Package 'timeDate'

July 17, 2025

Title Rmetrics - Chronological and Calendar Objects

Version 4041.110

Description The 'timeDate' class fulfils the conventions of the ISO 8601 standard as well as of the ANSI C and POSIX standards. Beyond these standards it provides the ``Financial Center'' concept which allows to handle data records collected in different time zones and mix them up to have always the proper time stamps with respect to your personal financial center, or alternatively to the GMT reference time. It can thus also handle time stamps from historical data records from the same time zone, even if the financial centers changed day light saving times at different calendar dates.

Depends R (>= 3.6.0), methods

Imports graphics, utils, stats

Suggests RUnit

License GPL (>= 2)

Encoding UTF-8

URL https://geobosh.github.io/timeDateDoc/ (doc), https://r-forge.r-project.org/scm/viewvc.php/pkg/timeDate/?root=rmetrics (devel), https://www.rmetrics.org

BugReports https://r-forge.r-project.org/projects/rmetrics

NeedsCompilation no

Author Diethelm Wuertz [aut] (original code),
Tobias Setz [aut],
Yohan Chalabi [aut],
Martin Maechler [ctb] (ORCID: <https://orcid.org/0000-0002-8685-9910>),
Joe W. Byers [ctb],
Georgi N. Boshnakov [cre, aut]

Maintainer Georgi N. Boshnakov <georgi.boshnakov@manchester.ac.uk>

Repository CRAN

Date/Publication 2024-09-22 10:10:17 UTC

Contents

`timeDate-package` *Utilities and tools package*

Description

Package of calendar, date, time tools and utilities for Rmetrics.

Overview of Topics

This help file describes the concepts and methods behind the S4 `"timeDate"` class used in Rmetrics for financial data and time management together with the management of public and ecclesiastical holidays.

The `"timeDate"` class fulfils the conventions of the ISO 8601 standard as well as of the ANSI C and POSIX standards. Beyond these standards it provides the "Financial Center" concept which allows to handle data records collected in different time zones and mix them up to have always the proper time stamps with respect to your personal financial center, or alternatively to the GMT reference time. It can thus also handle time stamps from historical data records from the same time zone, even if the financial centers changed day light saving times at different calendar dates.

Moreover `"timeDate"` is almost compatible with the `"timeDate"` class in Insightful's SPlus `"timeDate"` class. If you move between the two worlds of R and SPlus, you will not have to rewrite your code. This is important for business applications.

The `"timeDate"` class offers not only date and time functionality but it also offers sophisticated calendar manipulations for business days, weekends, public and ecclesiastical holidays.

This help page is presented in four sections:

1. S4 `"timeDate"` Class and Functions
2. Operations on `"timeDate"` Objects
3. Daylight Saving Time and Financial Centers
4. Holidays and Holiday Calendars

1. S4 "timeDate" Class and Generator Functions

Date and time stamps are represented by an S4 object of class "timeDate".

```
setClass("timeDate",
representation(
Data = "POSIXct",
format = "character",
FinCenter = "character"
))
```

They have three slots. The @Data slot holds the time stamps which are POSIXct formatted as specified in the @format slot. The time stamps are local and belong to the financial center expressed through the slot @FinCenter.

There are several possibilities to generate a "timeDate" object. The most forward procedure is to use one of the following functions:

timeDate – Creates a "timeDate" object from scratch,
timeSequence – creates a sequence of "timeDate" objects,
timeCalendar – creates a "timeDate" object from calendar atoms,
Sys.timeDate – returns the current date and time as a "timeDate" object.

With the function timeDate you can create "timeDate" objects from scratch by specifying a character vector of time stamps and a financial center which the character vector belongs to. "GMT" is used by default as the reference for all date/time operations. But you can set the variable myFinCenter to your local financial center reference if you want to reference dates/time to it.

Examples:

```
# Show My local Financial Center - Note, by Default this is "GMT"
getRmetricsOptions("myFinCenter")

# Compose Character Vectors of Dates and Times:
Dates <- c("1989-09-28","2001-01-15","2004-08-30","1990-02-09")
Times <- c(  "23:12:55",  "10:34:02",  "08:30:00",  "11:18:23")
charvec = paste(Dates, Times)

# Create a 'timeDate' object
timeDate(charvec)

# Create a 'timeDate' object with my financial center set to Zurich
myFinCenter <- "Zurich"
timeDate(charvec)

# if the 'timeDate' was recorded in a different financial center, it
# will be automatically converted to your financial center,
# i.e. "Zurich".
```

```
timeDate(charvec, zone = "Tokyo")

# You can also convert a recorded 'timeDate' from your financial
# center "Zurich" to another one, for example "NewYork".
timeDate(charvec, FinCenter = "NewYork")
```

NOTE: Rmetrics has implemented an automated date/time format identifier for many common date/time formats which tries to automatically recognise the format for the character vector of dates and times. You can have a look at
`whichFormat(charvec)`.

NOTE: Rmetrics always uses the midnight standard on dates and times. You can see it with
`.midnightStandard("2008-01-31 24:00:00")`

Alternatively we can create a sequence of `"timeDate"` objects with the help of the function `timeSequence`. This can be done in several ways, either by specifying the range of the data through the arguments `from` and `to`, or when `from` is missing, by setting the argument `length.out` of the desired series. Note in the case of a monthly sequence, you have further options. For example you can generate the series with the first or last day in each month, or use more complex rules like the last or n-th Friday in every month.

Examples:

```
# Lets work in an international environment:
setRmetricsOptions(myFinCenter = "GMT")

# Your 'timeDate' is now in the Financial Center "GMT"
timeDate(charvec)

# Daily January 2008 Sequence:
timeSequence(from = "2008-01-01", to = "2008-01-31", by = "day")

# Monthly 2008 Sequence:
tS = timeSequence(from = "2008-01-01", to = "2008-12-31", by = "month")
tS

# Do you want the last Day or the last Friday in Month Data ?
timeLastDayInMonth(tS)
timeLastNdayInMonth(tS, nday = 5)
```

A third possibility is to create `"timeDate"` objects from calendar atoms. You can specify values or vectors of equal length of integers denoting year, month, day, hour, minute and seconds. If every day has the same time stamp, you can just add an offset.

Examples:

```
# Monthly calendar for Current Year
getRmetricsOptions("currentYear")
timeCalendar()

# Daily 'timeDate' for January data from Tokyo local time 16:00
timeCalendar(2008, m=1, d=1:31, h=16, zone="Tokyo", FinCenter="Zurich")

# Or add16 hours in seconds ...
timeCalendar(2008, m=1, d=1:31, zone="Tokyo", FinCenter="Zurich") + 16*3600
```

2. Operations on "timeDate" Objects

Many operations can be performed on "timeDate" objects. You can add and subtract, round and truncate, subset, coerce or transform them to other objects. These are only few options among many others.

Math Operations

Math operations can add and subtract dates and times, and perform logical operations on "timeDate" objects.

Examples:

```
# Date and Time Now:
now = Sys.timeDate()

# One Hour Later:
now + 3600

# Which date/time is earlier or later ?
tC = timeCalendar()
tR = tC + round(3600*rnorm(12))
tR > tC
```

Lagging

You can generate suitable lagged and iterated differences:

`diff.timeDate` – Returns suitably lagged and iterated differences.

Examples:

```
# Monthly Dates 2008 and January 2009:
tC = c(timeCalendar(2008), timeCalendar(2009)[1])

# Number of days in months and total 2008:
diff(tC)
```

```
sum(as.integer(diff(tC)))
```

Rounding and Truncating

Dates and times can be rounded or truncated. This is useful lower frequencies than seconds, for example hourly.

round – rounds objects of class "timeDate",
trunc – truncates objects of class "timeDate".

Examples:

```
# Round the Random Time Stamps to the Nearest Hour:
tC = timeCalendar()
tR = tC + round(3600*rnorm(12))
tR
round(tR, "h")

# Truncate by Hour or to the Next Full Hour::
trunc(tR, "h")
trunc(tR + 3600, "h")
```

Subsetting

Subsetting a "timeDate" is a very important issue in the management of dates and times. Rmetrics offers several functions which are useful in this context:

"[" – Extracts or replaces subsets from "timeDate" objects,
window, cut – extract a piece from a "timeDate" object,

In this context it is also important to know the start and the end time stamp together with the total number of time stamps.

start – extracts the first entry of a "timeDate" object,
end – extracts the last entry of a "timeDate" object,
length – returns the length of a "timeDate" object.

Examples:

```
# Create Monthly Calendar for next year
tC = timeCalendar(getRmetricsOptions("currentYear") + 1)
tC

# Start, end and length of 'timeDate' objects
start(tC)
end(tC)
```

```
length(tC)

# The first Quarter - Several Alternative Solutions:
tC[1:3]
tC[-(4:length(tC))]
window(tC, start = tC[1], end = tC[3])
cut(tC, from = tC[1], to = tC[3])
tC[tC < tC[4]]

# The Quarterly Series:
tC[seq(3, 12, by = 3)]
```

Weekdays, weekends, business days, and holidays can be easily obtained with the following functions:

`isWeekday` – tests if a date is a weekday or not,
`isWeekend` – tests if a date is a weekend day or not,
`isBizday` – tests if a date is a business day or not,
`isHoliday` – tests if a date is a holiday day or not.

Examples:

```
# A 'timeDate' Sequence around Easter 2008
Easter(2008)
tS <- timeSequence(Easter(2008, -14), Easter(2008, +14))
tS

# Subset weekdays and business days:
tW <- tS[isWeekday(tS)]; tW
dayOfWeek(tW)
tB <- tS[isBizday(tS, holidayZURICH())]; tB
dayOfWeek(tB)
```

The functions `blockStart` and `blockEnd` gives time stamps for equally sized blocks.

`blockStart` – Creates start dates for equally sized blocks,
`blockEnd` – Creates end dates for equally sized blocks.

Examples:

```
# 'timeDate' object for the last 365 days:
tS = timeSequence(length.out = 360)
tS

# Subset Pointers for blocks of exactly 30 days:
blockStart(tS, 30)
blockEnd(tS, 30)
Sys.timeDate()
```

Coercions and Transformations

"timeDate" objects are not living in an isolated world. Coercions and transformations allow "timeDate" objects to communicate with other formatted time stamps. Be aware that in most cases information can be lost if the other date.time classes do not support this functionality. There exist several methods to coerce and transform "timeDate" objects into other objects.

as.timeDate – Implements Use Method,
as.timeDate.default – default Method,
as.timeDate.POSIXt – returns a 'POSIX' object as "timeDate" object,
as.timeDate.Date – returns a 'POSIX' object as "timeDate" object.

as.character.timeDate – Returns a "timeDate" object as 'character' string,
as.double.timeDate – returns a "timeDate" object as 'numeric' object,
as.data.frame.timeDate – returns a "timeDate" object as 'data.frame' object,
as.POSIXct.timeDate – returns a "timeDate" object as 'POSIXct' object,
as.POSIXlt.timeDate – returns a "timeDate" object as 'POSIXlt' object,
as.Date.timeDate – returns a "timeDate" object as 'Date' object.

Users or maintainers of other date/time classes can add their own generic functions. For example as.timeDate.zoo and as.zoo.timeDate.

Concatenations and Reorderings

It might be sometimes useful to concatenate or reorder "timeDate" objects. The generic functions to concatenate, replicate, sort, re-sample, unify and revert a "timeDate" objects are :

c – Concatenates "timeDate" objects,
rep – replicates a "timeDate" object,
sort – sorts a "timeDate" object,
sample – resamples a "timeDate" object,
unique – makes a "timeDate" object unique,
rev – reverts a "timeDate" object.

NOTE: The function c of a "timeDate" objects takes care of possible different financial centers specific to each object to be concatenated. In such cases, all time stamps will be transformed to the financial center of the first time stamp used in the concatenation:

Examples:

```
# Concatenate the local time stamps to Zurich time ...
ZH = timeDate("2008-01-01 16:00:00", zone = "GMT", FinCenter = "Zurich")
NY = timeDate("2008-01-01 18:00:00", zone = "GMT", FinCenter = "NewYork")
c(ZH, NY)
c(NY, ZH)

# Rordering:
tC = timeCalendar(); tC
tS = sample(tC); tS
tO = sort(tS); tO
tV = rev(tO); tV
tU = unique(c(tS, tS)); tU
```

3. Daylight Saving Time and Financial Centers

Each financial center worldwide has a function which returns Daylight Saving Time Rules. Almost 400 prototypes are made available through the Olson time zone data base. The cities and regions can be listed using the command `listFinCenter`. The DST rules for specific financial center can be viewed by their name, e.g. `Zurich()`. Additional financial centers can be added by the user taking care of the format specification of the DST functions.

Setting Financial Centers

All time stamps are handled according to the time zone and daylight saving time rules specified by the center through the variable `myFinCenter`. This variable is set by default to `"GMT"` but can be changed to your local financial center or to any other financial center you want to use.

NOTE: By setting the financial center to a continent/city which lies outside of the time zone used by your computer does not change any time settings or environment variables used by your computer.

To change the name of a financial center from one setting to another just assign to the variable `myFinCenter` the desired name of the city:

Examples:

```
# What is my current Financial Center ?
getRmetricsOptions("myFinCenter")

# Change to Zurich:
setRmetricsOptions(myFinCenter = "Zurich")
getRmetricsOptions("myFinCenter")
```

From now on, all dates and times are handled within the middle European time zone and the DST rules which are valid for Zurich.

List of Financial Centers

There are many other financial centers supported by Rmetrics. They can be displayed by the function `listFinCenter`. You can also display partial lists with wildcards and regular expressions:

Examples:

```
# List all supported Financial Centers Worldwide:
listFinCenter()

# List European Financial Centers:
listFinCenter("Europe/*")
```

DST Rules

For each financial center a function is available. It keeps the information of the time zones and the DST rules. The functions return a data.frame with 4Columns :

```
Zurich offSet isdst TimeZone
...
62  2008-03-30 01:00:00   7200     1     CEST
63  2008-10-26 01:00:00   3600     0      CET
...
```

The first column describes when the time was changed, the second gives the offset to "GMT", the third returns the daylight savings time flag which is positive if in force, zero if not, and negative if unknown. The last column gives the name of the time zone. You can have a look at the function `Zurich()` :

Examples:

```
# Show the DST Rules for Zurich:
Zurich()

# List European Financial Centers:
listFinCenter("Europe/*")
```

3. Holidays and Holiday Calendars

It is non-trivial to implement function for business days, weekends and holidays. It is not difficult in an algorithmic sense, but it can become tedious to implement the rules of the calendar themselves, for example the date of Easter.

