

Package ‘fCalendar’

May 25, 2009

Version 270.78.3

Revision 4191

Date 2009-05-25

Title Chronological and Calendarical Objects

Author Diethelm Wuertz, Yohan Chalabi

Depends R (>= 2.4.0), methods, MASS, fUtilities, fEcofin

Suggests RUnit

Maintainer Rmetrics Core Team <Rmetrics-core@r-project.org>

Description Environment for teaching “Financial Engineering and Computational Finance”

NOTE SEVERAL PARTS ARE STILL PRELIMINARY AND MAY BE CHANGED IN THE FUTURE. THIS TYPICALLY INCLUDES FUNCTION AND ARGUMENT NAMES, AS WELL AS DEFAULTS FOR ARGUMENTS AND RETURN VALUES.

LazyLoad yes

LazyData yes

License GPL (>= 2)

URL <http://www.rmetrics.org>

Repository CRAN

Date/Publication 2009-05-25 20:24:14

R topics documented:

| | |
|--------------------|----|
| fCalendar-package | 3 |
| as | 13 |
| as.timeDate | 14 |
| blockStart | 15 |
| c | 16 |
| currentYear | 16 |
| DaylightSavingTime | 17 |
| dayOfWeek | 18 |
| dayOfYear | 19 |
| diff | 20 |
| difftimeDate | 21 |
| Easter | 21 |
| firstDay | 22 |
| format-methods | 24 |
| holiday | 25 |
| holidayDate | 27 |
| holidayNYSE | 30 |
| holidayTSX | 30 |
| holidayZURICH | 31 |
| isBizday | 32 |
| isWeekday | 32 |
| julian | 33 |
| length | 34 |
| listFinCenter | 35 |
| listHolidays | 36 |
| midnightStandard | 36 |
| myFinCenter | 37 |
| myUnits | 38 |
| nDay | 38 |
| onOrAfter | 39 |
| plot-methods | 40 |
| rep | 41 |
| rev | 42 |
| round | 43 |
| rulesFinCenter | 44 |
| sample | 44 |
| show-methods | 45 |
| sort | 46 |
| start | 47 |
| subset | 48 |
| summary-methods | 49 |
| Sys.timeDate | 49 |
| timeCalendar | 50 |
| timeDate | 51 |
| timeDate-class | 52 |
| timeDateMathOps | 56 |

| | |
|------------------------|----|
| timeSequence | 57 |
| unique | 59 |
| whichFormat | 60 |
| window | 60 |

Index 62

fCalendar-package *Utilities and Tools Package*

Description

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

Details

Package: fCalendar
Type: Package
Version: 270.73
Date: 2008
License: GPL (>= 2)
Copyright: (c) 1999-2008 Diethelm Wuertz, Rmetrics Foundation, GPL
URL: <http://www.rmetrics.org>

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 Rmetrics has added 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"
print(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:
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
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 add 16 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*runif(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(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 ?
print(myFinCenter)

# Change to Zurich:
myFinCenter = "Zurich"
print(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:
Easter(currentYear:(currentYear+5))
```

Holiday Calendars:

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

We would like to thanks 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.

as

Any to 'timeDate' Coercion

Description

Coerce other time date representations into an object 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 <code>timeDate</code> . |
| 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, <code>FinCenter</code> , as used by <code>timeDate</code> objects. |
| method | a character string denoting the method how to determine the dates. |
| ... | arguments passed to other methods. |

Value

return an object of class `timeDate`.

Examples

```
## as.character -
# Convert 'timeDate' to a character strings:
as.character(timeCalendar())
```

as.timeDate *Coercion into 'timeDate' Objects*

Description

Coerce and transform objects of class 'timeDate'.

Usage

```
## Default S3 method:
as.timeDate(x, zone = myFinCenter, FinCenter =
myFinCenter)

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

## S3 method for class 'Date':
as.timeDate(x, zone = myFinCenter, FinCenter =
myFinCenter)
## S3 method for class 'timeDate':
as.timeDate(x, zone = x@FinCenter, FinCenter =
myFinCenter)
```

Arguments

x an object of class timeDate.
zone the time zone or financial center where the data were recorded.
FinCenter a character with the the location of the financial center named as "continent/city".

Value

as.timeDate.POSIXt returns an object of class timeDate.
as.timeDate.Date returns an object of class timeDate.

Examples

```
## timeDate -
tC = timeCalendar()

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

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 <code>timeDate</code> . |

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

returns an object of class "timeDate".

Examples

```
## timeSequence
# 360 Days Series:
tS <- timeSequence(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

returns 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))
```

currentYear *Current Year*

Description

A variable with the current year.

Usage

```
currentYear
```

Value

returns the current year, a numeric variable.

Note

It is not allowed to change this variable.

Examples

```
## currentYear -
  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.

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's GPL licence.

Examples

```
## DST Rules for Zurich:
  head(Zurich())
  tail(Zurich())
```

`dayOfWeek`*Day of the Week*

Description

returns the day of the year from a 'timeDate' object.

Usage

```
dayOfWeek(x)
```

Arguments

`x` an object of class `timeDate`.

Value

returns a three letter character string with the names in English of the day of the week,

Note

With version 2.7 the function has been renamed from `getDayOfWeek`.

See Also

[dayOfYear](#)

Examples

```
## timeCalendar -
  tC = timeCalendar()

## The days of the Year:
  dayOfWeek(tC)

## Use Deprecated Function:
  getDayOfWeek <- dayOfWeek
  getDayOfWeek(tC)
```

| | |
|-----------|------------------------|
| dayOfYear | <i>Day of the Year</i> |
|-----------|------------------------|

Description

returns the day of the year from a 'timeDate' object.

Usage

```
dayOfYear(x)
```

Arguments

x an object of class `timeDate`.

Value

returns the day count as integer value starting January, 1st.

Note

With version 2.7 the function has been renamed from `getDayOfYear`.

See Also

[dayOfWeek](#)

Examples

```
## timeCalendar -  
tC = timeCalendar()  
  
## The days of the Year:  
dayOfYear(tC)  
  
## Use Deprecated Function:  
getDayOfYear <- dayOfYear  
getDayOfYear(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 a 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

The function, 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)])
```

Easter *Date of Easter*

Description

Returns the date of Easter.

Usage

```
Easter(year = 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

returns the date of Easter as an object of class `timeDate`.

