned by `CFfactor()` and the `CFtime` instance from which the factor was derived, this function will return a numeric vector with the number of time units in each level of the factor.

Usage

```
CFfactor_units(cf, f)
```

Arguments

`cf`	CFtime. An instance of CFtime.
`f`	factor or list. A factor or a list of factors derived from the parameter `cf`. The factor or list thereof should generally be generated by the function `CFfactor()`.

Details

The result of this function is useful to convert between absolute and relative values. Climate change anomalies, for instance, are usually computed by differencing average values between a future period and a baseline period. Going from average values back to absolute values for an aggregate period (which is typical for temperature and precipitation, among other variables) is easily done with the result of this function, without having to consider the specifics of the calendar of the data set.

If the factor `f` is for an epoch (e.g. spanning multiple years and the levels do not indicate the specific year), then the result will indicate the number of time units of the period in a regular single year. In other words, for an epoch of 2041-2060 and a monthly factor on a standard calendar with a `days` unit, the result will be `c(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)`. Leap days are thus only considered for the `366_day` and `all_leap` calendars.

Note that this function gives the number of time units in each level of the factor - the actual number of data points in the `cf` instance per factor level may be different. Use `CFfactor_coverage()` to determine the actual number of data points or the coverage of data points relative to the factor level.

Value

If `f` is a factor, a numeric vector with a length equal to the number of levels in the factor, indicating the number of time units in each level of the factor. If `f` is a list of factors, a list with each element a numeric vector as above.

Examples

```
cf <- CFtime("days since 2001-01-01", "365_day", 0:364)
f <- CFfactor(cf, "dekad")
CFfactor_units(cf, f)
```

CFparse *Parse series of timestamps in CF format to date-time elements*

Description

This function will parse a vector of timestamps in ISO8601 or UDUNITS format into a data frame with columns for the elements of the timestamp: year, month, day, hour, minute, second, time zone. Those timestamps that could not be parsed or which represent an invalid date in the indicated CFtime instance will have NA values for the elements of the offending timestamp (which will generate a warning).

Usage

```
CFparse(cf, x)
```

Arguments

cf	CFtime. An instance of CFtime indicating the CF calendar and datum to use when parsing the date.
x	character. Vector of character strings representing timestamps in ISO8601 extended or UDUNITS broken format.

Details

The supported formats are the *broken timestamp* format from the UDUNITS library and ISO8601 *extended*, both with minor changes, as suggested by the CF Metadata Conventions. In general, the format is YYYY-MM-DD hh:mm:ss.sss hh:mm. The year can be from 1 to 4 digits and is interpreted literally, so 79-10-24 is the day Mount Vesuvius erupted and destroyed Pompeii, not 1979-10-24. The year and month are mandatory, all other fields are optional. There are defaults for all missing values, following the UDUNITS and CF Metadata Conventions. Leading zeros can be omitted in the UDUNITS format, but not in the ISO8601 format. The optional fractional part can have as many digits as the precision calls for and will be applied to the smallest specified time unit. In the result of this function, if the fraction is associated with the minute or the hour, it is converted into a regular hh:mm:ss.sss format, i.e. any fraction in the result is always associated with the second, rounded down to milli-second accuracy. The separator between the date and the time can be a single whitespace character or a T.

The time zone is optional and should have at least the hour or Z if present, the minute is optional. The time zone hour can have an optional sign. In the UDUNITS format the separator between the time and the time zone must be a single whitespace character, in ISO8601 there is no separation between the time and the timezone. Time zone names are not supported (as neither UDUNITS nor ISO8601 support them) and will cause parsing to fail when supplied, with one exception: the designator "UTC" is silently dropped (i.e. interpreted as "00:00").

Currently only the extended formats (with separators between the elements) are supported. The vector of timestamps may have any combination of ISO8601 and UDUNITS formats.

Timestamps that are prior to the datum are not allowed. The corresponding row in the result will have NA values.