In the following section we briefly summarise the functions which can calculate dates of ecclesiastical and public holidays. With the help of these functions we can also create business and holiday calendars.

Special Dates:

The implemented functions can compute the last day in a given month and year, the dates in a month that is a n-day (e.g. n- = Sun) on or after a given date, the dates in a month that is a n-day on or

before a specified date, the n-th occurrences of a n-day for a specified year/month vectors, or the last n-day for a specified year/month value or vector.

NOTE: n-days are numbered from 0 to 6 where 0 correspond to the Sunday and 6 to the Saturday.

`timeFirstDayInMonth` – Computes the first day in a given month and year,
`timeLastDayInMonth` – Computes the last day in a given month and year,
`timeFirstDayInQuarter` – Computes the first day in a given quarter and year,
`timeLastDayInQuarter` – Computes the last day in a given quarter and year,

`timeNdayOnOrAfter` – Computes date that is a "on-or-after" n-day,
`timeNdayOnOrBefore` –b Computes date that is a "on-or-before" n-day,

`timeNthNdayInMonth` – Computes n-th occurrence of a n-day in year/month,
`timeLastNdayInMonth` – Computes the last n-day in year/month.

Holidays:

Holidays may have two origins: ecclesiastical or public/federal. The ecclesiastical calendars of Christian churches are based on cycles of movable and immovable feasts. Christmas, December 25, is the principal immovable feast. Easter is the principal movable feast, and dates of most of the other movable feasts are determined with respect to Easter. However, the movable feasts of the Advent and Epiphany seasons are Sundays reckoned from Christmas and the Feast of the Epiphany, respectively.

Examples:

```
# List Holidays available in Rmetrics
listHolidays()

# The date of Easter for the next 5 years:
currentYear <- getRmetricsOptions("currentYear")
Easter(currentYear:(currentYear+5))
```

Holiday Calendars:

`holidayZURICH` – Zurich Business Calendar,
`holidayNYSE` – NYSE Stock Exchange Holiday Calendar,
`holidayZURICH` – TSX Holiday Calendar.

We would like to thank all Rmetrics users who gave us many additional information concerning local holidays.

References

Bateman R., (2000); *Time Functionality in the Standard C Library*, Novell AppNotes, September 2000 Issue, 73–85.

Becker R.A., Chambers J.M., Wilks A.R. (1988); *The New S Language*, Wadsworth & Brooks/Cole.

ISO-8601, (1988); *Data Elements and Interchange Formats - Information Interchange, Representation of Dates and Time*, International Organization for Standardization, Reference Number ISO 8601, 14 pages.

James D.A., Pregibon D. (1992), *Chronological Objects for Data Analysis*, Reprint.

Ripley B.D., Hornik K. (2001); *Date-Time Classes*, R-News, Vol. 1/2 June 2001, 8–12.

Zivot, E., Wang J. (2003); *Modeling Financial Time Series with S-Plus*, Springer, New-York.

`.endpoints` *Endpoints indexes*

Description

Returns endpoint indexes from a `"timeDate"` object.

Usage

```
.endpoints(x, on = c("months", "years", "quarters", "weeks", "days",
    "hours", "minutes", "seconds"), k=1)
```

Arguments

`x`	a `"timeDate"` object.
`on`	the periods endpoints to find as a character string. Select from: `"months"`, `"years"`, `"quarters"`, `"weeks"`, `"days"`, `"hours"`, `"minutes"`, `"seconds"`.
`k`	along every k-th element.

Details

`.endpoints` returns an integer vector corresponding to the last observation in each period specified by `on`, with a zero added to the beginning of the vector, and the index of the last observation in `x` at the end.

Value

an integer vector of endpoints beginning with 0 and ending with the value equal to the length of the `x` argument

Author(s)

Jeff Ryan, modified by Diethelm Wuertz for `"timeDate"` objects.

Examples

```
## endpoints

# Weekly Endpoints
.endpoints(timeCalendar(), on="w")
```

align *Align a 'timeDate' object to regular date/time stamps*

Description

Aligns a "timeDate" object to regular date/time stamps.

Usage

```
## S4 method for signature 'timeDate'
align(x, by = "1d", offset = "0s")

alignDaily(x, include.weekends=FALSE)
alignMonthly(x, include.weekends=FALSE)
alignQuarterly(x, include.weekends=FALSE)
```

Arguments

x	an object of class "timeDate".
by	a character string formed from an integer length and a period identifier. Valid values are "w", "d", "h", "m", "s", for weeks, days, hours, minutes and seconds. For example a bi-weekly period is expressed as "2w".
offset	a character string to set an offset formed from an integer length and a period identifier in the same way as for argument by.
include.weekends	logical value indicating whether weekends should be included.

Details

The functions alignDaily, alignMonthly, alignMonthly are simple to use functions which generate end-of-day, end-of-month, and end-of quarter "timeDate" objects. Weekends are excluded by default. Optionally they can be added setting the argument include.weekends = TRUE.

Value

an object of class "timeDate"

Examples

```
## align

# align bi-weekly with a 3 days offset
(tC <- timeCalendar())
align(tC, by = "2w", offset = "3d")

## alignDaily
```

```
# simple to use functions
alignDaily(tC)
alignDaily(tC, include.weekends = TRUE)

# align to end-of-month dates
alignMonthly(tC)
```

`blockStart` *Equally sized 'timeDate' blocks*

Description

Creates start (end) dates for equally sized `"timeDate"` blocks.

Usage

```
blockStart(x, block = 20)
blockEnd(x, block = 20)
```

Arguments

`block`	an integer value specifying the length in number of records for numerically sized blocks of dates.
`x`	an object of class `"timeDate"`.

Details

The functions `blockStart` and `blockEnd` create vectors of start and end values for equally sized `"timeDate"` blocks. Note, the functions are event counters and not a time counter between measuring time intervals between start and end dates! For equally sized blocks in time one has before to align the time stamps in equal time differences.

Value

an object of class `"timeDate"`

Examples

```
## timeSequence
# 360 Days Series:
tS <- timeSequence(to = "2022-09-23 09:39:23", length.out = 360)

## blockStart | blockEnd
Start <- blockStart(tS, 30)
End <- blockEnd(tS, 30)
Start
End
End - Start
```

c *Concatenating 'timeDate' objects*

Description

Concatenates "timeDate" objects.

Usage

```
## S3 method for class 'timeDate'
c(..., recursive = FALSE)
```

Arguments

recursive	a logical. If recursive is set to TRUE, the function recursively descends through lists combining all their elements into a vector.
...	arguments passed to other methods.

Value

an object of class "timeDate"

Examples

```
## timeCalendar
# Create Character Vectors:
GMT = timeCalendar(zone = "GMT", FinCenter = "GMT") + 16*3600
ZUR = timeCalendar(zone = "GMT", FinCenter = "Zurich") + 16*3600

## c
# concatenate and replicate timeDate objects
sort(c(GMT, ZUR))
sort(c(ZUR, GMT))
```

coerceToOther *Coercion from 'timeDate' to other classes*

Description

Coerce and transform objects of class "timeDate".

Usage

```
## S3 method for class 'timeDate'
as.character(x, ...)

## S3 method for class 'timeDate'
as.double(x,
    units = c("auto", "secs", "mins", "hours", "days", "weeks"), ...)

## S3 method for class 'timeDate'
as.data.frame(x, ...)

## S3 method for class 'timeDate'
as.POSIXct(x, tz = "", ...)

## S3 method for class 'timeDate'
as.POSIXlt(x, tz = "", ...)

## S3 method for class 'timeDate'
as.Date(x, method = c("trunc", "round", "next"), ...)
```

Arguments

x	an object of class "timeDate".
units	a character string denoting the date/time units in which the results are desired.
tz	inputs the time zone to POSIX objects, i.e. the time zone, zone, or financial center string, FinCenter, as used by "timeDate" objects.
method	a character string denoting the method how to determine the dates.
...	arguments passed to other methods.

Value

an object from the designated target class

See Also

timeDate and as.timeDate for creation of and conversion to "timeDate" objects

Examples

```
## timeDate
tC = timeCalendar()

## convert 'timeDate' to a character vector
as.character(tC)
```

currentYear *Current year*

Description

A variable containing the current year.

Note

It is not allowed to change this variable.

Examples

```
## currentYear
getRmetricsOptions("currentYear")
```

DaylightSavingTime *Daylight Saving Time Rules*

Description

Functions for about 400 cities and regions which return daylight saving time rules and time zone offsets.

Details

As a selection of these functions:

Adelaide Algiers Amsterdam Anchorage Andorra Athens Auckland Bahrain Bangkok Beirut Belfast Belgrade Berlin Bogota Bratislava Brisbane Brussels Bucharest Budapest BuenosAires Cairo Calcutta Caracas Casablanca Cayman Chicago Copenhagen Darwin Denver Detroit Dubai Dublin Eastern Edmonton Frankfurt Helsinki HongKong Honolulu Indianapolis Istanbul Jakarta Jerusalem Johannesburg Kiev KualaLumpur Kuwait Lagos Lisbon Ljubljana London LosAngeles Luxembourg Madrid Manila Melbourne MexicoCity Monaco Montreal Moscow Nairobi Nassau NewYork Nicosia Oslo Pacific Paris Perth Prague Riga Riyadh Rome Seoul Shanghai Singapore Sofia Stockholm Sydney Taipei Tallinn Tehran Tokyo Tunis Vaduz Vancouver Vienna Vilnius Warsaw Winnipeg Zagreb Zurich, ...

Note

There are currently two synonyms available "Pacific" for Los Angeles and "Eastern" for New York.

Specific time zones (`AST`, `CET`, `CST`, `EET`, `EST`, `MST` and `PST`) are also available.

Note we leave the space in all double named cities like New York or Hong Kong and use an underscore for it.

All the entries are retrieved from the tzdata library which is available under GNU GPL licence.

Examples

```
## DST rules for Zurich
head(Zurich())
tail(Zurich())

## list all available centers
listFinCenter()
```

---

dayOfWeek *Day of the week*

Description

Returns the days of the week of the data in a "timeDate" object.

Usage

```
dayOfWeek(x)
```

Arguments

x	an object of class "timeDate".

Value

a character vector giving the days of the week corresponding to the elements of x. The names are in English, abbreviated to three letters.

See Also

dayOfYear

Examples

```
## timeCalendar
tC <- timeCalendar(2022)

## the days of the year
dayOfWeek(tC)
```

dayOfYear *Day of the year*

Description

Returns the days of the year of the data in a "timeDate" object.

Usage

```
dayOfYear(x)
```

Arguments

x	an object of class "timeDate".

Value

vector of integers representing the number of days since the beginning of the year. For January, 1st it is one.

See Also

dayOfWeek

Examples

```
## timeCalendar
tC <- timeCalendar(2022)

## the days of the year
dayOfYear(tC)
```

diff *Lagged 'timeDate' differences*

Description

Returns suitably lagged and iterated differences.

Usage

```
## S3 method for class 'timeDate'
diff(x, lag = 1, differences = 1, ...)
```

Arguments

x	an object of class "timeDate".
lag	an integer indicating which lag to use.
differences	an integer indicating the order of the difference.
...	arguments passed to other methods.

Value

For the function, diff.timeDate, if x is a vector of length n and differences=1, then the computed result is equal to the successive differences x[(1+lag):n] - x[1:(n-lag)]. If difference is larger than one this algorithm is applied recursively to x. Note that the returned value is a vector which is shorter than x.

Examples

```
## create character vectors
dts = c("1989-09-28", "2001-01-15", "2004-08-30", "1990-02-09")
dts
tms = c(  "23:12:55",   "10:34:02",   "08:30:00",   "11:18:23")
tms

## timeDate
GMT = timeDate(dts, zone = "GMT", FinCenter = "GMT") + 24*3600
GMT
ZUR = timeDate(dts, zone = "GMT", FinCenter = "Europe/Zurich")
ZUR

## diff
# suitably lagged and iterated differences
diff(GMT)
diff(GMT, lag = 2)
diff(GMT, lag = 1, diff = 2)
```

---

difftimeDate *Difference of two 'timeDate' objects*

Description

Returns the difference of two 'timeDate' objects.

Usage

```
difftimeDate(time1, time2,
    units = c("auto", "secs", "mins", "hours", "days", "weeks"))
```

Arguments

time1, time2	two objects objects of class "timeDate".
units	a character string denoting the date/time units in which the results are desired.

Value

`difftimeDate`, takes a difference of two `"timeDate"` objects and returns an object of class `"difftime"` with an attribute indicating the units.

Examples

```
## create character vectors
dts = c("1989-09-28", "2001-01-15", "2004-08-30", "1990-02-09")
dts

## timeDate
GMT = timeDate(dts, zone = "GMT", FinCenter = "GMT")
GMT

## diff
# suitably lagged and iterated differences
difftimeDate(GMT[1:2], GMT[-(1:2)])
```

`earlyCloseNYSE` *Early closings of the New York Stock exchange*

Description

Get dates of early closings of the New York Stock exchange (NYSE).

Usage

```
earlyCloseNYSE(year)
```

Arguments

`year`	a vector of integers representing years (4 digits).

Details

`earlyCloseNYSE` gives the dates and times when NYSE was closed early. Some of these closing are scheduled (e.g. at 1pm on the day before or after a holiday), others are unscheduled.

The information is incomplete, particularly after 2011. For those dates the values are computed using explicitly declared rules or, if not available, ones derived from recent years.

Value

a `"timeDate"` object containing the dates (with closing times) of early closings

Note

The function is somewhat experimental but the type of the result will not change.