Examples

```
## Easter -

# Current Year:
Easter()

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

firstDay

First and Last Days

Description

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

Usage

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

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

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 the location of the financial center named as "continent/city".

Value

returns an object of class `timeDate`.

For the functions `timeLastDayInMonth` and `timeFirstDayInMonth` 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 `timeFirstDayInQuarter`.

Examples

```
## Date as character String:
charvec = "2006-04-16"

## 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, ...)

```

Arguments

```

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

```

Value

returns an ISO conform formatted character string.

Note

This S3 method will become in the future a S4 method

See Also

```

as.character.

```

Examples

```

## timeCalendar -
# Time Calebdar 16:00
tC = timeCalendar() + 16*3600
tC

## Format as ISO Character String:
format(tC)

```

| | |
|---------|----------------------|
| holiday | <i>Holiday Dates</i> |
|---------|----------------------|

Description

Returns the date of a holiday.

Usage

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

Arguments

| | |
|---------|---|
| Holiday | the unquoted function name of an ecclestial or public holiday in the G7 countries or Switzerland, see the list below. |
| year | an integer value or vector of years, formatted as YYYY. |

Details

Easter is the central ecclestial holiday. Many other holidays are related to this feast. The function `Easter` computes the dates of Easter and related ecclestial holidays for the requested year vector. `holiday` calculates the dates of ecclestial or public 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 ecclestial 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:

GBMayDay, GBBankHoliday, GBSummerBankHoliday, GBNewYearsEve.

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.

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, JPShuubun-no-hi, JPHealthandSportsDay, JPTaiikuNoHi, JPNationalCultureDay, JPBunkaNoHi, JPThanksgivingDay, JPKinrouKanshaNohi, JPKinrou-kansha-no-hi, JPEmperorsBirthday, JPTennou-tanjyou-bi, JPTennou-tanjyou-bi.

Value

The function `holiday` returns an object of class `timeDate`.

Examples

```
## holiday -
# Dates for GoodFriday from 2000 until 2010:
holiday(2000:2010, "GoodFriday")

## Easter -
Easter(2000:2010)
```

```
## GoodFriday -  
    GoodFriday(2000:2010)  
    Easter(2000:2010, -2)
```

holidayDate

Public and Ecclesiastical Holidays

Description

A collection and description of functions and methods dealing with holiday dates in the G7 countries and Switzerland.

Usage

```
Septuagesima(year = currentYear)  
Quinquagesima(year = currentYear)  
AshWednesday(year = currentYear)  
PalmSunday(year = currentYear)  
GoodFriday(year = currentYear)  
EasterSunday(year = currentYear)  
EasterMonday(year = currentYear)  
RogationSunday(year = currentYear)  
Ascension(year = currentYear)  
Pentecost(year = currentYear)  
PentecostMonday(year = currentYear)  
TrinitySunday(year = currentYear)  
CorpusChristi(year = currentYear)  
ChristTheKing(year = currentYear)  
Advent1st(year = currentYear)  
Advent2nd(year = currentYear)  
Advent3rd(year = currentYear)  
Advent4th(year = currentYear)  
ChristmasEve(year = currentYear)  
ChristmasDay(year = currentYear)  
BoxingDay(year = currentYear)  
NewYearsDay(year = currentYear)  
SolemnityOfMary(year = currentYear)  
Epiphany(year = currentYear)  
PresentationOfLord(year = currentYear)  
Annunciation(year = currentYear)  
TransfigurationOfLord(year = currentYear)  
AssumptionOfMary(year = currentYear)  
BirthOfVirginMary(year = currentYear)  
CelebrationOfHolyCross(year = currentYear)  
MassOfArchangels(year = currentYear)  
AllSaints(year = currentYear)
```

AllSouls(year = currentYear)
LaborDay(year = currentYear)
CHBerchtoldsDay(year = currentYear)
CHSechselaeuten(year = currentYear)
CHAscension(year = currentYear)
CHConfederationDay(year = currentYear)
CHKnabenschiessen(year = currentYear)
GBMayDay(year = currentYear)
GBBankHoliday(year = currentYear)
GBSummerBankHoliday(year = currentYear)
GBMilleniumDay(year = currentYear)
DEAscension(year = currentYear)
DECorpusChristi(year = currentYear)
DEGermanUnity(year = currentYear)
DEChristmasEve(year = currentYear)
DENewYearsEve(year = currentYear)
FRFetDeLaVictoire1945(year = currentYear)
FRAscension(year = currentYear)
FRBastilleDay(year = currentYear)
FRAssumptionVirginMary(year = currentYear)
FRAllSaints(year = currentYear)
FRArmisticeDay(year = currentYear)
ITEpiphany(year = currentYear)
ITLiberationDay(year = currentYear)
ITAssumptionOfVirginMary(year = currentYear)
ITAllSaints(year = currentYear)
ITStAmrose(year = currentYear)
ITImmaculateConception(year = currentYear)
USDecorationMemorialDay(year = currentYear)
USPresidentsDay(year = currentYear)
USNewYearsDay(year = currentYear)
USInaugurationDay(year = currentYear)
USMLKingsBirthday(year = currentYear)
USLincolnsBirthday(year = currentYear)
USWashingtonsBirthday(year = currentYear)
USMemorialDay(year = currentYear)
USIndependenceDay(year = currentYear)
USLaborDay(year = currentYear)
USColumbusDay(year = currentYear)
USElectionDay(year = currentYear)
USVeteransDay(year = currentYear)
USThanksgivingDay(year = currentYear)
USChristmasDay(year = currentYear)
USCPulaskisBirthday(year = currentYear)
USGoodFriday(year = currentYear)
CAVictoriaDay(year = currentYear)
CACanadaDay(year = currentYear)
CACivicProvincialHoliday(year = currentYear)

```

CALabourDay(year = currentYear)
CAThanksgivingDay(year = currentYear)
CaRemembranceDay(year = currentYear)
JPVernalEquinox(year = currentYear)
JPNewYearsDay(year = currentYear)
JPGantan(year = currentYear)
JPBankHolidayJan2(year = currentYear)
JPBankHolidayJan3(year = currentYear)
JPComingOfAgeDay(year = currentYear)
JPSeijinNoHi(year = currentYear)
JPNatFoundationDay(year = currentYear)
JPKenkokuKinenNoHi(year = currentYear)
JPGreeneryDay(year = currentYear)
JPMidoriNoHi(year = currentYear)
JPConstitutionDay(year = currentYear)
JPKenpouKinenBi(year = currentYear)
JPNationHoliday(year = currentYear)
JPKokuminNoKyuujitu(year = currentYear)
JPChildrensDay(year = currentYear)
JPKodomoNoHi(year = currentYear)
JPMarineDay(year = currentYear)
JPUmiNoHi(year = currentYear)
JPRespectForTheAgedDay(year = currentYear)
JPKeirouNOhi(year = currentYear)
JPAutumnalEquinox(year = currentYear)
JPShuubunNoHi(year = currentYear)
JPHealthandSportsDay(year = currentYear)
JPTaiikuNoHi(year = currentYear)
JPNationalCultureDay(year = currentYear)
JPBunkaNoHi(year = currentYear)
JPThanksgivingDay(year = currentYear)
JPKinrouKanshaNoHi(year = currentYear)
JPEmperorsBirthday(year = currentYear)
JPTennouTanjyouBi(year = currentYear)
JPBankHolidayDec31(year = currentYear)

```

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 function `listHolidays` returns a character vector with the names of the supported holidays.

The holiday functions return an ISO-8601 formatted 'timeDate' of the requested holiday.

Examples

```
## listHolidays -
```

```
listHolidays()

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

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

holidayNYSE *NYSE Holiday Calendar*

Description

Returns a holiday calendar for the New York Stock Exchange.

