Package 'CFtime'

November 2, 2023

Title Using CF-Compliant Calendars with Climate Projection Data

Version 1.2.0

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-1.10/cf-conventions.html#time-coordinate>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Imports methods, utils

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

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

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

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Patrick Van Laake [aut, cre, cph]

Maintainer Patrick Van Laake <patrick@vanlaake.net>

Repository CRAN

Date/Publication 2023-11-02 16:20:02 UTC

R topics documented:

`+,CFtime,CFtime-method`

Merge two CFtime objects

Description

Two `CFtime` instances can be merged into one with this operator, provided that 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

The order of the two parameters is indirectly significant. The resulting `CFtime` 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 two is thus preserved. When merging the data sets described by this time series, the order must be identical to the ordering here.

Value

A `CFtime` object with a set of offsets equal to the offsets of the instances of `CFtime` that the operator operates on. If the datums 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, e1) + c(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.

Negative offsets will generate an error.

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
```

CFcomplete *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

```
CFcomplete(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 monthly a 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)
CFcomplete(cf)
```

`CFdefinition` *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

```
CFdefinition(cf)

CFcalendar(cf)

CFunit(cf)

CForigin(cf)

CFoffsets(cf)

CFresolution(cf)
```

Arguments

`cf`	CFtime. An instance of `CFtime`.

Value

CFcalendar() and CFunit() return an atomic character string. CForigin() returns a data frame of timestamp elements with a single row of data. CFoffsets() returns a vector of offsets or NULL if no offsets have been set.

Functions

- CFdefinition(): The defintion string of the CFtime instance
- CFcalendar(): The calendar of the CFtime instance
- CFunit(): The unit of the CFtime instance
- CForigin(): The origin of the CFtime instance in timestamp elements
- CFoffsets(): The offsets of the CFtime instance as a vector
- CFresolution(): The average separation between the offsets in the CFtime instance

Examples

```
cf <- CFtime("days since 1850-01-01", "julian", 0:364)
CFdefinition(cf)
CFcalendar(cf)
CFunit(cf)
CForigin(cf)
CFoffsets(cf)
CFresolution(cf)
```

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 atomic instance of the CFtime class whose offsets will be used to construct the factor.
period	character. An atomic character string with one of the values "year", "season", "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 or seasons).

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 "DJF", "MAM", "JJA" or "SON", 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 "2021-DJF". 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).
- `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 `CFtimestamp()` function.

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.

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)
```

CFmonth_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

```
CFmonth_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")
CFmonth_days(cf, dates)

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

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

CFmonth_days(cf)
```

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 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. The separator between the date and the time can be a single whitespace character or a `T`; in the UDUNITS format the separator between the time and the time zone must be a single whitespace character.

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)
```

CFrange *Extreme time series values*

Description

Character representation of the extreme values in the time series

Usage

```
CFrange(x)

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

Arguments

x	An instance of the CFtime class

Value

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

Methods (by class)

- CFrange(CFtime): Extreme values of the time series

Examples

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

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. An atomic string describing the time coordinate of a CF-compliant data file.
calendar	character. An atomic 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 an atomic 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 atomic origin upon which the `offsets` are based.

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

`time` data.frame. Data frame of numeric date elements year, month, day, hour, minute, second and time zone from offsets defined by the `datum`, as well as the offset itself.

`CFtimestamp` *Create a vector that represents CF timestamps*

Description

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

Usage

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

Arguments

`cf`	CFtime. The `CFtime` instance that contains the offsets to use.
`format`	character. An atomic 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±hh:mm`, 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. Time zone information is not represented.

Examples

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

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

tail(CFtimestamp(cf2))

tail(CFtimestamp(cf2 + 1.5))
```

Index