Author(s)

Georgi N. Boshnakov

References

https://archive.fo/XecDq

See Also

holidayNYSE for a list of NYSE holidays

Examples

```
earlyCloseNYSE(1990)

earlyCloseNYSE(2022:2024) # early closings
holidayNYSE(2022:2024)    # holidays
## early closings & holidays combined
c(earlyCloseNYSE(2022:2024), holidayNYSE(2022:2024))
```

Easter *Date of Easter*

Description

Returns the date of Easter.

Usage

```
Easter(year = getRmetricsOptions("currentYear"), shift = 0)
```

Arguments

year	an integer value or integer vector for the year(s).
shift	an integer value, the number of days shifted from the Easter date. Negative integers are allowed.

Details

Holidays may have two origins, ecclesiastical and public/federal. The ecclesiastical calendars of Christian churches are based on cycles of moveable and immoveable feasts. Christmas, December 25th, is the principal immoveable feast. Easter is the principal moveable feast, and dates of most other moveable feasts are determined with respect to Easter.

The date of Easter is evaluated by a complex procedure whose detailed explanation goes beyond this description. The reason that the calculation is so complicate is, because the date of Easter is linked to (an inaccurate version of) the Hebrew calendar. But nevertheless a short answer to the

question "When is Easter?" is the following: Easter Sunday is the first Sunday after the first full moon after vernal equinox. For the long answer we refer to Toendering (1998).

The algorithm computes the date of Easter based on the algorithm of Oudin (1940). It is valid for any Gregorian Calendar year.

Value

the date of Easter as an object of class `"timeDate"`

Note

Doesn't have options to compute Eastern Orthodox Easter dates.

Examples

```
## Easter
# current year
Easter()

## From 2001 to 2010:
Easter(2001:2010)
```

---

finCenter *Financial Center of a timeDate object*

Description

Get or set the financial center of a `"timeDate"` object.

Usage

```
## S4 method for signature 'timeDate'
finCenter(x)
## S4 replacement method for signature 'timeDate'
finCenter(x) <- value
```

Arguments

`x`	a `timeSeries` object.
`value`	a character with the location of the financial center named as "continent/city".

Details

`"timeDate"` objects store the time in the GMT time zone. The financial center specifies a location whose local time is to be used to format the object, e.g., for printing.

`finCenter` gives the financial center associated with a 'timeDate' object. The assignment form changes it to the specified value. Both functions are S4 generics. This page describes the methods defined in package 'timeDate'.

See Also

listFinCenter

Examples

```
date <- timeDate("2008-01-01")
finCenter(date) <- "GMT"
date
format(date)

finCenter(date) <- "Zurich"
date
format(date)
```

`firstDay` *First and last days*

Description

Computes the first/last day in a given month/quarter.

Usage

```
timeFirstDayInMonth(charvec, format = "%Y-%m-%d", zone = "",
    FinCenter = "")
timeLastDayInMonth(charvec, format = "%Y-%m-%d", zone = "",
    FinCenter = "")

timeFirstDayInQuarter(charvec, format = "%Y-%m-%d", zone = "",
    FinCenter = "")
timeLastDayInQuarter(charvec, format = "%Y-%m-%d", zone = "",
    FinCenter = "")
```

Arguments

`charvec`	a character vector of dates and times.
`format`	the format specification of the input character vector.
`zone`	the time zone or financial center where the data were recorded.
`FinCenter`	a character with the location of the financial center named as "continent/city".

Details

The functions `timeLastDayInMonth` and `timeLastDayInMonth` return the last or first day, respectively, in a given month and year.

The same functionality for quarterly time horizons is returned by the functions `timeLastDayInQuarter` and `timeLastDayInQuarter`.

Value

an object of class "timeDate"

Examples

```
## date as character string
charvec = "2006-04-16"
myFinCenter = getRmetricsOptions("myFinCenter")

## timeLastDayInMonth
# What date has the last day in a month for a given date?
timeLastDayInMonth(charvec, format = "%Y-%m-%d",
  zone = myFinCenter, FinCenter = myFinCenter)
timeLastDayInMonth(charvec)
timeLastDayInMonth(charvec, FinCenter = "Zurich")

## timeFirstDayInMonth
# What date has the first day in a month for a given date?
timeFirstDayInMonth(charvec)

## timeLastDayInQuarter
# What date has the last day in a quarter for a given date?
timeLastDayInQuarter(charvec)

## timeFirstDayInQuarter
# What date has the first day in a quarter for a given date?
timeFirstDayInQuarter(charvec)

## timeNdayOnOrAfter
# What date has the first Monday on or after March 15, 1986?
timeNdayOnOrAfter("1986-03-15", 1)

## timeNdayOnOrBefore
# What date has Friday on or before April 22, 1977?
timeNdayOnOrBefore("1986-03-15", 5)

## timeNthNdayInMonth
# What date is the second Monday in April 2004?
timeNthNdayInMonth("2004-04-01", 1, 2)

## timeLastNdayInMonth
# What date has the last Tuesday in May, 1996?
timeLastNdayInMonth("1996-05-01", 2)
```

---

format-methods *Format methods*

Description

Formats "timeDate" objects as ISO conform character strings.

Usage

```
## S3 method for class 'timeDate'
format(x, format = "", tz = "", usetz = FALSE, ...)
```

Arguments

format	a character string describing the format.
tz	a timezone specification to be used for the conversion.
usetz	a logical.
x	an object of class "timeDate".
...	arguments passed to other methods.

Value

an ISO conforming formatted character string

See Also

as.character

Examples

```
## timeCalendar
# Time Calendar 16:00
tC = timeCalendar() + 16*3600
tC

## format as ISO character string
format(tC)
```

---

holiday *Holiday dates*

Description

Returns the date of a holiday.

Usage

```
holiday(year = getRmetricsOptions("currentYear"), Holiday = "Easter")
```

Arguments

Holiday	the function name (a character string or unquoted) of an ecclesiastical or public holiday in the G7 countries or Switzerland, see the list below. Can also be a character vector to specify several holidays.
year	an integer value or vector of years, formatted as YYYY.

Details

Easter is the central ecclesiastical holiday. Many other holidays are related to this feast. The function `Easter` computes the dates of Easter and related ecclesiastical holidays for the requested year vector. `holiday` calculates the dates of ecclesiastical or publich holidays in the G7 countries, e.g. `holiday(2003, "GoodFriday")`. Rmetrics contains holiday functions automatically loaded at startup time. The user can add easily additional holiday functions. The information for the holidays is collected from several web pages about holiday calendars. The following ecclesiastical and public `[HOLIDAY]` functions in the G7 countries and Switzerland are available:

Holidays Related to Easter:

Septuagesima, Quinquagesima, AshWednesday, PalmSunday, GoodFriday, EasterSunday, Easter, EasterMonday, RogationSunday, Ascension, Pentecost, PentecostMonday, TrinitySunday CorpusChristi.

Holidays Related to Christmas:

ChristTheKing, Advent1st, Advent1st, Advent3rd, Advent4th, ChristmasEve, ChristmasDay, BoxingDay, NewYearsDay.

Other Ecclestical Feasts:

SolemnityOfMary, Epiphany, PresentationOfLord, Annunciation, TransfigurationOfLord, AssumptionOfMary, AssumptionOfMary, BirthOfVirginMary, CelebrationOfHolyCross, MassOfArchangels, AllSaints, AllSouls.

CHZurich - Public Holidays:

CHBerchtoldsDay, CHSechselaeuten, CHAscension, CHConfederationDay, CHKnabenschiessen.

GBLondon - Public Holidays:

GBEarlyMayBankHoliday, GBSpringBankHoliday GBSummerBankHoliday, GBNewYearsEve.

(The deprecated GBMayDay and GBBankHoliday are still available but strongly discouraged. Instead, use GBEarlyMayBankHoliday and GBSpringBankHoliday, respectively)

DEFrankfurt - Public Holidays:

DEAscension, DECorpusChristi, DEGermanUnity, DEChristmasEve, DENewYearsEve.

FRParis - Public Holidays:

FRFetDeLaVictoire1945, FRAscension, FRBastilleDay, FRAssumptionVirginMary, FRAllSaints, FRArmisticeDay.

ITMilano - Public Holidays:

ITEpiphany, ITLiberationDay, ITRepublicAnniversary, ITAssumptionOfVirginMary, ITAllSaints, ITWWIVictoryAnniversary, ITStAmrose, ITImmaculateConception.

USNewYork/USChicago - Public Holidays:

USNewYearsDay, USInaugurationDay, USMLKingsBirthday, USLincolnsBirthday, USWashingtonsBirthday, USMemorialDay, USIndependenceDay, USLaborDay, USColumbusDay, USElectionDay, USVeteransDay, USThanksgivingDay, USChristmasDay, USCPulaskisBirthday, USGoodFriday, USJuneteenthNationalIndependenceDay.

CAToronto/CAMontreal - Public Holidays:

CAVictoriaDay, CACanadaDay, CACivicProvincialHoliday, CALabourDay, CAThanksgivingDay, CaRemembranceDay.

JPTokyo/JPOsaka - Public Holidays:

JPNewYearsDay, JPGantan, JPBankHolidayJan2, JPBankHolidayJan3, JPComingOfAgeDay, JPSeijinNoHi, JPNatFoundationDay, JPKenkokuKinenNoHi, JPGreeneryDay, JPMidoriNoHi, JPConstitutionDay, JPKenpouKinenBi, JPNationHoliday, JPKokuminNoKyujitu, JPChildrensDay, JPKodomoNoHi, JPMarineDay, JPUmiNoHi, JPRespectForTheAgedDay, JPKeirouNoHi, JPAutumnalEquinox, JPShuubunno-hi, JPHealthandSportsDay, JPTaiikuNoHi, JPNationalCultureDay, JPBunkaNoHi, JPThanksgivingDay, JPKinrouKanshaNohi, JPKinrou-kansha-no-hi, JPEmperorsBirthday, JPTennou-tanjyoubi, JPTennou-tanjyou-bi.
JPMountainDay

Value

an object of class `"timeDate"`

Examples

```
## holiday
# Dates for GoodFriday from 2000 until 2005:
holiday(2000:2005, "GoodFriday")
holiday(2000:2005, GoodFriday) # same (GoodFriday is a function)

# Good Friday and Easter
holiday(2000:2005, c("GoodFriday", "Easter"))
holiday(2000:2005, c(GoodFriday, Easter))

## Easter
Easter(2000:2005)

## GoodFriday
GoodFriday(2000:2005)
Easter(2000:2005, -2)
```

`holidayDate` *Public and ecclesiastical holidays*

Description

A collection of functions giving holiday dates in the G7 countries and Switzerland.