Usage

```
holidayNYSE(year = currentYear)
```

Arguments

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

Value

returns an object of class `timeDate`.

Examples

```
## holidayNYSE -
holidayNYSE()
holidayNYSE(2008:2010)
```

holidayTSX *TSX Holiday Calendar*

Description

Returns a holiday calendar for the Toronto Stock Exchange.

Usage

```
holidayTSX(year = currentYear)
```

Arguments

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

Value

returns an object of class `timeDate`.

Examples

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

| | |
|---------------|--------------------------------|
| holidayZURICH | <i>Zurich Holiday Calendar</i> |
|---------------|--------------------------------|

Description

Returns a holiday calendar for Zurich.

Usage

```
holidayZURICH(year = 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`, `CHSechse-laeuten`, `CHAscension`, `CHConfederationDay`, `CHKnabenschiessen`.

Value

returns an object of class `timeDate`.

Examples

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

`isBizday`*Business and Holidays*

Description

Tests if a date is a business day or not.

Usage

```
isBizday(x, holidays = holidayNYSE())  
isHoliday(x, holidays = holidayNYSE())
```

Arguments

`holidays` holiday dates from a holiday calendar. An object of class `timeDate`.
`x` an object of class `timeDate`.

Value

the functions return logical vectors indicating if a date is a business day, or a holiday.

Examples

```
## Dates in April, 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())]
```

`isWeekday`*Weekdays and Weekends*

Description

Tests if a date is a weekday or not.

Usage

```
isWeekday(x)  
isWeekend(x)
```

Arguments

`x` an object of class `timeDate`.

Value

the functions return logical vectors indicating if a date is a weekday, or a weekend day.

Examples

```
## Dates in April, 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, ...)

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

## S3 method for class 'timeDate':
months(x, abbreviate = NULL)
```

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 with the the location of the financial center named as "continent/city".
`abbreviate` currently not used.
`...` arguments passed to other methods.

Value

`julian` returns a `timeDate` object as a Julian count.
`atoms` and `months` extrac from a `timeDate` object the calendar atoms, i.e. the year, month, day, and optionally hour, minute and second.

Examples

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

## atoms -
  atoms(tC)

## months -
  months(tC)
```

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

returns an integer of length 1.

Examples

```
## timCalendar -
  tC = timeCalendar()

## length -
  length(tC)
```

| | |
|---------------|----------------------------------|
| listFinCenter | <i>List of Financial Centers</i> |
|---------------|----------------------------------|

Description

Lists supported financial centers.

Usage

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

Arguments

`pattern` a pattern character string as required by the [grep](#) function.

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 all time zone and day light saving time information is stored locally in ASCII files.

Value

returns a list of supported financial centers.

Examples

```
## myFinCenter - the global setting currently used:
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 Holidayss*

Description

Returns the list of holidays.

Usage

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

Arguments

pattern a pattern character string as required by the [grep](#) function.

Value

returns a list of holidays as a character vector.

Examples

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

midnightStandard *Midnight Standard*

Description

Corrects 'timeDate' objects if they do not fulfill the ISO8601 midnigth standard.

Usage

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

Arguments

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

Value

returns a `timeDate` object.

Examples

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

| | |
|-------------|-----------------------------|
| myFinCenter | <i>myFinCenter Variable</i> |
|-------------|-----------------------------|

Description

A character string with the name of my financial center.

Usage

```
myFinCenter
```

Value

returns the name of myFinCenter.

Note

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

See Also

[listFinCenter](#)

Examples

```
## myFinCenter - the global setting currently used:  
myFinCenter  
  
## Change to another Financier Center:  
# myFinCenter = "Zurich"  
  
## Do not take care about DST ...  
# myFinCenter = "GMT"
```

| | |
|---------|-------------------------------------|
| myUnits | <i>Frequency of date/time Units</i> |
|---------|-------------------------------------|

Description

A variable with the frequency of date/units.

Usage

```
myUnits
```

Value

returns the the date/time units, a acharacter value. By default "days".

Examples

```
## myUnits -
  myUnits
```

| | |
|------|-------------------------|
| nDay | <i>n-th n-day Dates</i> |
|------|-------------------------|

Description

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

Usage

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

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

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 occurrence. |
| 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 the location of the financial center named as "continent/city". |

Value

returns an object of class `timeDate`.

For function `timeNthNdayInMonth` the `nth` occurrence of a `n`-day (`nth = 1,...,5`) in `year`, `month`, and for `timeLastNdayInMonth` the last `nday` in `year`, `month` will be returned.

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

OnOrAfter/Before Dates

Description

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

Usage

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

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

Arguments

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

Value

returns an object of class `timeDate`.

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

For the function `timeNdayOnOrBefore` the date that is a `n`-day on or before the given date will be returned.

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)
```

plot-methods

Plot Methods

Description

Plot methods for `timeDate` objects.

Usage

```
## S3 method for class 'timeDate':
plot(x, y, ...)
## S3 method for class 'timeDate':
lines(x, y, ...)
## S3 method for class 'timeDate':
points(x, y, ...)
## S3 method for class 'timeDate':
axis(side, x, at, format = NULL, labels = TRUE, ...)
```

Arguments

| | |
|-----------------------|---|
| <code>x, y, at</code> | an object of class <code>timeDate</code> . |
| <code>side</code> | 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. |
| <code>format</code> | format - format string. |
| <code>labels</code> | either a logical value specifying whether annotations are to be made at the tickmarks, or a vector of character strings to be placed at the tickpoints. |
| <code>...</code> | 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.

Note

These S3 methods will become S4 methods in the future.

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 a 'timeDate' object.

Usage

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

Arguments

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

Value

returns 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)

```

rev

Reverting 'timeDate' Objects

Description

Reverts a 'timeDate' object.

Usage

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

Arguments

x an object of class timeDate.

Value

returns 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)
```

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

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

Value

The two functions `round` and `trunc` allow to round or to truncate `timeDate` objects to the specified unit and return them as `timeDate` objects. - Note, there is an inconsistency `round` uses `digits` as argument and not `units`.

Examples

```
## round -
## truncate -
```

```
rulesFinCenter      Financial Centers DST Rules
```

Description

Returns DST rules for a financial center.

Usage

```
rulesFinCenter(FinCenter = myFinCenter)
```

Arguments

`FinCenter` a character with the 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 all time zone and day light saving time information is stored locally in ASCII files.

Value

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

Examples

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

```
sample      Resampling 'timeDate' Objects
```

Description

Resamples a 'timeDate' object.