Value

A data frame with constituent elements of the parsed timestamps in numeric format. The columns are year, month, day, hour, minute, second (with an optional fraction), time zone (character string), and the corresponding offset value from the datum. Invalid input data will appear as NA - if this is the case, a warning message will be displayed - other missing information on input will use default values.

Examples

```
cf <- CFtime("days since 0001-01-01", "proleptic_gregorian")

# This will have `NA`s on output and generate a warning
timestamps <- c("2012-01-01T12:21:34Z", "12-1-23", "today",
                "2022-08-16T11:07:34.45-10", "2022-08-16 10.5+04")
CFparse(cf, timestamps)
```

CFtime *Create a CFtime object*

Description

This function creates an instance of the CFtime class. The arguments to the call are typically read from a CF-compliant data file with climatological observations or climate projections. Specification of arguments can also be made manually in a variety of combinations.

Usage

```
CFtime(definition, calendar = "standard", offsets = NULL)
```

Arguments

definition	character. A character string describing the time coordinate of a CF-compliant data file.
calendar	character. A character string describing the calendar to use with the time dimension definition string. Default value is "standard".
offsets	numeric or character, optional. When numeric, a vector of offsets from the origin in the time series. When a character vector, timestamps in ISO8601 or UDUNITS format. When a character string, a timestamp in ISO8601 or UDUNITS format and then a time series will be generated with a separation between steps equal to the unit of measure in the definition, inclusive of the definition timestamp. The unit of measure of the offsets is defined by the time series definition.

Value

An instance of the CFtime class.

Examples

```
CFtime("days since 1850-01-01", "julian", 0:364)

CFtime("hours since 2023-01-01", "360_day", "2023-01-30T23:00")
```

`CFtime-class` *CF Metadata Conventions time representation*

Description

CF Metadata Conventions time representation

Value

An object of class CFtime.

Slots

`datum` CFdatum. The origin upon which the `offsets` are based.

`resolution` numeric. The average number of time units between offsets.

`offsets` numeric. A vector of offsets from the datum.

`bounds` Optional, the bounds for the offsets. If not set, it is the logical value `FALSE`. If set, it is the logical value `TRUE` if the bounds are regular with respect to the regularly spaced offsets (e.g. successive bounds are contiguous and at mid-points between the offsets); otherwise a `matrix` with columns for `offsets` and low values in the first row, high values in the second row.

`cut,CFtime-method` *Create a factor for a CFtime instance*

Description

Method for `base::cut()` applied to CFtime objects.

Usage

```
## S4 method for signature 'CFtime'
cut(x, breaks, ...)
```

Arguments

`x`	An instance of CFtime.
`breaks`	A character string of a factor period (see `CFfactor()` for a description), or a character vector of timestamps that conform to the calendar of `x`, with a length of at least 2. Timestamps must be given in ISO8601 format, e.g. "2024-04-10 21:31:43".
`...`	Ignored.

Details

When breaks is one of "year", "season", "quarter", "month", "dekad", "day" a factor is generated like by CFfactor().

When breaks is a vector of character timestamps a factor is produced with a level for every interval between timestamps. The last timestamp, therefore, is only used to close the interval started by the pen-ultimate timestamp - use a distant timestamp (e.g. range(x)[2]) to ensure that all offsets to the end of the CFtime time series are included, if so desired. The last timestamp will become the upper bound in the CFtime instance that is returned as an attribute to this function so a sensible value for the last timestamp is advisable. The earliest timestamp cannot be earlier than the origin of the datum of x.

This method works similar to base::cut.POSIXt() but there are some differences in the arguments: for breaks the set of options is different and no preceding integer is allowed, labels are always assigned using values of breaks, and the interval is always left-closed.

Value

A factor with levels according to the breaks argument, with attributes 'period', 'epoch' and 'CFtime'. When breaks is a factor period, attribute 'period' has that value, otherwise it is '"day"'. When breaks is a character vector of timestamps, attribute 'CFtime' holds an instance of CFtime that has the same definition as x, but with (ordered) offsets generated from the breaks. Attribute 'epoch' is always -1.