Usage

```
Septuagesima(year = getRmetricsOptions("currentYear"),
             value = "timeDate", na_drop = TRUE, ...)

Quinquagesima(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

AshWednesday(year = getRmetricsOptions("currentYear"),
             value = "timeDate", na_drop = TRUE, ...)

PalmSunday(year = getRmetricsOptions("currentYear"),
           value = "timeDate", na_drop = TRUE, ...)

GoodFriday(year = getRmetricsOptions("currentYear"),
           value = "timeDate", na_drop = TRUE, ...)

EasterSunday(year = getRmetricsOptions("currentYear"),
             value = "timeDate", na_drop = TRUE, ...)

EasterMonday(year = getRmetricsOptions("currentYear"),
             value = "timeDate", na_drop = TRUE, ...)

RogationSunday(year = getRmetricsOptions("currentYear"),
               value = "timeDate", na_drop = TRUE, ...)

Ascension(year = getRmetricsOptions("currentYear"), value = "timeDate",
          na_drop = TRUE, ...)

Pentecost(year = getRmetricsOptions("currentYear"), value = "timeDate",
          na_drop = TRUE, ...)

PentecostMonday(year = getRmetricsOptions("currentYear"),
                value = "timeDate", na_drop = TRUE, ...)

TrinitySunday(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

CorpusChristi(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

ChristTheKing(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

Advent1st(year = getRmetricsOptions("currentYear"), value = "timeDate",
          na_drop = TRUE, ...)

Advent2nd(year = getRmetricsOptions("currentYear"), value = "timeDate",
          na_drop = TRUE, ...)
```

```
Advent3rd(year = getRmetricsOptions("currentYear"), value = "timeDate",
          na_drop = TRUE, ...)

Advent4th(year = getRmetricsOptions("currentYear"), value = "timeDate",
          na_drop = TRUE, ...)

ChristmasEve(year = getRmetricsOptions("currentYear"),
             value = "timeDate", na_drop = TRUE, ...)

ChristmasDay(year = getRmetricsOptions("currentYear"),
             value = "timeDate", na_drop = TRUE, ...)

BoxingDay(year = getRmetricsOptions("currentYear"), value = "timeDate",
          na_drop = TRUE, ...)

NewYearsDay(year = getRmetricsOptions("currentYear"),
            value = "timeDate", na_drop = TRUE, ...)

SolemnityOfMary(year = getRmetricsOptions("currentYear"),
                value = "timeDate", na_drop = TRUE, ...)

Epiphany(year = getRmetricsOptions("currentYear"), value = "timeDate",
         na_drop = TRUE, ...)

PresentationOfLord(year = getRmetricsOptions("currentYear"),
                   value = "timeDate", na_drop = TRUE, ...)

Annunciation(year = getRmetricsOptions("currentYear"),
             value = "timeDate", na_drop = TRUE, ...)

TransfigurationOfLord(year = getRmetricsOptions("currentYear"),
                      value = "timeDate", na_drop = TRUE, ...)

AssumptionOfMary(year = getRmetricsOptions("currentYear"),
                 value = "timeDate", na_drop = TRUE, ...)

BirthOfVirginMary(year = getRmetricsOptions("currentYear"),
                  value = "timeDate", na_drop = TRUE, ...)

CelebrationOfHolyCross(year = getRmetricsOptions("currentYear"),
                       value = "timeDate", na_drop = TRUE, ...)

MassOfArchangels(year = getRmetricsOptions("currentYear"),
                 value = "timeDate", na_drop = TRUE, ...)

AllSaints(year = getRmetricsOptions("currentYear"), value = "timeDate",
          na_drop = TRUE, ...)
```

```
AllSouls(year = getRmetricsOptions("currentYear"), value = "timeDate",
         na_drop = TRUE, ...)

LaborDay(year = getRmetricsOptions("currentYear"), value = "timeDate",
         na_drop = TRUE, ...)

CHBerchtoldsDay(year = getRmetricsOptions("currentYear"),
                value = "timeDate", na_drop = TRUE, ...)

CHSechselaeuten(year = getRmetricsOptions("currentYear"),
                value = "timeDate", na_drop = TRUE, ...)

CHAscension(year = getRmetricsOptions("currentYear"),
            value = "timeDate", na_drop = TRUE, ...)

CHConfederationDay(year = getRmetricsOptions("currentYear"),
                   value = "timeDate", na_drop = TRUE, ...)

CHKnabenschiessen(year = getRmetricsOptions("currentYear"),
                  value = "timeDate", na_drop = TRUE, ...)

GBMayDay(year = getRmetricsOptions("currentYear"), value = "timeDate",
         na_drop = TRUE, ...)

GBEarlyMayBankHoliday(year = getRmetricsOptions("currentYear"),
                      value = "timeDate", na_drop = TRUE, ...)

GBBankHoliday(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

GBSpringBankHoliday(year = getRmetricsOptions("currentYear"),
                    value = "timeDate", na_drop = TRUE, ...)

GBSummerBankHoliday(year = getRmetricsOptions("currentYear"),
                    value = "timeDate", na_drop = TRUE, ...)

GBMilleniumDay(year = getRmetricsOptions("currentYear"),
               value = "timeDate", na_drop = TRUE, ...)

DEAscension(year = getRmetricsOptions("currentYear"),
            value = "timeDate", na_drop = TRUE, ...)

DECorpusChristi(year = getRmetricsOptions("currentYear"),
                value = "timeDate", na_drop = TRUE, ...)

DEGermanUnity(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)
```

```
DEChristmasEve(year = getRmetricsOptions("currentYear"),
               value = "timeDate", na_drop = TRUE, ...)

DENewYearsEve(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

FRFetDeLaVictoire1945(year = getRmetricsOptions("currentYear"),
                      value = "timeDate", na_drop = TRUE, ...)

FRAscension(year = getRmetricsOptions("currentYear"),
            value = "timeDate", na_drop = TRUE, ...)

FRBastilleDay(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

FRAssumptionVirginMary(year = getRmetricsOptions("currentYear"),
                       value = "timeDate", na_drop = TRUE, ...)

FRAllSaints(year = getRmetricsOptions("currentYear"),
            value = "timeDate", na_drop = TRUE, ...)

FRArmisticeDay(year = getRmetricsOptions("currentYear"),
               value = "timeDate", na_drop = TRUE, ...)

ITEpiphany(year = getRmetricsOptions("currentYear"),
           value = "timeDate", na_drop = TRUE, ...)

ITLiberationDay(year = getRmetricsOptions("currentYear"),
                value = "timeDate", na_drop = TRUE, ...)

ITAssumptionOfVirginMary(year = getRmetricsOptions("currentYear"),
                         value = "timeDate", na_drop = TRUE, ...)

ITAllSaints(year = getRmetricsOptions("currentYear"),
            value = "timeDate", na_drop = TRUE, ...)

ITStAmrose(year = getRmetricsOptions("currentYear"),
           value = "timeDate", na_drop = TRUE, ...)

ITImmaculateConception(year = getRmetricsOptions("currentYear"),
                       value = "timeDate", na_drop = TRUE, ...)

USDecorationMemorialDay(year = getRmetricsOptions("currentYear"),
                        value = "timeDate", na_drop = TRUE, ...)

USPresidentsDay(year = getRmetricsOptions("currentYear"),
                value = "timeDate", na_drop = TRUE, ...)
```

```
USNewYearsDay(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

USInaugurationDay(year = getRmetricsOptions("currentYear"),
                  value = "timeDate", na_drop = TRUE, ...)

USMLKingsBirthday(year = getRmetricsOptions("currentYear"),
                  value = "timeDate", na_drop = TRUE, ...)

USLincolnsBirthday(year = getRmetricsOptions("currentYear"),
                   value = "timeDate", na_drop = TRUE, ...)

USWashingtonsBirthday(year = getRmetricsOptions("currentYear"),
                      value = "timeDate", na_drop = TRUE, ...)

USMemorialDay(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

USIndependenceDay(year = getRmetricsOptions("currentYear"),
                  value = "timeDate", na_drop = TRUE, ...)

USLaborDay(year = getRmetricsOptions("currentYear"),
           value = "timeDate", na_drop = TRUE, ...)

USColumbusDay(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

USElectionDay(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

USVeteransDay(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

USThanksgivingDay(year = getRmetricsOptions("currentYear"),
                  value = "timeDate", na_drop = TRUE, ...)

USChristmasDay(year = getRmetricsOptions("currentYear"),
               value = "timeDate", na_drop = TRUE, ...)

USCPulaskisBirthday(year = getRmetricsOptions("currentYear"),
                    value = "timeDate", na_drop = TRUE, ...)

USGoodFriday(year = getRmetricsOptions("currentYear"),
             value = "timeDate", na_drop = TRUE, ...)

USJuneteenthNationalIndependenceDay(
                                    year = getRmetricsOptions("currentYear"),
```

```
                                        value = "timeDate", na_drop = TRUE,
                                        ...)

CAVictoriaDay(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

CACanadaDay(year = getRmetricsOptions("currentYear"),
            value = "timeDate", na_drop = TRUE, ...)

CACivicProvincialHoliday(year = getRmetricsOptions("currentYear"),
                         value = "timeDate", na_drop = TRUE, ...)

CALabourDay(year = getRmetricsOptions("currentYear"),
            value = "timeDate", na_drop = TRUE, ...)

CAThanksgivingDay(year = getRmetricsOptions("currentYear"),
                  value = "timeDate", na_drop = TRUE, ...)

CaRemembranceDay(year = getRmetricsOptions("currentYear"),
                 value = "timeDate", na_drop = TRUE, ...)

JPVernalEquinox(year = getRmetricsOptions("currentYear"),
                value = "timeDate", na_drop = TRUE, ...)

JPNewYearsDay(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

JPGantan(year = getRmetricsOptions("currentYear"), value = "timeDate",
         na_drop = TRUE, ...)

JPBankHolidayJan2(year = getRmetricsOptions("currentYear"),
                  value = "timeDate", na_drop = TRUE, ...)

JPBankHolidayJan3(year = getRmetricsOptions("currentYear"),
                  value = "timeDate", na_drop = TRUE, ...)

JPComingOfAgeDay(year = getRmetricsOptions("currentYear"),
                 value = "timeDate", na_drop = TRUE, ...)

JPSeijinNoHi(year = getRmetricsOptions("currentYear"),
             value = "timeDate", na_drop = TRUE, ...)

JPNatFoundationDay(year = getRmetricsOptions("currentYear"),
                   value = "timeDate", na_drop = TRUE, ...)

JPKenkokuKinenNoHi(year = getRmetricsOptions("currentYear"),
                   value = "timeDate", na_drop = TRUE, ...)
```

```
JPGreeneryDay(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

JPMidoriNoHi(year = getRmetricsOptions("currentYear"),
             value = "timeDate", na_drop = TRUE, ...)

JPConstitutionDay(year = getRmetricsOptions("currentYear"),
                  value = "timeDate", na_drop = TRUE, ...)

JPKenpouKinenBi(year = getRmetricsOptions("currentYear"),
                value = "timeDate", na_drop = TRUE, ...)

JPNationHoliday(year = getRmetricsOptions("currentYear"),
                value = "timeDate", na_drop = TRUE, ...)

JPKokuminNoKyujitu(year = getRmetricsOptions("currentYear"),
                   value = "timeDate", na_drop = TRUE, ...)

JPChildrensDay(year = getRmetricsOptions("currentYear"),
               value = "timeDate", na_drop = TRUE, ...)

JPKodomoNoHi(year = getRmetricsOptions("currentYear"),
             value = "timeDate", na_drop = TRUE, ...)

JPMarineDay(year = getRmetricsOptions("currentYear"),
            value = "timeDate", na_drop = TRUE, ...)

JPUmiNoHi(year = getRmetricsOptions("currentYear"), value = "timeDate",
          na_drop = TRUE, ...)

JPRespectForTheAgedDay(year = getRmetricsOptions("currentYear"),
                       value = "timeDate", na_drop = TRUE, ...)

JPKeirouNOhi(year = getRmetricsOptions("currentYear"),
             value = "timeDate", na_drop = TRUE, ...)

JPMountainDay(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

JPAutumnalEquinox(year = getRmetricsOptions("currentYear"),
                  value = "timeDate", na_drop = TRUE, ...)

JPShuubunNoHi(year = getRmetricsOptions("currentYear"),
              value = "timeDate", na_drop = TRUE, ...)

JPHealthandSportsDay(year = getRmetricsOptions("currentYear"),
                     value = "timeDate", na_drop = TRUE, ...)
```

```
JPTaiikuNoHi(year = getRmetricsOptions("currentYear"),
             value = "timeDate", na_drop = TRUE, ...)

JPNationalCultureDay(year = getRmetricsOptions("currentYear"),
                     value = "timeDate", na_drop = TRUE, ...)

JPBunkaNoHi(year = getRmetricsOptions("currentYear"),
            value = "timeDate", na_drop = TRUE, ...)

JPThanksgivingDay(year = getRmetricsOptions("currentYear"),
                  value = "timeDate", na_drop = TRUE, ...)

JPKinrouKanshaNoHi(year = getRmetricsOptions("currentYear"),
                   value = "timeDate", na_drop = TRUE, ...)

JPEmperorsBirthday(year = getRmetricsOptions("currentYear"),
                   value = "timeDate", na_drop = TRUE, ...)

JPTennouTanjyouBi(year = getRmetricsOptions("currentYear"),
                  value = "timeDate", na_drop = TRUE, ...)

JPBankHolidayDec31(year = getRmetricsOptions("currentYear"),
                   value = "timeDate", na_drop = TRUE, ...)

InternationalWomensDay(year = getRmetricsOptions("currentYear"),
                       value = "timeDate", na_drop = TRUE, ...)
```

Arguments

year	an integer value or vector of year numbers including the century. These are integers of the form CCYY, e.g. 2000.
value	the class of the returned value. If "timeDate", the default, return a "timeDate" object, otherwise return a character vector.
na_drop	how to treat NAs, TRUE, FALSE or a character string, see section 'Details'.
...	further arguments for format.timeDate, most notably "format". Ignored if value = "timeDate".

Details

This help page discusses the public and ecclesiastical holidays per se. Some holidays fall by definition on a working day or a particular day of the week. For holidays that fall on weekends many countries have rules to declare a close by weekday a holiday. The functions here do not consider such issues but they are handled by the holidayXXX functions (e.g., holidayLONDON), see their help pages.

Public holidays change over time as new ones are introduced, dropped or move to different days. When a holiday date is requested for a year when it did not exist, what should be returned? The same question arises when the information is not available in this package.

The ecclesiastical holidays are computed by traditional rules and in practice should be correct for all years.

Traditionally, package **timeDate** was computing the dates of the holidays according to the current rules. In versions of package **timeDate** greater than 4022.108 historical information was added for England and Japan holidays. The updated functions return the dates according to the rules for the particular years.

For future years the returned dates are always computed accoding to the current rules.

For years before the first available rules, the default is to use those rules, whether the holiday existed or not.

Argument `na_drop` can be used to control this. If `na_drop` is `TRUE`, an entry will not be incuded in the result at all. If `na_drop` is `FALSE` the value for years when the holiday didn't exist will be `NA`. If it is a character string, the default, the closest available rules will be used.

Not all functions respect argument `na_drop`. In that case they act as if `na_drop` is a character string.

The deprecated `GBMayDay` and `GBBankHoliday` are still available but strongly discouraged. Instead, use `GBEarlyMayBankHoliday` and `GBSpringBankHoliday`, respectively.

Value

the date of the requested holiday as a `"timeDate"` object

Note

The holiday information for most countries is incomplete. Contributions are welcome. Please include references for your sources, whenever possible.

Examples

```
## CHSechselaeuten -
# Sechselaeuten a half Day Bank Holiday in Switzerland
CHSechselaeuten(2000:2010)
CHSechselaeuten(getRmetricsOptions("currentYear"))

## German Unification Day:
DEGermanUnity(getRmetricsOptions("currentYear"))
```

`holidayLONDON` *London Bank Holidays*

Description

Returns bank holidays in London.

Usage

```
holidayLONDON(year = getRmetricsOptions("currentYear"))
```

Arguments

year	an integer value or vector of years, formatted as YYYY.

Details

There are currently 8 bank holidays in Britain every year: New Year's Day, Good Friday, Easter Monday, Early Spring Holiday (first Monday of May), Spring Holiday (Last Monday of May), Summer Holiday (Last Monday of August), Christmas Day and Boxing Day.

Some of these holidays are referred also by alternative names or may have had other names in the past. Also the rules according to which the dates for some of them are calculated have changed over time.

Occasionaly there are one-off special holidays, usually related to significant Royal events. Also as one-off, the dates of some holidays are sometimes moved. For example, the Early spring holiday was moved several times to 8th May to coincide with Victory day on big anniversaries.

Value

an object of class "timeDate".

Author(s)

Original function contributed by Menon Murali; amended, corrected and rewritten by Georgi N. Boshnakov

Examples

```
## holidayLONDON
holidayLONDON()
holidayLONDON(2008:2010)
```

---

holidayNERC *NERC holiday calendar*

Description

Returns a holiday calendar for NERC, the North American Reliability Council.

Usage

```
holidayNERC(year = getRmetricsOptions("currentYear"), FinCenter = "Eastern")
```

Arguments

year	an integer value or vector of years, formatted as YYYY.
FinCenter	a character value, the name of the financial center to use.