Usage

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

Arguments

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

Value

returns 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())
```

`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

returns 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)
```

start *Terminal Times and Range*

Description

Extracts the time the first or last observation was taken, or computes the range.

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 <code>timeDate</code> . |
| ... | [start][end] - not used, [min][max] - 'timeDates' objects. |
| na.rm | not used. |

Details

Note, the series will be time ordered before the start or end time stamps are extracted. Sorting is done in the way that the first observation appears in time before the last observation.

Value

returns 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.

Usage

```
## S3 method for class 'timeDate':
x[... , drop = TRUE]
```

Arguments

| | |
|------|------------------------------------|
| x | an object of class timeDate. |
| ... | arguments passed to other methods. |
| drop | a logical flag, by default TRUE. |

Value

returns 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.

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.

Note

This S3 method will become in the future a S4 method

Examples

```
## summary -
tC = timeCalendar()
summary(tC)
```

Sys.timeDate *System Time as 'timeDate' Object*

Description

Returns system time as an object of class 'timeDate'.

Usage

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

Arguments

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

Value

returns the system time as class "timeDate" object.

Examples

```
## Sys.time -
# direct
Sys.timeDate()

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

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

timeCalendar *'timeDate' from Calendar Atoms*

Description

Create a 'timeDate' object from calendar atoms.

Usage

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

Arguments

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

Value

returns a S4 object of class "timeDate".

Examples

```
## timeCalendar -

# Current Year:
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 | <i>'timeDate' Objects from Scratch</i> |
|----------|--|

Description

Create a 'timeDate' object from scratch using a character vector.

Usage

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

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 the location of the financial center named as "continent/city". |

Value

returns an object of class timeDate.

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")

## 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

```

```
timeDate-class      Class "timeDate"
```

Description

The 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 system in 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 differences. 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 generate 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 "-" opera-

tions, 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 for example by calls of the functions `timeDate`, `timeCalender` and `timeCalendar` among others.

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 the location of the financial center named as "continent/city", or just "city".

Methods

`show signature(object = "timeDate")`: prints an object of class '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 has not longer 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 -

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

## 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()
print(tC)
summary(tC)
format(tC)

```

timeDateMathOps *timeDate Mathematical Operations*

Description

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

The functions are:

| | |
|--------------|--|
| Ops.timeDate | Group 'Ops' generic functions for 'timeDate' objects, |
| +.timeDate | Performs arithmetic + operation on 'timeDate' objects, |
| -.timeDate | Performs arithmetic - operation on 'timeDate' objects. |

Usage

```

## S3 method for class 'timeDate':
Ops(e1, e2)

## S3 method for class 'timeDate':
e1 + e2

```

```
## S3 method for class 'timeDate':
e1 - e2
```

Arguments

`e1`, `e2` usually objects of class `timeDate`, in the case of addition and subtraction `e2` may be of class `numeric`.

Value

`Ops.timeDate`
these are functions for mathematical operations. Group `Ops` are generic functions which manage mathematical operations.

`+.timeDate`

`-.timeDate`

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`.

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 = format(Sys.time(), "%Y-%m-%d"),
             by = c("day", "year", "quarter", "month", "week", "hour", "min", "sec"),
             length.out = NULL, format = NULL,
             zone = myFinCenter, FinCenter = myFinCenter)

## S3 method for class 'timeDate':
seq(from, to, by = c("day", "year", "quarter", "month",
                    "week", "hour", "min", "sec"), length.out = NULL, ...)
```

Arguments

| | |
|------------|---|
| from, to | [timeSequence] - starting date, required, and end date, optional. If supplied to must be after from, [seq] - cr in this case the from and to dates must be objects of class timeDate. |
| 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". |
| length.out | length.out integer, optional. Desired length of the sequence, if specified "to" will be ignored. |
| 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 the location of the financial center named as "continent/city". |
| ... | arguments passed to other methods. |

Value

returns a S4 object of class "timeDate".

Note

seq is a synonyme generic function for timeSequence.

Examples

```
## 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")
```

unique

Making a 'timeDate' object unique

Description

Makes a 'timeDate' object unique.

Usage

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

Arguments

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

Value

returns 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)
```


Value

returns an object of class `timeDate`.

Examples

```
## timeCalendar -  
# Monthly Dates in Current Year:  
tS = timeCalendar()  
tS  
  
## window -  
# 2nd Quarter Window:  
tS[4:6]  
window(tS, tS[4], tS[6])
```

Index

*Topic **chron**

- as, 12
- as.timeDate, 13
- blockStart, 14
- c, 15
- currentYear, 15
- dayOfWeek, 17
- dayOfYear, 18
- diff, 19
- difftimeDate, 20
- Easter, 20
- firstDay, 21
- format-methods, 23
- holiday, 24
- holidayDate, 26
- holidayNYSE, 29
- holidayTSX, 29
- holidayZURICH, 30
- isBizday, 31
- isWeekday, 31
- julian, 32
- length, 33
- listFinCenter, 34
- listHolidays, 35
- midnightStandard, 35
- myFinCenter, 36
- myUnits, 37
- nDay, 37
- onOrAfter, 38
- plot-methods, 39
- rep, 40
- rev, 41
- round, 42
- rulesFinCenter, 43
- sample, 43
- show-methods, 44
- sort, 45
- start, 46
- subset, 47

- summary-methods, 48
- Sys.timeDate, 48
- timeCalendar, 49
- timeDate, 50
- timeDate-class, 51
- timeDateMathOps, 55
- timeSequence, 56
- unique, 58
- whichFormat, 59
- window, 59

*Topic **data**

- DaylightSavingTime, 16

*Topic **hplot**

- plot-methods, 39

*Topic **package**

- fCalendar-package, 2

- + .timeDate (timeDateMathOps), 55
- .timeDate (timeDateMathOps), 55
- [.timeDate (subset), 47
- [<- .timeDate (subset), 47

- Abidjan (DaylightSavingTime), 16
- Accra (DaylightSavingTime), 16
- Adak (DaylightSavingTime), 16
- Addis_Ababa (DaylightSavingTime), 16
- Adelaide (DaylightSavingTime), 16
- Aden (DaylightSavingTime), 16
- Advent1st (holidayDate), 26
- Advent2nd (holidayDate), 26
- Advent3rd (holidayDate), 26
- Advent4th (holidayDate), 26
- Algiers (DaylightSavingTime), 16
- AllSaints (holidayDate), 26
- AllSouls (holidayDate), 26
- Almaty (DaylightSavingTime), 16
- Amman (DaylightSavingTime), 16
- Amsterdam (DaylightSavingTime), 16
- Anadyr (DaylightSavingTime), 16
- Anchorage (DaylightSavingTime), 16