See Also

CFfactor() produces a factor for several fixed periods, including for epochs.

Examples

```
x <- CFtime("days since 2021-01-01", "365_day", 0:729)
breaks <- c("2022-02-01", "2021-12-01", "2023-01-01")
cut(x, breaks)
```

---

definition *Properties of a CFtime object*

Description

These functions return the properties of an instance of the CFtime class. The properties are all read-only, but offsets can be added using the + operator.

Usage

```
definition(cf)

calendar(cf)

unit(cf)
```

```
origin(cf)

timezone(cf)

offsets(cf)

resolution(cf)
```

Arguments

cf	CFtime. An instance of CFtime.

Value

calendar() and unit() return a character string. origin() returns a data frame of timestamp elements with a single row of data. timezone() returns the datum time zone as a character string. offsets() returns a vector of offsets or NULL if no offsets have been set.

Functions

- definition(): The definition string of the CFtime instance
- calendar(): The calendar of the CFtime instance
- unit(): The unit of the CFtime instance
- origin(): The origin of the CFtime instance in timestamp elements
- timezone(): The time zone of the datum of the CFtime instance as a character string
- offsets(): The offsets of the CFtime instance as a vector
- resolution(): The average separation between the offsets in the CFtime instance

Examples

```
cf <- CFtime("days since 1850-01-01", "julian", 0:364)
definition(cf)
calendar(cf)
unit(cf)
timezone(cf)
origin(cf)
offsets(cf)
resolution(cf)
```

deprecated_functions *Deprecated functions*

Description

These functions are deprecated and should no longer be used in new code. The below table gives the replacement function to use instead. The function arguments of the replacement function are the same as those of the deprecated function if no arguments are given in the table.

Deprecated function	Replacement function
CFcomplete()	`is_complete()`
CFmonth_days()	`month_days()`
CFrange()	`range()`
CFsubset()	`slab()`
CFtimestamp()	`as_timestamp()`

Usage

```
CFtimestamp(cf, format = NULL, asPOSIX = FALSE)

CFmonth_days(cf, x = NULL)

CFcomplete(x)
```

Arguments

`cf`, `x`, `format`, `asPOSIX`
: See replacement functions.

Value

See replacement functions.

format,CFtime-method *Format time elements using format specifiers*

Description

Format timestamps using a specific format string, using the specifiers defined for the `base::strptime()` function, with limitations. The only supported specifiers are `bBdeFhHIjmMpRSTYz%`. Modifiers `E` and `O` are silently ignored. Other specifiers, including their percent sign, are copied to the output as if they were adorning text.

Usage

```
## S4 method for signature 'CFtime'
format(x, format)
```

Arguments

x	CFtime. A CFtime instance whose offsets will be returned as timestamps.
format	character. A character string with strptime format specifiers. If omitted, the most economical format will be used: a full timestamp when time information is available, a date otherwise.

Details

The formatting is largely oblivious to locale. The reason for this is that certain dates in certain calendars are not POSIX-compliant and the system functions necessary for locale information thus do not work consistently. The main exception to this is the (abbreviated) names of months (`bB`), which could be useful for pretty printing in the local language. For separators and other locale-specific adornments, use local knowledge instead of depending on system locale settings; e.g. specify `%m/%d/%Y` instead of `%D`.

Week information, including weekday names, is not supported at all as a "week" is not defined for non-standard CF calendars and not generally useful for climate projection data. If you are working with observed data and want to get pretty week formats, use the `as_timestamp()` function to generate `POSIXct` timestamps (observed data generally uses a standard calendar) and then use the `base::format()` function which supports the full set of specifiers.

Value

A vector of character strings with a properly formatted timestamp. Any format specifiers not recognized or supported will be returned verbatim.