Value

an object of class "timeDate"

Author(s)

Joe W. Byers

References

http://www.nerc.com/~oc/offpeaks.html

Examples

```
## holidayNERC
holidayNERC()
holidayNERC(2008:2010)
```

---

holidayNYSE *NYSE holiday calendar*

Description

Returns a holiday (closing days) calendar for the New York Stock Exchange.

Usage

```
holidayNYSE(year = getRmetricsOptions("currentYear"),
            type = c("", "standard", "special"))
```

Arguments

year	an integer value or vector of years, formatted as YYYY.
type	what to include, a character string. The default is to return all closing days (holidays and specials). "standard" requests only closings associated with the standard public holidays, "special" gives the special closings only.

Details

holidayNYSE generates a list of the clossing days of the exchange for the requested years.

The default is to return all closing days (holidays and specials). type = "standard" requests only closings associated with the standard public holidays, type = "special" gives the special closings only.

Value

an object of class "timeDate"

Note

The list of closing days returned by `holidayNYSE` was changed in **timeDate** version 4021.105, in that previously it did not include special closing days. This was perceived by some users as buggy. Also, the intent by the authors of the package seems to have been for it to return all closing days. Indeed, the default for `isisBizday()` is to drop weekends and days returned by `holidayNYSE`.

Argument `type` was also included in version 4021.105. The old behaviour can be obtained by using `type = "standard"`.

The default for argument `type` is currently the empty string, since I couldn't come up with another string that would be universally easy to remember. Suggestions are welcome but a change will be only feasible if they come soon.

Author(s)

Diethelm Wuertz (original author); Yohan Chalabi improved speed and handling of time zone; Georgi N. Boshnakov added the special closings and argument 'type'.

See Also

`earlyCloseNYSE` for times of early closings

Examples

```
## holidayNYSE
holidayNYSE() ## current year
holidayNYSE(2008:2010)

## January 2, 2007 was a memorial day for president G.R. Ford,
##    not a regular public holiday
holidayNYSE(2007)
holidayNYSE(2007, type = "standard")
holidayNYSE(2007, type = "special")
```

`holidayTSX` *TSX holiday calendar*

Description

Returns a holiday calendar for the Toronto Stock Exchange.

Usage

```
holidayTSX(year = getRmetricsOptions("currentYear"))
```

Arguments

`year`	an integer value or vector of years, formatted as `YYYY`.

Value

an object of class "timeDate"

Examples

```
## holidayTSX
holidayTSX()
holidayTSX(2008:2010)
```

---

holidayZURICH *Zurich holiday calendar*

Description

Returns a holiday calendar for Zurich.

Usage

```
holidayZURICH(year = getRmetricsOptions("currentYear"))
```

Arguments

year	an integer value or vector of years, formatted as YYYY.

Details

The Zurich holiday calendar includes the following holidays: NewYearsDay, GoodFriday, EasterMonday, LaborDay, PentecostMonday, ChristmasDay, BoxingDay, CHBerchtoldsDay, CHSechselaeuten, CHAscension, CHConfederationDay, CHKnabenschiessen.

Value

an object of class "timeDate"

Examples

```
## holidayZURICH
holidayZURICH()
holidayZURICH(2008:2010)
```

is.na-methods *Methods for 'is.na'*

Description

is.na methods for "timeDate" objects.

Examples

```
# create a timeCalendar sequence
(td <- timeCalendar())
is.na(td)

# insert NA's
is.na(td) <- 2:3
td

# test of NA's
is.na(td)
```

isBizday *Check if dates are business or holidays*

Description

Tests if a date is a business day or not.

Usage

```
isBizday(x, holidays = holidayNYSE(), wday = 1:5)
isHoliday(x, holidays = holidayNYSE(), wday = 1:5)
```

Arguments

x	an object of class "timeDate".
holidays	holiday dates from a holiday calendar. An object of class "timeDate".
wday	Specify which days should be considered as weekdays. By default from Mondays to Fridays.

Details

Returns a logical vector of the same length as x indicating if a date is a business day, or a holiday, respectively.

Value

a logical vector of the same length as x

Examples

```
## dates in April, current year
currentYear <- getRmetricsOptions("currentYear")
tS <- timeSequence(from = paste(currentYear, "-03-01", sep = ""),
                   to = paste(currentYear, "-04-30", sep = ""))
tS

## subset business days at NYSE
holidayNYSE()
isBizday(tS, holidayNYSE())
tS[isBizday(tS, holidayNYSE())]
```

`isRegular` *Checks if a date/time vector is regular*

Description

Checks if a date/time vector is regular. i.e. if it is a daily, a monthly, or a quarterly date/time vector. If the date/time vector is regular the frequency can determined calling the function frequency.

Usage

```
## S4 method for signature 'timeDate'
isDaily(x)
## S4 method for signature 'timeDate'
isMonthly(x)
## S4 method for signature 'timeDate'
isQuarterly(x)

## S4 method for signature 'timeDate'
isRegular(x)

## S3 method for class 'timeDate'
frequency(x, ...)
```

Arguments

`x`	an object of class `"timeDate"`.
`...`	arguments to be passed.

Details

A date/time vector is defined as daily if the vector has not more than one date/time stamp per day.

A date/time vector is defined as monthly if the vector has not more than one date/time stamp per month.

A date/time vector is defined as quarterly if the vector has not more than one date/time stamp per quarter.

A monthly date/time vector is also a daily vector, a quarterly date/time vector is also a monthly vector.

A regular date/time vector is either a monthly or a quarterly vector.

NOT yet implemented is the case of weekly vectors.

Value

The `is*` functions return `TRUE` or `FALSE` depending on whether the date/time vector fulfills the condition or not.

The function frequency returns in general 1, for quarterly date/time vectors 4, and for monthly vectors 12.

Examples

```
tC <- timeCalendar(2023)
tC
isRegular(tC)
frequency(tC)

isMonthly(tC)
isQuarterly(tC)
isDaily(tC)
```

---

isWeekday *Weekdays and weekends*

Description

Tests if a date is a weekday or not.

Usage

```
isWeekday(x, wday = 1:5)
isWeekend(x, wday = 1:5)
```

Arguments

`x`	an object of class `"timeDate"`.
`wday`	Specify which days should be considered as weekdays. By default from Mondays to Fridays.

Value

a logical vector indicating if a date is a weekday or a weekend day

Examples

```
## dates in april, current year
currentYear = getRmetricsOptions("currentYear")
tS = timeSequence(
   from = paste(currentYear, "-03-01", sep = ""),
   to = paste(currentYear, "-04-30", sep = ""))
tS

## subset of weekends
isWeekend(tS)
tS[isWeekend(tS)]
```

`julian` *Julian counts and calendar atoms*

Description

Returns Julian day counts, date/time atoms from a `"timeDate"` object, and extracts month atoms from a `"timeDate"` object.

Usage

```
## S3 method for class 'timeDate'
julian(x, origin = timeDate("1970-01-01"),
       units = c("auto", "secs", "mins", "hours", "days", "weeks"),
       zone = NULL, FinCenter = NULL, ...)

## S4 method for signature 'timeDate'
atoms(x, ...)

## S3 method for class 'timeDate'
months(x, abbreviate = FALSE)

## S3 method for class 'timeDate'
weekdays(x, abbreviate = FALSE)

## S3 method for class 'timeDate'
quarters(x, abbreviate)

## S4 method for signature 'timeDate'
x$name
```

Arguments

`x`	an object of class `"timeDate"`.
`origin`	a length-one object inheriting from class `"timeDate"` setting the origin for the julian counter.

units	a character string denoting the date/time units in which the results are desired.
zone	the time zone or financial center where the data were recorded.
FinCenter	a character string with the location of the financial center named as "continent/city".
abbreviate	currently not used.
name	one of year, month, day, hour, minute, second, wday (or weekday), wday0 (or weekday0), and quarter. Completion is available in interactive sessions.
...	arguments passed to other methods.

Details

Generic functions to extract properties of "timeDate" objects. julian and months are generics from base R, while atoms is a generic defined in this package.

julian extracts the number of days since origin (can be fractional), see also julian.

atoms extracts the calendar atoms from a "timeDate" object, i.e., the year, month, day, and optionally, hour, minute and second. The result is a data frame with the financial center in atrribute "control".

months extracts the months, see section 'Note'.

The dollar operator applied to a "timeDate" object, e.g. td$name, extracts a component of the date/time values as a numeric vector. Currently, name can be one of year, month, day, hour, minute, second, wday (or weekday), wday0 (or weekday0), and quarter. wday0 (weekday0) starts with 0 (for Sunday), the meaning of the rest should be clear.

In interactive sessions, completion is available for the dollar operator.

Value

for julian, a difftime object;

for atoms, a data.frame with attribute "control" containing the financial center of the input vector x. The data frame has the following components:

Y	year,
m	month,
d	day,
H	hour,
M	minute,
S	second;

for months, a numeric vector with attribute "control" containing the financial center. (**Note:** this use is deprecated, use $month instead.)

for the dollar method, the corresponding component as numeric vector.

Note

Deprecation Warning: a ‘timeDate‘ method for ‘months‘ has existed for a long time but it was returning a numeric vector, which is inconsistent with the other methods for months in base R (they return names of months). Returning a numeric vector when ’abbreviate’ is missing is a temporary compromise, to avoid breaking old code but this should be considered deprecated. Use ‘td$month‘ to get the numbers.

See Also

dayOfWeek, dayOfYear;

the base R functions julian, difftime, months;

Examples

```
## julian
tC = timeCalendar(2022)
julian(tC)[1:3]

## atoms
atoms(tC)

## months
months(tC)

weekdays(tC)
weekdays(tC, TRUE)

## the dollar method
tC$year
tC$month
tC$day
tC$hour
tC$minute
tC$second
tC$weekday
tC$weekday0

tC$quarter
```

---

kurtosis *Kurtosis*

Description

Generic function for computation of kurtosis. The methods defined in package **timeDate** are described here.

Usage

```
kurtosis(x, ...)

## Default S3 method:
kurtosis(x, na.rm = FALSE,
                  method = c("excess", "moment", "fisher"), ...)

## S3 method for class 'data.frame'
kurtosis(x, na.rm = FALSE,
                  method = c("excess", "moment", "fisher"), ...)

## S3 method for class 'POSIXct'
kurtosis(x, ...)

## S3 method for class 'POSIXlt'
kurtosis(x, ...)
```

Arguments

x	a numeric vector or object.
na.rm	a logical. Should missing values be removed?
method	a character string, the method of computation, see section 'Details'.
...	arguments to be passed.

Details

kurtosis is an S3 generic function. This page describes the methods defined in package dateTime.

Argument "method" can be one of "moment", "fisher", or "excess". If "excess" is selected, then the value of the kurtosis is computed by the "moment" method and a value of 3 will be subtracted. The "moment" method is based on the definitions of kurtosis for distributions and this method should be used when resampling (bootstrap or jackknife). The "fisher" method correspond to the usual "unbiased" definition of sample variance, although in the case of kurtosis exact unbiasedness is not possible.

If x is numeric the kurtosis is computed according to the description given for argument method. A logical vector is treated as a vector of 1's and 0's.

The data.frame method applies kurtosis recursively to each column. The POSIXlt method computes the kurtosis of the underlying numerical representation of the date/times. The method for POSIXct does the same after converting the argument to POSIXlt.

The default method returns NA, with a warning, if it can't handle argument x.

Value

a numeric value or vector with attribute "method" indicating the method.

See Also

skewness

Examples

```
r = rnorm(100)
mean(r)
var(r)

## kurtosis
kurtosis(r)

kurtosis(data.frame(r = r, r2 = r^2))
```

`length` *Length of a 'timeDate' object*

Description

Returns the length of a "timeDate" object.

Usage

```
## S3 method for class 'timeDate'
length(x)
```

Arguments

x	an object of class "timeDate".

Value

an integer of length 1

Examples

```
## timeCalendar
tC = timeCalendar()

## length -
length(tC)
```

`listFinCenter` *List of financial centers*

Description

Lists supported financial centers.

Usage

```
listFinCenter(pattern = ".*")
```

Arguments

`pattern`	a pattern character string as required by the `grep` function. The default, `".*"`, gives all supported financial centers

Details

The list returned by `listFinCenter` doesn't contain all financial centers supported by **timeDate**. Rather it contains currently supported 'standard names' of time zones defined in the tz (a.k.a. Zoneinfo) database. Names supported by previous versions of by **timeDate** are recognised, even though they are not in the list.

Value

a character vector listing the financial centers whose names match `pattern`.

See Also

`rulesFinCenter` for the daylight saving rules

Examples

```
## myFinCenter - the global setting currently used
getRmetricsOptions("myFinCenter")

## Other Financial Centers
listFinCenter("Asia/")
listFinCenter("^A")    # all beginning with "A"
listFinCenter("^[^A]") # all *not* beginning with "A"
listFinCenter(".*/L")  # cities with L*

stopifnot(identical(sort(listFinCenter()), ## 'A' and 'not A' == everything:
     sort(union(listFinCenter("^A"),
         listFinCenter("^[^A]")))))
```

listHolidays *List of holidays*

Description

Returns a list of holidays.

Usage

```
listHolidays(pattern = ".*")
```

Arguments

pattern	a character string containing a regular expression.

Details

Returns a character vector containing the names of supported holidays matching `pattern`. The default is to return all holidays.

The list is sorted alphabetically. It is changed from time to time. So, the use of character indexing (possibly representing patterns) on the returned list is strongly recommended.

Value

a character vector

Examples

```
## Local Swiss Holidays:
listHolidays("CH")

listHolidays("Easter")
listHolidays("NewYear")

## All Holidays
listHolidays()
```

midnightStandard *Midnight standard*

Description

Corrects `"timeDate"` objects if they do not fulfill the ISO8601 midnight standard.

`midnightStandard2()` relies on `strptime` wherever possible, and there simply returns `as.POSIXct(strptime(charvec, format, tz = "GMT"))`.

Usage

```
midnightStandard (charvec, format)
midnightStandard2(charvec, format)
```

Arguments

charvec	a character string or vector of dates and times.
format	a string, the format specification of the input character vector.

Value

midnightStandard returns a character and midnightStandard2 a POSIXct object.