- Andorra (*DaylightSavingTime*), 16
- Anguilla (*DaylightSavingTime*), 16
- Annunciation (*holidayDate*), 26
- Antananarivo
 - (*DaylightSavingTime*), 16
- Antigua (*DaylightSavingTime*), 16
- Any to 'timeDate' Coercion (*as*), 12
- Apia (*DaylightSavingTime*), 16
- Aqtau (*DaylightSavingTime*), 16
- Aqtobe (*DaylightSavingTime*), 16
- Araguaina (*DaylightSavingTime*), 16
- Aruba (*DaylightSavingTime*), 16
- as*, 12
- as.character.timeDate* (*as*), 12
- as.data.frame.timeDate* (*as*), 12
- as.Date.timeDate* (*as*), 12
- as.double.timeDate* (*as*), 12
- as.POSIXct.timeDate* (*as*), 12
- as.POSIXlt.timeDate* (*as*), 12
- as.timeDate*, 13
- Ascension (*holidayDate*), 26
- Ashgabat (*DaylightSavingTime*), 16
- AshWednesday (*holidayDate*), 26
- Asmara (*DaylightSavingTime*), 16
- AssumptionOfMary (*holidayDate*), 26
- Asuncion (*DaylightSavingTime*), 16
- Athens (*DaylightSavingTime*), 16
- Atikokan (*DaylightSavingTime*), 16
- atoms.timeDate* (*julian*), 32
- Auckland (*DaylightSavingTime*), 16
- axis.timeDate* (*plot-methods*), 39
- Azores (*DaylightSavingTime*), 16

- Baghdad (*DaylightSavingTime*), 16
- Bahia (*DaylightSavingTime*), 16
- Bahrain (*DaylightSavingTime*), 16
- Baku (*DaylightSavingTime*), 16
- Bamako (*DaylightSavingTime*), 16
- Bangkok (*DaylightSavingTime*), 16
- Bangui (*DaylightSavingTime*), 16
- Banjul (*DaylightSavingTime*), 16
- Barbados (*DaylightSavingTime*), 16
- Beirut (*DaylightSavingTime*), 16
- Belem (*DaylightSavingTime*), 16
- Belgrade (*DaylightSavingTime*), 16
- Belize (*DaylightSavingTime*), 16
- Berlin (*DaylightSavingTime*), 16
- Bermuda (*DaylightSavingTime*), 16

- BirthOfVirginMary (*holidayDate*), 26
- Bishkek (*DaylightSavingTime*), 16
- Bissau (*DaylightSavingTime*), 16
- Blanc-Sablon
 - (*DaylightSavingTime*), 16
- Blantyre (*DaylightSavingTime*), 16
- blockEnd* (*blockStart*), 14
- blockStart*, 14
- Boa_Vista (*DaylightSavingTime*), 16
- Bogota (*DaylightSavingTime*), 16
- Boise (*DaylightSavingTime*), 16
- BoxingDay (*holidayDate*), 26
- Bratislava (*DaylightSavingTime*), 16
- Brazzaville (*DaylightSavingTime*), 16
- Brisbane (*DaylightSavingTime*), 16
- Broken_Hill (*DaylightSavingTime*), 16
- Brunei (*DaylightSavingTime*), 16
- Brussels (*DaylightSavingTime*), 16
- Bucharest (*DaylightSavingTime*), 16
- Budapest (*DaylightSavingTime*), 16
- Buenos_Aires
 - (*DaylightSavingTime*), 16
- BuenosAires (*DaylightSavingTime*), 16
- Bujumbura (*DaylightSavingTime*), 16

- c*, 15
- CACanadaDay (*holidayDate*), 26
- CACivicProvincialHoliday
 - (*holidayDate*), 26
- CAFfamilyDay (*holidayDate*), 26
- Cairo (*DaylightSavingTime*), 16
- CALabourDay (*holidayDate*), 26
- Calcutta (*DaylightSavingTime*), 16
- Cambridge_Bay
 - (*DaylightSavingTime*), 16
- Campo_Grande
 - (*DaylightSavingTime*), 16
- Canary (*DaylightSavingTime*), 16
- Cancun (*DaylightSavingTime*), 16
- Cape_Verde (*DaylightSavingTime*), 16
- Caracas (*DaylightSavingTime*), 16
- CaRemembranceDay (*holidayDate*), 26

- Casablanca (*DaylightSavingTime*),
16
- Casey (*DaylightSavingTime*), 16
- Catamarca (*DaylightSavingTime*), 16
- CAThanksgivingDay (*holidayDate*),
26
- CAVictoriaDay (*holidayDate*), 26
- Cayenne (*DaylightSavingTime*), 16
- Cayman (*DaylightSavingTime*), 16
- CelebrationOfHolyCross
(*holidayDate*), 26
- Center (*DaylightSavingTime*), 16
- Ceuta (*DaylightSavingTime*), 16
- Chagos (*DaylightSavingTime*), 16
- CHAscension (*holidayDate*), 26
- Chatham (*DaylightSavingTime*), 16
- CHBerchtoldsDay (*holidayDate*), 26
- CHConfederationDay (*holidayDate*),
26
- Chicago (*DaylightSavingTime*), 16
- Chihuahua (*DaylightSavingTime*), 16
- Chisinau (*DaylightSavingTime*), 16
- CHKnabenschuessen (*holidayDate*),
26
- Choibalsan (*DaylightSavingTime*),
16
- Chongqing (*DaylightSavingTime*), 16
- Christmas (*DaylightSavingTime*), 16
- ChristmasDay (*holidayDate*), 26
- ChristmasEve (*holidayDate*), 26
- ChristTheKing (*holidayDate*), 26
- CHSechselaeuten (*holidayDate*), 26
- class, 49, 57
- Cocos (*DaylightSavingTime*), 16
- Colombo (*DaylightSavingTime*), 16
- Comoro (*DaylightSavingTime*), 16
- Conakry (*DaylightSavingTime*), 16
- Copenhagen (*DaylightSavingTime*),
16
- Cordoba (*DaylightSavingTime*), 16
- CorpusChristi (*holidayDate*), 26
- Costa_Rica (*DaylightSavingTime*),
16
- Cuiaba (*DaylightSavingTime*), 16
- Curacao (*DaylightSavingTime*), 16
- currentYear, 15
- Currie (*DaylightSavingTime*), 16
- cut.timeDate (*window*), 59
- Dakar (*DaylightSavingTime*), 16
- Damascus (*DaylightSavingTime*), 16
- Danmarkshavn
(*DaylightSavingTime*), 16
- Dar_es_Salaam
(*DaylightSavingTime*), 16
- Darwin (*DaylightSavingTime*), 16
- Davis (*DaylightSavingTime*), 16
- Dawson (*DaylightSavingTime*), 16
- Dawson_Creek
(*DaylightSavingTime*), 16
- DaylightSavingTime, 16
- dayOfWeek, 17, 18
- dayOfYear, 17, 18
- DEAscension (*holidayDate*), 26
- DEChristmasEve (*holidayDate*), 26
- DECorpusChristi (*holidayDate*), 26
- DEGermanUnity (*holidayDate*), 26
- DENewYearsEve (*holidayDate*), 26
- Denver (*DaylightSavingTime*), 16
- Detroit (*DaylightSavingTime*), 16
- Dhaka (*DaylightSavingTime*), 16
- diff, 19, 53
- difftoDate, 20, 53
- Dili (*DaylightSavingTime*), 16
- Djibouti (*DaylightSavingTime*), 16
- Dominica (*DaylightSavingTime*), 16
- Douala (*DaylightSavingTime*), 16
- Dubai (*DaylightSavingTime*), 16
- Dublin (*DaylightSavingTime*), 16
- DumontDUrville
(*DaylightSavingTime*), 16
- Dushanbe (*DaylightSavingTime*), 16
- Easter, 20
- EasterMonday (*holidayDate*), 26
- Eastern (*DaylightSavingTime*), 16
- EasterSunday (*holidayDate*), 26
- Edmonton (*DaylightSavingTime*), 16
- Efate (*DaylightSavingTime*), 16
- Eirunepe (*DaylightSavingTime*), 16
- El_Aaiun (*DaylightSavingTime*), 16
- El_Salvador (*DaylightSavingTime*),
16
- end (*start*), 46
- Enderbury (*DaylightSavingTime*), 16
- Epiphany (*holidayDate*), 26
- Eucla (*DaylightSavingTime*), 16