Examples

```
cf <- CFtime("days since 2020-01-01", "standard", 0:365)
format(cf, "%Y-%b")

# Use system facilities on a standard calendar
format(as_timestamp(cf, asPOSIX = TRUE), "%A, %x")
```

`indexOf,ANY,CFtime-method`

Find the index of timestamps in the time series

Description

In the CFtime instance y, find the index in the time series for each timestamp given in argument x. Values of x that are before the earliest value in y will be returned as 0 (except when the value is before the datum of y, in which case the value returned is NA); values of x that are after the latest values in y will be returned as .Machine$integer.max. Alternatively, when x is a numeric vector of index values, return the valid indices of the same vector, with the side effect being the attribute "CFtime" associated with the result.

Usage

```
## S4 method for signature 'ANY,CFtime'
indexOf(x, y, method = "constant")
```

Arguments

x	Vector of character, POSIXt or Date values to find indices for, or a numeric vector.
y	CFtime instance.
method	Single value of "constant" or "linear". If "constant" or when bounds are set on argument y, return the index value for each match. If "linear", return the index value with any fractional value.

Details

Timestamps can be provided as vectors of character strings, POSIXct or Date.

Matching also returns index values for timestamps that fall between two elements of the time series - this can lead to surprising results when time series elements are positioned in the middle of an interval (as the CF Metadata Conventions instruct us to "reasonably assume"): a time series of days in January would be encoded in a netCDF file as c("2024-01-01 12:00:00", "2024-01-02 12:00:00", "2024-01-03 12:00:00", ...) so x <- c("2024-01-01", "2024-01-02", "2024-01-03") would result in (NA, 1, 2) (or (NA, 1.5, 2.5) with method = "linear") because the date values in x are at midnight. This situation is easily avoided by ensuring that y has bounds set (use bounds(y) <- TRUE as a proximate solution if bounds are not stored in the netCDF file). See the Examples.

If bounds are set, the indices are taken from those bounds. Returned indices may fall in between bounds if the latter are not contiguous, with the exception of the extreme values in x.

Values of x that are not valid timestamps according to the calendar of y will be returned as NA.

x can also be a numeric vector of index values, in which case the valid values in x are returned. If negative values are passed, the positive counterparts will be excluded and then the remainder returned. Positive and negative values may not be mixed. Using a numeric vector has the side effect that the result has the attribute "CFtime" describing the temporal dimension of the slice. If index values outside of the range of y (1:length(y)) are provided, an error will be thrown.

Value

A numeric vector giving indices into the "time" dimension of the dataset associated with y for the values of x. If there is at least 1 valid index, then attribute "CFtime" contains an instance of CFtime

that describes the dimension of filtering the dataset associated with y with the result of this function, excluding any NA, 0 and .Machine$integer.max values.

Examples

```
cf <- CFtime("days since 2020-01-01", "360_day", 1440:1799 + 0.5)
as_timestamp(cf)[1:3]
x <- c("2024-01-01", "2024-01-02", "2024-01-03")
indexOf(x, cf)
indexOf(x, cf, method = "linear")

bounds(cf) <- TRUE
indexOf(x, cf)

# Non-existent calendar day in a `360_day` calendar
x <- c("2024-03-30", "2024-03-31", "2024-04-01")
indexOf(x, cf)

# Numeric x
indexOf(c(29, 30, 31), cf)
```

---

is_complete *Indicates if the time series is complete*

Description

This function indicates if the time series is complete, meaning that the time steps are equally spaced and there are thus no gaps in the time series.

Usage

```
is_complete(x)
```

Arguments

x	An instance of the CFtime class

Details

This function gives exact results for time series where the nominal *unit of separation* between observations in the time series is exact in terms of the datum unit. As an example, for a datum unit of "days" where the observations are spaced a fixed number of days apart the result is exact, but if the same datum unit is used for data that is on a monthly basis, the *assessment* is approximate because the number of days per month is variable and dependent on the calendar (the exception being the 360_day calendar, where the assessment is exact). The *result* is still correct in most cases (including all CF-compliant data sets that the developers have seen) although there may be esoteric constructions of CFtime and offsets that trip up this implementation.

Value

logical. TRUE if the time series is complete, with no gaps; FALSE otherwise. If no offsets have been added to the CFtime instance, NA is returned.