Examples

```
ch <- "2007-12-31 24:00"
midnightStandard(ch)
(ms2 <- midnightStandard2(ch))
class(ms2)
```

---

myFinCenter *myFinCenter variable*

Description

A character string with the name of my financial center.

Note

Can be modified by the user to his/her own or any other financial center. The default is "GMT". To list all supported financial centers use the function listFinCenter.

See Also

listFinCenter

Examples

```
## myFinCenter - the global setting currently used
getRmetricsOptions("myFinCenter")

## change to another financial center
# setRmetricsOptions(myFinCenter = "Zurich")

## Do not care about DST
# setRmetricsOptions(myFinCenter = "GMT")
```

myUnits *Frequency of date/time units*

Description

A variable with the frequency of date/units.

Value

the date/time units, a character value, yy default "days"

Examples

```
## myUnits
getRmetricsOptions("myUnits")
```

names-methods *The names of a 'timeDate' object*

Description

Functions to get or set the names of a "timeDate" object.

Usage

```
## S4 method for signature 'timeDate'
names(x)
## S4 replacement method for signature 'timeDate'
names(x) <- value
```

Arguments

x	an object of class "timeDate".
value	a character vector of up to the same length as x, or NULL.

Examples

```
td <- timeCalendar()
td
names(td) <- LETTERS[seq_along(td)]
td
```

nDay *n-th n-day dates*

Description

Computes the date for the n-th or last occurrence of an n-day in year/month.

Usage

```
timeNthNdayInMonth(charvec, nday = 1, nth = 1, format = "%Y-%m-%d",
    zone = "", FinCenter = "")

timeLastNdayInMonth(charvec, nday = 1, format = "%Y-%m-%d",
    zone = "", FinCenter = "")
```

Arguments

charvec	a character vector of dates and times.
nday	an integer vector with entries ranging from 0 (Sunday) to 6 (Saturday).
nth	an integer vector numbering the n-th occurence.
format	the format specification of the input character vector.
zone	the time zone or financial center where the data were recorded.
FinCenter	a character with the location of the financial center named as "continent/city".

Details

timeNthNdayInMonth returns the nth occurrence of a n-day (nth = 1,...,5) in year, month.

timeLastNdayInMonth returns the last nday in year, month.

Value

an object of class "timeDate"

Examples

```
## timeNthNdayInMonth
# What date is the second Monday in April 2004?
timeNthNdayInMonth("2004-04-01", 1, 2)

## timeLastNdayInMonth
# What date has the last Tuesday in May, 1996?
timeLastNdayInMonth("1996-05-01", 2)
```

onOrAfter *On-or-after/before dates*

Description

Compute the date that is a "on-or-after" or "on-or-before" n-day.

Usage

```
timeNdayOnOrAfter(charvec, nday = 1, format = "%Y-%m-%d",
    zone = "", FinCenter = "")

timeNdayOnOrBefore(charvec, nday = 1, format = "%Y-%m-%d",
    zone = "", FinCenter = "")
```

Arguments

charvec	a character vector of dates and times.
nday	an integer vector with entries ranging from 0 (Sunday) to 6 (Saturday).
format	the format specification of the input character vector.
zone	the time zone or financial center where the data were recorded.
FinCenter	a character with the location of the financial center named as "continent/city".

Details

timeNdayOnOrAfter returns the date in the specified month that is a n-day (e.g. Sunday) on or after the given date. Month and date are given through argument charvec.

The function timeNdayOnOrBefore returns the date that is a n-day on or before the given date.

Value

an object of class "timeDate"

Examples

```
## date as character string
charvec = "2006-04-16"

## timeNdayOnOrAfter
# What date has the first Monday on or after March 15, 1986?
timeNdayOnOrAfter("1986-03-15", 1)

## timeNdayOnOrBefore
# What date has Friday on or before April 22, 1977?
timeNdayOnOrBefore("1986-03-15", 5)
```

periods *Rolling periods*

Description

Returns start and end dates for rolling periods.

Usage

```
periods(x, period = "12m", by = "1m", offset = "0d")
periodicallyRolling(x, period = "52w", by = "4w", offset = "0d")
monthlyRolling(x, period = "12m", by = "1m")
```

Arguments

x	an object of class timeDate.
period	a span string, consisting of a length integer and a unit value, e.g. "52w" for 52 weeks.
by	a span string, consisting of a length integer and a unit value, e.g. "4w" for 4 weeks.
offset	a span string, consisting of a length integer and a unit value, e.g. "0d" for no offset.

Details

Periodically Rolling - Allowed unit values are "m" for 4 weeks, "w" for weeks, "d" for days, "H" for hours, "M" for minutes, and "S" for seconds.

Monthly Calendar Rolling - The only allowed allowed unit value is "m" for monthly periods. Express a quarterly period by "3m", a semester by "6m", a year by "12m" etc.

Examples

```
## create time sequence
x <- timeSequence(from = "2001-01-01", to = "2009-01-01", by = "day")

## generate periods
periods(x, "12m", "1m")
periods(x, "52w", "4w")

## roll periodically
periodicallyRolling(x)

## roll monthly
monthlyRolling(x)
```

plot-methods *Plot methods*

Description

Plot methods for "timeDate" objects.

Usage

```
## S4 method for signature 'timeDate'
plot(x, y, ...)
## S4 method for signature 'timeDate'
lines(x, y, ...)
## S4 method for signature 'timeDate'
points(x, y, ...)

axis.timeDate(side, x, at, format = NULL, labels = TRUE, ...)

## S3 method for class 'timeDate'
pretty(x, n=5, min.n=n%/%3, shrink.sml=0.75,
        high.u.bias=1.5, u5.bias=0.5+1.5*high.u.bias,
        eps.correct=0, ...)
```

Arguments

x, y, at	an object of class timeDate.
side	an integer specifying which side of the plot the axis is to be drawn on. The axis is placed as follows: 1=below, 2=left, 3=above and 4=right.
format	a POSIX format string, e.g. "%Y-%m-%d".
labels	either a logical value specifying whether annotations are to be made at the tick-marks, or a vector of character strings to be placed at the tickpoints.
n	an integer giving the desired number of intervals.
min.n	a nonnegative integer giving the minimal number of intervals.
shrink.sml	a positive numeric by a which a default scale is shrunk in the case when range(x) is very small.
high.u.bias	a non-negative numeric, typically > 1. Larger high.u.bias values favor larger units.
u5.bias	a non-negative numeric multiplier favoring factor 5 over 2.
eps.correct	an integer code, one of 0, 1, or 2. If non-0, a correction is made at the boundaries.
...	arguments passed to other methods.

Value

returns a summary report of the details of a "timeDate" object. This includes the starting and end date, the number of dates the format and the financial center in use.

Examples

```
## timeCalendar
x <- timeCalendar()
y <- rnorm(12)

## Plotting
plot(x, y, type = "l")
points(x, y, pch = 19, col = "red")

plot(x, y, type = "l", xaxt = "n")
axis.timeDate(1, at = x[c(1, 3, 5, 7, 9, 11)], format = "%b")
axis.timeDate(1, at = x[12], format = "%Y")
```

---

rep *Replicating 'timeDate' objects*

Description

Replicates `"timeDate"` objects.

Usage

```
## S3 method for class 'timeDate'
rep(x, ...)
```

Arguments

`x`	an object of class `"timeDate"`.
`...`	arguments passed to the method for `'POSIXct'`, `rep`.

Value

an object of class `"timeDate"`

Examples

```
## rep
dts = c("1989-09-28", "2001-01-15", "2004-08-30", "1990-02-09")
ZUR = timeDate(dts, zone = "GMT", FinCenter = "Europe/Zurich")

rep(ZUR[2], times = 3)
rep(ZUR[2:3], times = 2)
```

rev *Reverse 'timeDate' objects*

Description

Reverse a "timeDate" object.

Usage

```
## S3 method for class 'timeDate'
rev(x)
```

Arguments

x	an object of class "timeDate".

Value

an object of class "timeDate"

Examples

```
dts <- c("1989-09-28", "2001-01-15", "2004-08-30", "1990-02-09")
ZUR <- timeDate(dts, zone = "GMT", FinCenter = "Europe/Zurich")
ZUR
rev(ZUR)
```

RmetricsOptions *Rmetrics option settings*

Description

Allow the user to set and examine a variety of global options which affect the way in which Rmetrics functions compute and display their results.

Usage

```
setRmetricsOptions(...)
getRmetricsOption(x, unset = "")
```

Arguments

unset	a character string holding the return value is x is not set.
x	a character string holding an option name.
...	any options can be defined, using name = value or by passing a list of such tagged values.

round *Rounding and truncating 'timeDate' objects*

Description

Rounds and truncates objects of class 'timeDate'.

Usage

```
## S3 method for class 'timeDate'
round(x, digits = c("days", "hours", "mins"))

## S3 method for class 'timeDate'
trunc(x, units = c("days", "hours", "mins"), ...)
```

Arguments

digits, units	a character string denoting the date/time units in which the results are desired.
x	an object of class "timeDate".
...	arguments passed to other methods.

Details

The two functions round and trunc allow to round or to truncate "timeDate" objects to the specified unit and return them as "timeDate" objects.

There is an inconsistency in that round uses digits as argument and not units.

Value

an object of class "timeDate"

Examples

```
## round

## truncate
```

rulesFinCenter *Financial centers DST rules*

Description

Returns DST rules for a financial center.

Usage

```
rulesFinCenter(FinCenter = "")
```

Arguments

FinCenter	a character string with the location of the financial center named as "continent/city".

Details

The function rulesFinCenter lists the daylight saving rules for a selected financial center.

There is no dependency on the POSIX implementation of your operating system because **timeDate** comes with a database containing the necessary time zone and day light saving time information.

Value

a list of time zones and DST rules available in the database

See Also

listFinCenter for a list of the available financial centers

Examples

```
## rulesFinCenter
rulesFinCenter("Zurich")
```

sample *Resampling 'timeDate' objects*

Description

Resamples a "timeDate" object.

Value

an object of class "timeDate"

Examples

```
## c
# Create Character Vectors:
dts = c("1989-09-28", "2001-01-15", "2004-08-30", "1990-02-09")
dts
tms = c(  "23:12:55",   "10:34:02",   "08:30:00",   "11:18:23")
tms

## "+/-"
# Add One Day to a Given timeDate Object:
GMT = timeDate(dts, zone = "GMT", FinCenter = "GMT")
GMT
ZUR = timeDate(dts, zone = "GMT", FinCenter = "Europe/Zurich")
ZUR

## c
# Concatenate and Replicate timeDate Objects:
c(GMT[1:2], ZUR[1:2])
c(ZUR[1:2], GMT[1:2])

## rep
rep(ZUR[2], times = 3)
rep(ZUR[2:3], times = 2)
```

---

show-methods *Show methods*

Description

Show methods for "timeDate" objects.

Methods

object = "ANY" Generic function.

object = "timeDate" Print function for objects of class "timeDate".

Examples

```
## print | show
print(timeCalendar())
```

skewness *Skewness*

Description

Functions to compute skewness.

Usage

```
skewness(x, ...)

## Default S3 method:
skewness(x, na.rm = FALSE, method = c("moment", "fisher"), ...)

## S3 method for class 'data.frame'
skewness(x, na.rm = FALSE, method = c("moment", "fisher"), ...)

## S3 method for class 'POSIXct'
skewness(x, ...)

## S3 method for class 'POSIXlt'
skewness(x, ...)
```

Arguments

x	a numeric vector or object.
na.rm	a logical. Should missing values be removed?
method	a character string, the method of computation, see section 'Detaials'.
...	arguments to be passed.

Details

Argument method can be one of "moment" or "fisher". The "moment" method is based on the definitions of skewness for distributions and this should be used when resampling (bootstrap or jackknife). The "fisher" method correspond to the usual "unbiased" definition of sample variance, although in the case of skewness exact unbiasedness is not possible.

Value

a numeric value or vector with attribute "method" indicating the method.

See Also

kurtosis

Examples

```
r = rnorm(100)
mean(r)
var(r)

## skewness
skewness(r)
```

---

sort *Sorting 'timeDate' objects*

Description

Sorts a "timeDate" object.

Usage

```
## S3 method for class 'timeDate'
sort(x, ...)
```

Arguments

x	an object of class "timeDate".
...	arguments passed to other methods.

Value

an object of class "timeDate"

Examples

```
## c
# create character vectors
dts = c("1989-09-28", "2001-01-15", "2004-08-30", "1990-02-09")
dts
tms = c(  "23:12:55",   "10:34:02",   "08:30:00",   "11:18:23")
tms

## "+/-"
# add one day to a given timeDate object:
GMT = timeDate(dts, zone = "GMT", FinCenter = "GMT")
GMT
ZUR = timeDate(dts, zone = "GMT", FinCenter = "Europe/Zurich")
ZUR

## c
# concatenate and replicate timeDate objects
c(GMT[1:2], ZUR[1:2])
c(ZUR[1:2], GMT[1:2])
```

```
## rep
rep(ZUR[2], times = 3)
rep(ZUR[2:3], times = 2)
```

---

specialHolidayGB *Dates of special one-off holidays in the UK*

Description

Gives dates of special one-off holidays in the UK.

Usage

```
specialHolidayGB(year = getRmetricsOptions("currentYear"),
                     value = "timeDate", named = FALSE, ...)
```

Arguments

year	the year(s) for which special holidays are required, a vector containing four-digit integer number(s) of the form CCYY, e.g. 2023.
value	the class of the returned value. If "timeDate", the default, return a "timeDate" object, if "" return a character vector.
named	if TRUE, the dates are named, otherwise unnamed.
...	further arguments for as.character when value = "".

Details

specialHolidayGB gives the special Bank holidays in England for the years specified by argument year, such as the Millenium day at the end of 1999 and significant Royal events. Don't assume that there is at most one special holiday in a given year, 2022 had two.

Years that do not contain special Bank holidays are omitted. If there are no special holidays in the specified year(s) the results is a "timeDate" or "character" object of length zero.

The holidays are sorted in increasing time order.