- Fakaofo (*DaylightSavingTime*), 16
- Faroe (*DaylightSavingTime*), 16
- fCalendar (*fCalendar-package*), 2
- fCalendar-package, 2
- Fiji (*DaylightSavingTime*), 16
- firstDay, 21
- format (*format-methods*), 23
- format-methods, 23
- Fortaleza (*DaylightSavingTime*), 16
- FRAllSaints (*holidayDate*), 26
- Frankfurt (*DaylightSavingTime*), 16
- FRArmisticeDay (*holidayDate*), 26
- FRAscension (*holidayDate*), 26
- FRAssumptionVirginMary (*holidayDate*), 26
- FRBastilleDay (*holidayDate*), 26
- Freetown (*DaylightSavingTime*), 16
- FRFetDeLaVictoire1945 (*holidayDate*), 26
- Funafuti (*DaylightSavingTime*), 16

- Gaborone (*DaylightSavingTime*), 16
- Galapagos (*DaylightSavingTime*), 16
- Gambier (*DaylightSavingTime*), 16
- Gaza (*DaylightSavingTime*), 16
- GBBankHoliday (*holidayDate*), 26
- GBMayDay (*holidayDate*), 26
- GBMilleniumDay (*holidayDate*), 26
- GBSummerBankHoliday (*holidayDate*), 26
- Gibraltar (*DaylightSavingTime*), 16
- Glace_Bay (*DaylightSavingTime*), 16
- Godthab (*DaylightSavingTime*), 16
- GoodFriday (*holidayDate*), 26
- Goose_Bay (*DaylightSavingTime*), 16
- Grand_Turk (*DaylightSavingTime*), 16
- Grenada (*DaylightSavingTime*), 16
- grep, 34, 35
- Guadalcanal (*DaylightSavingTime*), 16
- Guadeloupe (*DaylightSavingTime*), 16
- Guam (*DaylightSavingTime*), 16
- Guatemala (*DaylightSavingTime*), 16
- Guayaquil (*DaylightSavingTime*), 16
- Guernsey (*DaylightSavingTime*), 16
- Guyana (*DaylightSavingTime*), 16

- Halifax (*DaylightSavingTime*), 16
- Harare (*DaylightSavingTime*), 16
- Harbin (*DaylightSavingTime*), 16
- Havana (*DaylightSavingTime*), 16
- Helsinki (*DaylightSavingTime*), 16
- Hermosillo (*DaylightSavingTime*), 16
- Hobart (*DaylightSavingTime*), 16
- holiday, 24
- holidayDate, 26
- holidayNYSE, 29
- holidayTSX, 29
- holidayZURICH, 30
- Hong_Kong (*DaylightSavingTime*), 16
- HongKong (*DaylightSavingTime*), 16
- Honolulu (*DaylightSavingTime*), 16
- Hovd (*DaylightSavingTime*), 16

- Indianapolis (*DaylightSavingTime*), 16
- Inuvik (*DaylightSavingTime*), 16
- Iqaluit (*DaylightSavingTime*), 16
- Irkutsk (*DaylightSavingTime*), 16
- isBizday, 31
- isHoliday (*isBizday*), 31
- Isle_of_Man (*DaylightSavingTime*), 16
- Istanbul (*DaylightSavingTime*), 16
- isWeekday, 31
- isWeekend (*isWeekday*), 31
- ITAllSaints (*holidayDate*), 26
- ITAssumptionOfVirginMary (*holidayDate*), 26
- ITEpiphany (*holidayDate*), 26
- ITImmaculateConception (*holidayDate*), 26
- ITLiberationDay (*holidayDate*), 26
- ITStAmrose (*holidayDate*), 26

- Jakarta (*DaylightSavingTime*), 16
- Jamaica (*DaylightSavingTime*), 16
- Jayapura (*DaylightSavingTime*), 16
- Jersey (*DaylightSavingTime*), 16
- Jerusalem (*DaylightSavingTime*), 16
- Johannesburg (*DaylightSavingTime*), 16
- Johnston (*DaylightSavingTime*), 16
- JPAutumnalEquinox (*holidayDate*), 26