Examples

```
cf <- CFtime("days since 1850-01-01", "julian", 0:364)
is_complete(cf)
```

---

length,CFtime-method *The length of the offsets contained in the CFtime instance.*

Description

The length of the offsets contained in the CFtime instance.

Usage

```
## S4 method for signature 'CFtime'
length(x)
```

Arguments

x	The CFtime instance whose length will be returned

Value

The number of offsets in the specified CFtime instance.

Examples

```
cf <- CFtime("days since 1850-01-01", "julian", 0:364)
length(cf)
```

---

month_days *Return the number of days in a month given a certain CF calendar*

Description

Given a vector of dates as strings in ISO 8601 or UDUNITS format and a CFtime object, this function will return a vector of the same length as the dates, indicating the number of days in the month according to the calendar specification. If no vector of days is supplied, the function will return an integer vector of length 12 with the number of days for each month of the calendar (disregarding the leap day for standard and julian calendars).

Usage

```
month_days(cf, x = NULL)
```

Arguments

cf	CFtime. The CFtime definition to use.
x	character. An optional vector of dates as strings with format YYYY-MM-DD. Any time part will be silently ingested.

Value

A vector indicating the number of days in each month for the vector of dates supplied as a parameter to the function. If no dates are supplied, the number of days per month for the calendar as a vector of length 12. Invalidly specified dates will result in an NA value.

See Also

When working with factors generated by CFfactor(), it is usually better to use CFfactor_units() as that will consider leap days for non-epoch factors. CFfactor_units() can also work with other time periods and datum units, such as "hours per month", or "days per season".

Examples

```
dates <- c("2021-11-27", "2021-12-10", "2022-01-14", "2022-02-18")
cf <- CFtime("days since 1850-01-01", "standard")
month_days(cf, dates)

cf <- CFtime("days since 1850-01-01", "360_day")
month_days(cf, dates)

cf <- CFtime("days since 1850-01-01", "all_leap")
month_days(cf, dates)

month_days(cf)
```

---

range,CFtime-method *Extreme time series values*

Description

Character representation of the extreme values in the time series

Usage

```
## S4 method for signature 'CFtime'
range(x, format = "", bounds = FALSE, ..., na.rm = FALSE)
```

Arguments

x	An instance of the CFtime class.
format	A character string with format specifiers, optional. If it is missing or an empty string, the most economical ISO8601 format is chosen: "date" when no time information is present in x, "timestamp" otherwise. Otherwise a suitable format specifier can be provided.
bounds	Logical to indicate if the extremes from the bounds should be used, if set. Defaults to FALSE.
...	Ignored.
na.rm	Ignored.

Value

Vector of two character representations of the extremes of the time series.

Examples

```
cf <- CFtime("days since 1850-01-01", "julian", 0:364)
range(cf)
range(cf, "%Y-%b-%e")
```

---

slab *Which time steps fall within two extreme values*

Description

Given two extreme character timestamps, return a logical vector of a length equal to the number of time steps in the CFtime instance with values TRUE for those time steps that fall between the two extreme values, FALSE otherwise. This can be used to select slices from the time series in reading or analysing data.

Usage

```
slab(x, extremes, rightmost.closed = FALSE)
```

Arguments

x	CFtime. The time series to operate on.
extremes	character. Vector of two timestamps that represent the extremes of the time period of interest. The timestamps must be in increasing order. The timestamps need not fall in the range of the time steps in the CFtime stance.
rightmost.closed	Is the larger extreme value included in the result? Default is FALSE.

Details

If bounds were set these will be preserved.

Value

A logical vector with a length equal to the number of time steps in `x` with values `TRUE` for those time steps that fall between the two extreme values, `FALSE` otherwise. The earlier timestamp is included, the later timestamp is excluded. A specification of `c("2022-01-01", "2023-01-01")` will thus include all time steps that fall in the year 2022.

Examples

```
cf <- CFtime("hours since 2023-01-01 00:00:00", "standard", 0:23)
slab(cf, c("2022-12-01", "2023-01-01 03:00"))
```

Index