Argument value controls the class of the result. The default is "timeDate". The result is a character vector if value = "" (the empty string). In the latter case, further arguments for the transformation to character can be passed in argument "..." (e.g., format).

If argument named is TRUE, the dates get names asssociated with them, so one can see which date represents which holiday.

Value

a "timeDate" or a character vector, as requested by argument value

Note

While most of the holidays given by the functions with prefix `GBxxx` are valid for the UK as a whole and they are (or should be) fully correct for England, there are variations in Scotland, Wales and Northern Ireland.

Functions containing 'London' in their name refer to the London Stock Exchange. Currently, the Bank holidays given by those functions are the same as for England. Actually, the 'official' holidays between 1834 and 1870 were set by the Bank of England. The first Act of Parlament on the issue is from 1871.

Author(s)

Georgi N. Boshnakov

See Also

`GBSummerBankHoliday` for functions giving specific regular Bank holidays,

`holidayLONDON` for all London Stock Exchange holidays (actually, England holidays) in requested years.

Examples

```
## UK Millenium day
specialHolidayGB(1999)     # as a dateTime object
specialHolidayGB(1999, "") # as a character string

## 2 special holidays in UK in 2022
specialHolidayGB(2022)     # [2022-06-03] [2022-09-19]
## what are their names?
specialHolidayGB(2022, named = TRUE)

## the Spring BH is usually on last Monday of May, but not in 2022
dayOfWeek(GBSpringBankHoliday(2020:2024))

## the above formed a nice 4-day weekend in early June 2022
## (look at the Thu-Fri sequence on 2-3 June)
dayOfWeek(holidayLONDON(2022))
```

`start` *Terminal times and range*

Description

Extracts the time when the first or last observation was taken, or computes the range of the dates in a `"timeDate"` object.

Usage

```
## S3 method for class 'timeDate'
start(x, ...)

## S3 method for class 'timeDate'
end(x, ...)

## S3 method for class 'timeDate'
min(..., na.rm = FALSE)

## S3 method for class 'timeDate'
max(..., na.rm = FALSE)

## S3 method for class 'timeDate'
range(..., na.rm = FALSE)
```

Arguments

x	an object of class "timeDate".
...	ignored by start and end; a 'timeDate' object for min, max, and range.
na.rm	not used.

Details

Conceptually, the "timeDate" object is sorted before the computations. In particular, start is not necessarilly the first element of the object and similarly for the other functions.

min and max are equivalent to start end end, respectively.

range returns the earlies and the latest times in a "timeDate" object. The remaining functions return only one of them, as suggested by their names.

Value

an object of class "timeDate"

Examples

```
## timeCalendar
# Random Calendar Dates:

tR = sample(timeCalendar())
sort(tR)
tR

## start | end
start(tR)
end(tR)

## the first and last time stamp
tR[1]
```

```
tR[length(tR)]
rev(tR)[1]

## the range
c(start(tR), end(tR))
range(tR)
```

`subset` *Subsetting a 'timeDate' object*

Description

Extracts or replaces subsets from `"timeDate"` objects.

Value

an object of class `"timeDate"`

Examples

```
## timeCalendar
tS = timeCalendar()

## [
# Subsetting Second Quarter:
tS[4:6]

## [<-
# Replacing:
```

`summary-methods` *Summary method*

Description

Summarizes details of a `"timeDate"` object.

Usage

```
## S3 method for class 'timeDate'
summary(object, ...)
```

Arguments

`object`	an object of class `"timeDate"`.
`...`	arguments passed to other methods.

Details

Creates a summary report of the details of a "timeDate" object. This includes the starting and end date, the number of dates the format and the financial center in use.

Value

an object from S3 class "timeDate_summary", which has a print method

Examples

```
tC <- timeCalendar()
summary(tC)
```

Sys.timeDate *System time as 'timeDate' object*

Description

Returns the system time as an object of class "timeDate".

Usage

```
Sys.timeDate(FinCenter = "")
```

Arguments

FinCenter	a character string with the location of the financial center named as "continent/city".

Value

a "timeDate" object

Examples

```
## Not run:
## direct
Sys.timeDate()

## Local Time in Zurich
Sys.timeDate(FinCenter = "Zurich")

## transformed from "POSIX(c)t" with timeDate()
timeDate(Sys.time())

## Local Time in Zurich
timeDate(Sys.time(), FinCenter = "Zurich")

## End(Not run)
```

timeCalendar *'timeDate' from calendar atoms*

Description

Create a "timeDate" object from calendar atoms.

Usage

```
timeCalendar(y = getRmetricsOptions("currentYear"), m = 1:12, d = 1,
    h = 0, min = 0, s = 0,
    zone = "", FinCenter = "")
```

Arguments

y, m, d	calendar years (e.g. 1997), defaults are 1960, calendar months (1-12), defaults are 1, and calendar days (1-31), defaults are 1,
h, min, s	hours of the days (0-23), defaults are 0, minutes of the days (0-59), defaults are 0, and seconds of the days (0-59), defaults are 0.
zone	a character string, denoting the time zone or financial center where the data were recorded.
FinCenter	a character with the location of the financial center named as "continent/city".

Value

an object of class "timeDate"

Examples

```
## timeCalendar

# Current Year:
getRmetricsOptions("currentYear")

# 12 months of current year
timeCalendar()

timeCalendar(m = c(9, 1, 8, 2), d = c(28, 15, 30, 9),
             y = c(1989, 2001, 2004, 1990), FinCenter = "GMT")

timeCalendar(m = c(9, 1, 8, 2), d = c(28, 15, 30, 9),
             y = c(1989, 2001, 2004, 1990), FinCenter = "Europe/Zurich")

timeCalendar(h = c(9, 14), min = c(15, 23))
```

timeDate *Create 'timeDate' objects*

Description

Create a "timeDate" object from scratch from a character vector or other suitable objects.

Usage

```
timeDate(charvec, format = NULL, zone = "", FinCenter = "", ...)

## S4 method for signature 'character'
timeDate(charvec, format = NULL, zone = "", FinCenter = "",
         dst_gap = "+")

## methods for as.timeDate
## Default S3 method:
as.timeDate(x, zone = "", FinCenter = "")

## S3 method for class 'POSIXt'
as.timeDate(x, zone = "", FinCenter = "")

## S3 method for class 'Date'
as.timeDate(x, zone = "", FinCenter = "")

## S3 method for class 'timeDate'
as.timeDate(x, zone = x@FinCenter, FinCenter = "")

strptimeDate(x, format = whichFormat(x), tz = "")
```

Arguments

charvec	a character string or vector of dates and times.
format	the format specification of the input character vector.
zone	the time zone or financial center where the data were recorded.
FinCenter	a character with the location of the financial center named as "continent/city".
dst_gap	a character string specifying what to do with non-existent times falling in a DST gap: add an hour ("+"), subtract an hour ("-"), set to NA ("NA"), or ignore (""). When the 'ignore' option is used the code to check for this kind of faulty times is skipped and the result will be equivalent to "+" or "-" but which one is not defined. This could be useful when you are certain that there are no times in DST gaps or don't care how they are dealt with.
x	for strptimeDate, a character string or vector of dates and times. For the as.timeDate methods, an object from a class that can be converted to "timeDate". The default method converts x to character.

tz	a character with the location of the financial center named as "continent/city", or short "city".
...	further arguments for methods.

Details

timeDate creates objects from class "timeDate" from character vectors, objects from several date/time classes, and other suitable objects.. It is an S4 generic function and this page describes the methods defined in package **timeDate**, see section 'Methods'.

Note that zone is the time zone of the input, while FinCenter is the 'current' time zone, typically but not necessarilly where the code is run. To change one or both of these time zones of an existing "timeDate" object, call timeDate() on it, see the method for charvec = "timeDate" in section 'Methods'.

The methods for as.timeDate call timeDate, maybe after some minor preparation. The default method for as.timeDate converts x to character before calling timeDate.

strptimeDate is a wrapper of timeDate, suitable when zone and FinCenter are the same, It has the same arguments as strptime. If format is missing it tries to deduce it. If tz is missing it sets it to the value of the Rmetrics option "myFinCenter".

Value

an object of class "timeDate"

Methods

The following methods for timeDate are defined in package **timeDate**.

signature(charvec = "ANY") Converts charvec to character and calls timeDate on the result.

signature(charvec = "character") ...

signature(charvec = "Date") ...

signature(charvec = "missing") Returns the current time as "timeDate" object.

signature(charvec = "numeric") ...

signature(charvec = "POSIXt") ...

signature(charvec = "timeDate") Changes the time zone and/or financial center of charvec to the requested ones. If zone is missing or equal to the empty string, just changes the financial center.

See Also

as.character, as.POSIXct, etc., for conversion from "timeDate" to other classes

Examples

```
## timeDate

# character vector strings:
dts = c("1989-09-28", "2001-01-15", "2004-08-30", "1990-02-09")
tms = c(  "23:12:55",   "10:34:02",   "08:30:00",   "11:18:23")

dts; tms

t1 <- timeDate(dts, format = "%Y-%m-%d", FinCenter = "GMT" )
t1

stopifnot(identical(t1, timeDate(dts, FinC = "GMT"))) # auto-format

timeDate(dts, format = "%Y-%m-%d", FinCenter = "Europe/Zurich")

timeDate(paste(dts, tms), format = "%Y-%m-%d %H:%M:%S",
     zone = "GMT", FinCenter = "GMT")

timeDate(paste(dts, tms),
     zone = "Europe/Zurich", FinCenter = "Europe/Zurich")

timeDate(paste(dts, tms), format = "%Y-%m-%d %H:%M:%S",
     zone = "GMT", FinCenter = "Europe/Zurich")

## non standard format:
timeDate(paste(20:31, "03.2005", sep="."), format = "%d.%m.%Y")

## ISO and American formats are auto-detected:
timeDate("2004-12-31", FinCenter = "GMT")
timeDate("12/11/2004", FinCenter = "GMT")
timeDate("1/31/2004") # auto-detect American format

## From POSIX?t, and using NAs
## lsec <- as.POSIXlt(.leap.seconds) ; lsec[c(2,4:6)] <- NA
## timeDate(lsec)

## dtms <- paste(dts,tms)
## dtms[2:3] <- NA
## timeDate(dtms, FinCenter = "Europe/Zurich") # but in GMT

## would need  change in R :
##   tms[3] <- dts[2] <- NA
##   timeDate(paste(dts,tms), FinCenter = "Europe/Zurich") # but in GMT

## Coerce a 'Date' object into a 'timeDate' object:
as.timeDate(Sys.Date())
```

`timeDate-class` *Class "timeDate"*

Description

Class "timeDate" represents date and time objects.

Details

For the management of chronological objects under R three concepts are available: The first is the implementation of date and time in R's chron package neglecting locals, time zones and day light saving times. This approach is in most cases appropriate for economic time series. The second approach, available in R's base package implements the POSIX standard to date and time objects, named "POSIXt".

Unfortunately, the representation of these objects is in some cases operating system dependent and especially under MS Windows several problems appeared over the time in the management of time zones and day light saving times. Rmetrics overcomes these difficulties with POSIX objects and introduce a new S4 class of "timeDate" objects which allow for powerful methods to represent dates and times in different financial centers around the world.

Many of the basic functionalities of these objects are in common with S-Plus' "timeDate" objects and thus many of your privately written functions for SPlus/FinMetrics may also be used within the R/Rmetrics environment.

A major difference is the time zone concept which is replaced by the "Financial Center" concept. The FinCenter character variable specifies where you are living and at which financial center you are working. With the variable myFinCenter you can overwrite the default setting with your personal settings. With the specification of the FinCenter your system knows what rules rules for day light saving times should be applied, what is your holiday calendar, what is your currency, what are your interest rate conventions. (Not all specifications are already implemented.) Many other aspects can be easily accessed when a financial center is named. So we can distinguish between Frankfurt and Zurich, which both belong to the same time zone, but differed in DST changes in the eighties and have different holiday calendars. Futhermore, since the underlying time refers to "GMT" and DST rules and all other information is available in local (ASCII) databases, we are sure, that R/Rmetrics delivers with such a date/time concept on every computer independent of the operating systemin use, identical results.

Another important feature of the "timeDate" concept used here is the fact that we don't rely on American or European ways to write dates. We use consequently the ISO-8601 standard for date and time notations.

Generation of "timeDate" Objects

We have defined a "timeDate" class which is in many aspects similar to the S-Plus class with the same name, but has also some important advantageous differeneces. The S4 class has four Slots, the Data slot which holds date and time as 'POSIXct' objects in the standard ISO-8601 format, the Dim slot which gives the dimension of the data object (i.e. its length), the format specification slot and the FinCenter slot which holds the name of the financial center. By default this is the value

Three functions allow to cgenerate date/time objects: "timeDate" from character vectors, timeCalendar from date and time atoms, and timeSequence from a "from/to" or from a "from/length" sequence specification. Note, time zone transformations are easily handled by by the "timeDate" functions which can also take "timeDate" and POSIXt objects as inputs, while transforming them between financial centers and/or time zones specified by the arguments zone and FinCenter. Finally the

function Sys.timeDate returns current system time in form of a "timeDate" object.

Tests and Representation of timeDate Objects:

Rmetrics has implemented several methods to represent "timeDate" objects. For example, the print method returns the date/time in square "[]" brackets to distinguish the output from other date and time objects. On top of the date and time output the name of the FinCenter is printed. The summary method returns a printed report with information about the "timeDate" object. Finally, the format methods allows to transform objects into a ISO conform formatted character strings.

Mathematical Operations:

Rmetrics supports methods to perform many mathematical operations. Included are methods to extract or to replace subsets from "timeDate" objects, to perform arithmetic "+" and "-" operations, to group Ops generic functions, to return suitably lagged and iterated differences diff, to return differences difftimeDate of two "timeDate" objects, to concatenate objects, to replicate objects, to round objects, to truncate objects using trunc, to extract the first or last entry of a vector, to sort the objects of the elements of a date/time vector, and to revert "timeDate" vector objects, among other functions.