- JPBankHolidayDec31 (*holidayDate*),
26
- JPBankHolidayJan2 (*holidayDate*),
26
- JPBankHolidayJan3 (*holidayDate*),
26
- JPBunkaNoHi (*holidayDate*), 26
- JPChildrensDay (*holidayDate*), 26
- JPComingOfAgeDay (*holidayDate*), 26
- JPConstitutionDay (*holidayDate*),
26
- JPEmperorsBirthday (*holidayDate*),
26
- JPGantan (*holidayDate*), 26
- JPGreeneryDay (*holidayDate*), 26
- JPHealthandSportsDay
(*holidayDate*), 26
- JPKeirouNOhi (*holidayDate*), 26
- JPKenkokuKinenNoHi (*holidayDate*),
26
- JPKenpouKinenBi (*holidayDate*), 26
- JPKinrouKanshaNoHi (*holidayDate*),
26
- JPKodomoNoHi (*holidayDate*), 26
- JPKokuminNoKyujitu (*holidayDate*),
26
- JPMarineDay (*holidayDate*), 26
- JPMidoriNoHi (*holidayDate*), 26
- JPNatFoundationDay (*holidayDate*),
26
- JPNationalCultureDay
(*holidayDate*), 26
- JPNationHoliday (*holidayDate*), 26
- JPNewYearsDay (*holidayDate*), 26
- JPRespectForTheAgedDay
(*holidayDate*), 26
- JPSeijinNoHi (*holidayDate*), 26
- JPShuubunNoHi (*holidayDate*), 26
- JPTaiikuNoHi (*holidayDate*), 26
- JPTennouTanjyouBi (*holidayDate*),
26
- JPThanksgivingDay (*holidayDate*),
26
- JPUmiNoHi (*holidayDate*), 26
- JPVernalEquinox (*holidayDate*), 26
- Jujuy (*DaylightSavingTime*), 16
- julian, 32, 53
- julian.timeDate (*julian*), 32
- Juneau (*DaylightSavingTime*), 16
- Kabul (*DaylightSavingTime*), 16
- Kaliningrad (*DaylightSavingTime*),
16
- Kamchatka (*DaylightSavingTime*), 16
- Kampala (*DaylightSavingTime*), 16
- Karachi (*DaylightSavingTime*), 16
- Kashgar (*DaylightSavingTime*), 16
- Katmandu (*DaylightSavingTime*), 16
- Kerguelen (*DaylightSavingTime*), 16
- Khartoum (*DaylightSavingTime*), 16
- Kiev (*DaylightSavingTime*), 16
- Kigali (*DaylightSavingTime*), 16
- Kinshasa (*DaylightSavingTime*), 16
- Kiritimati (*DaylightSavingTime*),
16
- Knox (*DaylightSavingTime*), 16
- Kosrae (*DaylightSavingTime*), 16
- Krasnoyarsk (*DaylightSavingTime*),
16
- Kuala_Lumpur
(*DaylightSavingTime*), 16
- KualaLumpur (*DaylightSavingTime*),
16
- Kuching (*DaylightSavingTime*), 16
- Kuwait (*DaylightSavingTime*), 16
- Kwajalein (*DaylightSavingTime*), 16
- La_Paz (*DaylightSavingTime*), 16
- La_Rioja (*DaylightSavingTime*), 16
- LaborDay (*holidayDate*), 26
- Lagos (*DaylightSavingTime*), 16
- lastDay (*firstDay*), 21
- length, 33
- Libreville (*DaylightSavingTime*),
16
- Lima (*DaylightSavingTime*), 16
- Lindeman (*DaylightSavingTime*), 16
- lines.timeDate (*plot-methods*), 39
- Lisbon (*DaylightSavingTime*), 16
- listFinCenter, 34, 36
- listHolidays, 35
- Ljubljana (*DaylightSavingTime*), 16
- Lome (*DaylightSavingTime*), 16
- London (*DaylightSavingTime*), 16
- Longyearbyen
(*DaylightSavingTime*), 16
- Lord_Howe (*DaylightSavingTime*), 16

- Los_Angeles (*DaylightSavingTime*),
16
- LosAngeles (*DaylightSavingTime*),
16
- Louisville (*DaylightSavingTime*),
16
- Luanda (*DaylightSavingTime*), 16
- Lubumbashi (*DaylightSavingTime*),
16
- Lusaka (*DaylightSavingTime*), 16
- Luxembourg (*DaylightSavingTime*),
16

- Macau (*DaylightSavingTime*), 16
- Maceio (*DaylightSavingTime*), 16
- Madeira (*DaylightSavingTime*), 16
- Madrid (*DaylightSavingTime*), 16
- Magadan (*DaylightSavingTime*), 16
- Mahe (*DaylightSavingTime*), 16
- Majuro (*DaylightSavingTime*), 16
- Makassar (*DaylightSavingTime*), 16
- Malabo (*DaylightSavingTime*), 16
- Maldives (*DaylightSavingTime*), 16
- Malta (*DaylightSavingTime*), 16
- Managua (*DaylightSavingTime*), 16
- Manaus (*DaylightSavingTime*), 16
- Manila (*DaylightSavingTime*), 16
- Maputo (*DaylightSavingTime*), 16
- Marengo (*DaylightSavingTime*), 16
- Mariehamn (*DaylightSavingTime*), 16
- Marigot (*DaylightSavingTime*), 16
- Marquesas (*DaylightSavingTime*), 16
- Martinique (*DaylightSavingTime*),
16
- Maseru (*DaylightSavingTime*), 16
- MassOfArchangels (*holidayDate*), 26
- Mauritius (*DaylightSavingTime*), 16
- Mawson (*DaylightSavingTime*), 16
- max.timeDate (*start*), 46
- Mayotte (*DaylightSavingTime*), 16
- Mazatlan (*DaylightSavingTime*), 16
- Mbabane (*DaylightSavingTime*), 16
- McMurdo (*DaylightSavingTime*), 16
- Melbourne (*DaylightSavingTime*), 16
- Mendoza (*DaylightSavingTime*), 16
- Menominee (*DaylightSavingTime*), 16
- Merida (*DaylightSavingTime*), 16
- Mexico_City (*DaylightSavingTime*),
16

- MexicoCity (*DaylightSavingTime*),
16
- midnightStandard, 35
- Midway (*DaylightSavingTime*), 16
- min.timeDate (*start*), 46
- Minsk (*DaylightSavingTime*), 16
- Miquelon (*DaylightSavingTime*), 16
- Mogadishu (*DaylightSavingTime*), 16
- Monaco (*DaylightSavingTime*), 16
- Moncton (*DaylightSavingTime*), 16
- Monrovia (*DaylightSavingTime*), 16
- Monterrey (*DaylightSavingTime*), 16
- Montevideo (*DaylightSavingTime*),
16
- months, 53
- months.timeDate (*julian*), 32
- Monticello (*DaylightSavingTime*),
16
- Montreal (*DaylightSavingTime*), 16
- Montserrat (*DaylightSavingTime*),
16
- Moscow (*DaylightSavingTime*), 16
- Muscat (*DaylightSavingTime*), 16
- myFinCenter, 36
- myUnits, 37

- Nairobi (*DaylightSavingTime*), 16
- Nassau (*DaylightSavingTime*), 16
- Nauru (*DaylightSavingTime*), 16
- nDay, 37
- Ndjamena (*DaylightSavingTime*), 16
- New_Salem (*DaylightSavingTime*), 16
- New_York (*DaylightSavingTime*), 16
- NewYearsDay (*holidayDate*), 26
- NewYork (*DaylightSavingTime*), 16
- Niamey (*DaylightSavingTime*), 16
- Nicosia (*DaylightSavingTime*), 16
- Nipigon (*DaylightSavingTime*), 16
- Niue (*DaylightSavingTime*), 16
- Nome (*DaylightSavingTime*), 16
- Norfolk (*DaylightSavingTime*), 16
- Noronha (*DaylightSavingTime*), 16
- Nouakchott (*DaylightSavingTime*),
16
- Noumea (*DaylightSavingTime*), 16
- Novosibirsk (*DaylightSavingTime*),
16