Transformation of Objects:

Rmetrics has also functions to transform dat/time objects between different representations. Included are methods to transform "timeDate" objects to character strings, to data frames, to POSIXct or POSIXlt objects, to julian counts. One can extract date/time atoms from calendar dates, and the months atoms from a "timeDate" object.

Objects from the Class

Objects can be created by calls of the functions timeDate, timeSequence, timeCalendar and as.timeDate, among others. There is also a "timeDate" method for seq.

Slots

Data: Object of class "POSIXct": a vector of POSIXct dates and times always related to "GMT".

format: Object of class "character": a character string denoting the format specification of the input data character vector.

FinCenter: Object of class "character": a character string with the location of the financial center named as "continent/city", or just "city".

Methods

timeDate signature(charvec = "timeDate"): create objects from class "timeDate", see timeDate;

show signature(object = "timeDate"): prints an object of class "timeDate";

plot signature(x = "timeDate"):

points signature(x = "timeDate"):

lines signature(x = "timeDate"):

abline signature(a = "ANY", b = "ANY", h = "ANY", v ="timeDate"): see plot-methods.

[signature(x = "timeDate", i = "ANY", j = "missing"):

[signature(x = "timeDate", i = "character", j = "missing"):

[signature(x = "timeDate", i = "logical", j = "missing"):

[signature(x = "timeDate", i = "missing", j = "missing"):

[signature(x = "timeDate", i = "numeric", j = "missing"): take parts of a "timeDate" object, see subset.

finCenter signature(x = "timeDate"):

finCenter<- signature(x = "timeDate"): see finCenter.

atoms signature(x = "timeDate"):

months signature(x = "timeDate"):

julian signature(x = "timeDate"): see julian.

align signature(x = "timeDate"): see align.

isDaily signature(x = "timeDate"):

isMonthly signature(x = "timeDate"):

isQuarterly signature(x = "timeDate"):

isRegular signature(x = "timeDate"): see see isRegular.

frequency signature(x = "timeDate"): see frequency.

is.na signature(x = "timeDate"): see is.na-methods.

sample signature(x = "timeDate"): see sample.

Ops signature(e1 = "timeDate", e2 = "timeDate"):

+ signature(e1 = "numeric", e2 = "timeDate"):

+ signature(e1 = "timeDate", e2 = "numeric"):

+ signature(e1 = "timeDate", e2 = "timeDate"):

- signature(e1 = "numeric", e2 = "timeDate"):

- signature(e1 = "timeDate", e2 = "numeric"):

- signature(e1 = "timeDate", e2 = "timeDate"): see timeDateMathOps.

coerce signature(from = "ANY", to = "timeDate"):

coerce signature(from = "Date", to = "timeDate"):

coerce signature(from = "POSIXt", to = "timeDate"):

coerce signature(from = "timeDate", to = "character"):

coerce signature(from = "timeDate", to = "data.frame"):

coerce signature(from = "timeDate", to = "Date"):

coerce signature(from = "timeDate", to = "list"):

coerce signature(from = "timeDate", to = "numeric"):

coerce signature(from = "timeDate", to = "POSIXct"):

coerce signature(from = "timeDate", to = "POSIXlt"): convert from/to "timeDate" objects. These are methods for as, to be used with the syntax as(from, to), where from is the object to be converted and to is the desired target class. Most conversions can also be done with specialised functions such as as.character and as.timeDate, see as.timeDate.

names `signature(x = "timeDate")`:

names<- `signature(x = "timeDate")`: see `names-methods`.

getDataPart `signature(object = "timeDate")`: ...

initialize `signature(.Object = "timeDate")`: ...

Note

Originally, these functions were written for Rmetrics users using R and Rmetrics under Microsoft's Windows XP operating system where time zones, daylight saving times and holiday calendars are not or insufficiently supported.

The usage of the Ical Library and the introduction of the FinCenter concept was originally developed for R Version 1.5. The `"timeDate"` and `timeSeries` objects were added for R Version 1.8.1. Minor changes were made to adapt the functions for R Version 1.9.1. As a consequence, newer concepts like the `Date` objects were not yet considered and included in this collection of date and time concepts. With R Version 2.3.0 a major update has been made adding many new generic functions and renaming a few already existing functions, please be aware of this.

Note, the date/time conversion from an arbitrary time zone to GMT cannot be unique, since date/time objects appear twice during the hour when DST changes and the isdt flag was not recorded. A bookkeeping which takes care if DST is effective or not is not yet included. However, in most applications this is not necessary since the markets are closed on weekends, especially at times when DST usually changes. It is planned for the future to implement the DST supporting this facility.

The ISO-8601 midnight standard has been implemented. Note, that for example "2005-01-01 24:00:00" is accepted as a valid date/time string.

Also available is an automated format recognition, so the user does not have to specify the format string for the most common date/time formats.

Examples

```
## Examples for Objects of class 'timeDate'

## timeDate

## Sys.timeDate()           # direct
## timeDate(Sys.time())     # transformed from "POSIX(c)t"

## # Local Time in Zurich
## timeDate(Sys.time(), FinCenter = "Zurich")

# Character Vector Strings:
dts = c("1989-09-28", "2001-01-15", "2004-08-30", "1990-02-09")
dts
tms = c(  "23:12:55",   "10:34:02",   "08:30:00",   "11:18:23")
tms

t1 <- timeDate(dts, format = "%Y-%m-%d", FinCenter = "GMT" )
t1

stopifnot(identical(t1, timeDate(dts, FinC = "GMT"))) # auto-format
```

```
timeDate(dts, format = "%Y-%m-%d", FinCenter = "Europe/Zurich")

timeDate(paste(dts, tms), format = "%Y-%m-%d %H:%M:%S",
  zone = "GMT", FinCenter = "GMT")

timeDate(paste(dts, tms),
  zone = "Europe/Zurich", FinCenter = "Europe/Zurich")

timeDate(paste(dts, tms), format = "%Y-%m-%d %H:%M:%S",
  zone = "GMT", FinCenter = "Europe/Zurich")

## Non Standard Format:
timeDate(paste(20:31, "03.2005", sep="."), format = "%d.%m.%Y")

# Note, ISO and American Formats are Auto-Detected:
timeDate("2004-12-31", FinCenter = "GMT")
timeDate("12/11/2004", FinCenter = "GMT")
timeDate("1/31/2004") # auto-detect American format

## ... from POSIX?t, and Using NAs:
## lsec <- as.POSIXlt(.leap.seconds)
## lsec[c(2,4:6)] <- NA
## timeDate(lsec)

## dtms <- paste(dts,tms)
## dtms[2:3] <- NA
## timeDate(dtms, FinCenter = "Europe/Zurich") # but in GMT


## timeCalendar
## getRmetricsOptions("currentYear")
## timeCalendar()  # 12 months of current year

timeCalendar(2022) # 12 months of 2022
timeCalendar(y = c(1989, 2001, 2004, 1990),
    m = c(9, 1, 8, 2), d = c(28, 15, 30, 9), FinCenter = "GMT")
timeCalendar(y = c(1989, 2001, 2004, 1990),
    m = c(9, 1, 8, 2), d = c(28, 15, 30, 9), FinCenter = "Europe/Zurich")

## timeCalendar(h = c(9, 14), min = c(15, 23))
timeCalendar(2022, h = c(9, 14), min = c(15, 23))

## timeSequence
timeSequence(from = "2004-03-12", to = "2004-04-11",
  format = "%Y-%m-%d", FinCenter = "GMT")
timeSequence(from = "2004-03-12", to = "2004-04-11",
  format = "%Y-%m-%d", FinCenter = "Europe/Zurich")

## print, summary, format
tC = timeCalendar(2022)
tC
print(tC)
```

```
summary(tC)
format(tC)
```

---

timeDateMathOps *Mathematical operations with 'timeDate' objects*

Description

Functions for mathematical and logical operations on "timeDate" objects.

Usage

```
## S4 method for signature 'timeDate,timeDate'
Ops(e1, e2)
```

Arguments

e1, e2	objects of class "timeDate". In the case of addition and subtraction one of them may be of class numeric, specifying the number of seconds to add or subtract.

Details

Group "Ops" represents the binary mathematical operators. Methods are defined for such operations when one or both arguments are from class "timeDate".

Operations that don't make sense, such as addition of two "timeDate" objects, throw error.

The plus operator "+" performs arithmetic "+" operation on "timeDate" objects,

and the minus operator "-" returns a difftime object if both arguments e1 and e2 are "timeDate" objects, or returns a "timeDate" object e2 seconds earlier than e1.

Value

addition of numeric to "timeDate" returns "timeDate",

subtraction of numeric from "timeDate" returns "timeDate",

subtraction of two "timeDate" objects returns "difftime",

other operations between two "timeDate" objects are applied to the underlying times (slot "Date"). The result of that operation is converted to "timeDate" if it represents a time and returned as is otherwise.

Examples

```
## create character vectors
dts = c("1989-09-28", "2001-01-15", "2004-08-30", "1990-02-09")
dts
tms = c(  "23:12:55",   "10:34:02",   "08:30:00",   "11:18:23")
tms

## "+/-"
```

```
# add one day to a given timeDate object
GMT = timeDate(dts, zone = "GMT", FinCenter = "GMT")
GMT
ZUR = timeDate(dts, zone = "GMT", FinCenter = "Europe/Zurich")
ZUR
GMT + 24*3600
ZUR[2] - ZUR[1]
```

`timeSequence` *Regularly spaced 'timeDate' objects*

Description

Create a regularly spaced object of class `"timeDate"`.

Usage

```
timeSequence(from, to = Sys.timeDate(), by, length.out = NULL,
             format = NULL, zone = "", FinCenter = "")

## S3 method for class 'timeDate'
seq(from, to, by, length.out = NULL, along.with = NULL, ...)
```

Arguments

`from`, `to`	starting date, required, and end date, optional. If supplied, `to` must be after (later than) `from`.
`by`	• a character string, containing one of "sec", "min", "hour", "day", "week", "month" or "year". This can optionally be preceded by an integer and a space, or followed by "s". • character string "quarter" that is equivalent to "3 months". • a number, taken to be in seconds. • an object of class 'difftime'. • character string `"DSTday"` gives a sequence taken at the same clock time every day. Note that on the days when the DST changes, the requested time may not exist or be ambiguous, see the examples.
`length.out`	integer, optional. Desired length of the sequence, if specified "to" will be ignored.
`along.with`	Take the length from the length of this argument.
`format`	the format specification of the input character vector.
`zone`	the time zone or financial center where the data were recorded.
`FinCenter`	a character with the location of the financial center named as "continent/city".
`...`	arguments passed to other methods.

Value

an object of class "timeDate"

Note

timeSequence() is a wrapper for the "timeDate" method of seq(), and that has been closely modeled after base R's POSIXt method, seq.POSIXt.

Examples

```
## timeSequence

## autodetection of format
(t1 <- timeSequence(from = "2004-03-12", to = "2004-04-11"))

stopifnot( ## different formats even:
  identical(t1, timeSequence(from = "2004-03-12", to = "11-Apr-2004")),
  identical(t1, ## explicit format and FinCenter :
      timeSequence(from = "2004-03-12", to = "2004-04-11",
                   format = "%Y-%m-%d", FinCenter = "GMT")))

## observe "switch to summer time":
timeSequence(from = "2004-03-26 05:00:00", to = "2004-04-02 05:00:00",
             zone = "Europe/Zurich", FinCenter = "Europe/Zurich")

## ensure fixed clock time:
timeSequence(from = "2004-03-26 05:00:00", to = "2004-04-01 05:00:00",
             by = "DSTday", zone = "Europe/Zurich", FinCenter = "Europe/Zurich")

## on the day of DST change the time may not exist (notice 2004-03-28 00:00:00):
timeSequence(from = "2004-03-26 01:00:00", to = "2004-04-01 01:00:00",
             by = "DSTday", zone = "Europe/Zurich", FinCenter = "Europe/Zurich")
```

---

unique *Remove duplicated dates from 'timeDate' objects*

Description

Remove duplicated dates from "timeDate" objects.

Usage

```
## S3 method for class 'timeDate'
unique(x, ...)
```

Arguments

x	an object of class "timeDate".
...	arguments passed to other methods.

Value

an object of class "timeDate"

Examples

```
## c
# Create Character Vectors:
dts = c("1989-09-28", "2001-01-15", "2004-08-30", "1990-02-09")
dts
tms = c(  "23:12:55",   "10:34:02",   "08:30:00",   "11:18:23")
tms

## "+/-"
# add one day to a given timeDate object
GMT = timeDate(dts, zone = "GMT", FinCenter = "GMT")
GMT
ZUR = timeDate(dts, zone = "GMT", FinCenter = "Europe/Zurich")
ZUR

## c
# Concatenate and Replicate timeDate Objects:
c(GMT[1:2], ZUR[1:2])
c(ZUR[1:2], GMT[1:2])

## rep
rep(ZUR[2], times = 3)
rep(ZUR[2:3], times = 2)
```

---

whichFormat *Format recognation*

Description

Tries to recognize the date/time format.

Usage

```
whichFormat(charvec, silent = FALSE)
```

Arguments

charvec	a character string or vector of dates and times.
silent	a logical flag. Should a warning be printed if the format cannot be recognized?

Value

a format string

Examples

```
## midnightStandard
whichFormat("2007-12-31 24:00")
```

---

window *Time windows*

Description

Extract the subset of a "timeDate" object observed between two time stamps.

Usage

```
## S3 method for class 'timeDate'
window(x, start , end, ...)

## S3 method for class 'timeDate'
cut(x, from , to, ...)
```

Arguments

x	an object of class "timeDate".
start, end	starting date, required, and end date, optional. If supplied to must be after from.
from, to	starting date, required, and end date, optional. If supplied to must be after from.
...	arguments passed to other methods.

Value

an object of class "timeDate"

Note

The method for cut has been discouraged in the sources for a long time (with a recommendation to use window). It was officially deprecated in v4032.109 and will be removed or replaced by a method that is consistent with the methods for cut in base R,

Examples

```
## timeCalendar
# monthly dates in current year
tS = timeCalendar()
tS

## window
# 2nd quarter window:
tS[4:6]
window(tS, tS[4], tS[6])
```

Index