- Omsk (*DaylightSavingTime*), 16

- onOrAfter, 38
- onOrBefore (*onOrAfter*), 38
- Ops, 53
- Ops.timeDate (*timeDateMathOps*), 55
- Oral (*DaylightSavingTime*), 16
- Oslo (*DaylightSavingTime*), 16
- Ouagadougou (*DaylightSavingTime*), 16
- Pacific (*DaylightSavingTime*), 16
- Pago_Pago (*DaylightSavingTime*), 16
- Palau (*DaylightSavingTime*), 16
- Palmer (*DaylightSavingTime*), 16
- PalmSunday (*holidayDate*), 26
- Panama (*DaylightSavingTime*), 16
- Pangnirtung (*DaylightSavingTime*), 16
- Paramaribo (*DaylightSavingTime*), 16
- Paris (*DaylightSavingTime*), 16
- Pentecost (*holidayDate*), 26
- PentecostMonday (*holidayDate*), 26
- Perth (*DaylightSavingTime*), 16
- Petersburg (*DaylightSavingTime*), 16
- Phnom_Penh (*DaylightSavingTime*), 16
- Phoenix (*DaylightSavingTime*), 16
- Pitcairn (*DaylightSavingTime*), 16
- plot-methods, 39
- plot.timeDate (*plot-methods*), 39
- Podgorica (*DaylightSavingTime*), 16
- points.timeDate (*plot-methods*), 39
- Ponape (*DaylightSavingTime*), 16
- Pontianak (*DaylightSavingTime*), 16
- Port-au-Prince (*DaylightSavingTime*), 16
- Port_Moresby (*DaylightSavingTime*), 16
- Port_of_Spain (*DaylightSavingTime*), 16
- Porto-Novo (*DaylightSavingTime*), 16
- Porto_Velho (*DaylightSavingTime*), 16
- Prague (*DaylightSavingTime*), 16
- PresentationOfLord (*holidayDate*), 26
- Puerto_Rico (*DaylightSavingTime*), 16
- Pyongyang (*DaylightSavingTime*), 16
- Qatar (*DaylightSavingTime*), 16
- Quinquagesima (*holidayDate*), 26
- Qyzylorda (*DaylightSavingTime*), 16
- Rainy_River (*DaylightSavingTime*), 16
- range.timeDate (*start*), 46
- Rangoon (*DaylightSavingTime*), 16
- Rankin_Inlet (*DaylightSavingTime*), 16
- Rarotonga (*DaylightSavingTime*), 16
- Recife (*DaylightSavingTime*), 16
- Regina (*DaylightSavingTime*), 16
- rep, 40
- Resolute (*DaylightSavingTime*), 16
- Reunion (*DaylightSavingTime*), 16
- rev, 41
- Reykjavik (*DaylightSavingTime*), 16
- Riga (*DaylightSavingTime*), 16
- Rio_Branco (*DaylightSavingTime*), 16
- Rio_Gallegos (*DaylightSavingTime*), 16
- Riyadh (*DaylightSavingTime*), 16
- RogationSunday (*holidayDate*), 26
- Rome (*DaylightSavingTime*), 16
- Rothera (*DaylightSavingTime*), 16
- round, 42, 53
- rulesFinCenter, 43
- Saigon (*DaylightSavingTime*), 16
- Saipan (*DaylightSavingTime*), 16
- Sakhalin (*DaylightSavingTime*), 16
- Samara (*DaylightSavingTime*), 16
- Samarkand (*DaylightSavingTime*), 16
- sample, 43
- San_Juan (*DaylightSavingTime*), 16
- San_Marino (*DaylightSavingTime*), 16
- Santiago (*DaylightSavingTime*), 16
- Santo_Domingo (*DaylightSavingTime*), 16
- Sao_Paulo (*DaylightSavingTime*), 16
- Sao_Tome (*DaylightSavingTime*), 16
- Sarajevo (*DaylightSavingTime*), 16

- Scoresbysund
 - (*DaylightSavingTime*), 16
- Seoul (*DaylightSavingTime*), 16
- Septuagesima (*holidayDate*), 26
- seq.timeDate (*timeSequence*), 56
- Shanghai (*DaylightSavingTime*), 16
- Shiprock (*DaylightSavingTime*), 16
- show, ANY-method (*show-methods*), 44
- show, timeDate-method
 - (*show-methods*), 44
- show-methods, 44
- show.timeDate (*show-methods*), 44
- Simferopol (*DaylightSavingTime*), 16
- Singapore (*DaylightSavingTime*), 16
- Skopje (*DaylightSavingTime*), 16
- Sofia (*DaylightSavingTime*), 16
- SolemnityOfMary (*holidayDate*), 26
- sort, 45, 53
- South_Georgia
 - (*DaylightSavingTime*), 16
- South_Pole (*DaylightSavingTime*), 16
- St_Barthelemy
 - (*DaylightSavingTime*), 16
- St_Helena (*DaylightSavingTime*), 16
- St_Johns (*DaylightSavingTime*), 16
- St_Kitts (*DaylightSavingTime*), 16
- St_Lucia (*DaylightSavingTime*), 16
- St_Thomas (*DaylightSavingTime*), 16
- St_Vincent (*DaylightSavingTime*), 16
- Stanley (*DaylightSavingTime*), 16
- start, 46
- Stockholm (*DaylightSavingTime*), 16
- subset, 47
- summary-methods, 48
- summary.timeDate
 - (*summary-methods*), 48
- Swift_Current
 - (*DaylightSavingTime*), 16
- Sydney (*DaylightSavingTime*), 16
- Syowa (*DaylightSavingTime*), 16
- Sys.timeDate, 48
- Tahiti (*DaylightSavingTime*), 16
- Taipei (*DaylightSavingTime*), 16
- Tallinn (*DaylightSavingTime*), 16
- Tarawa (*DaylightSavingTime*), 16
- Tashkent (*DaylightSavingTime*), 16
- Tbilisi (*DaylightSavingTime*), 16
- Tegucigalpa (*DaylightSavingTime*), 16
- Tehran (*DaylightSavingTime*), 16
- Tell_City (*DaylightSavingTime*), 16
- Thimphu (*DaylightSavingTime*), 16
- Thule (*DaylightSavingTime*), 16
- Thunder_Bay (*DaylightSavingTime*), 16
- Tijuana (*DaylightSavingTime*), 16
- timeCalendar, 49
- timeDate, 50
- timeDate-class, 51
- timeDateMathOps, 55
- timeFirstDayInMonth (*firstDay*), 21
- timeFirstDayInQuarter (*firstDay*), 21
- timeLastDayInMonth (*firstDay*), 21
- timeLastDayInQuarter (*firstDay*), 21
- timeLastNdayInMonth (*nDay*), 37
- timeNdayOnOrAfter (*onOrAfter*), 38
- timeNdayOnOrBefore (*onOrAfter*), 38
- timeNthNdayInMonth (*nDay*), 37
- timeSequence, 56
- Tirane (*DaylightSavingTime*), 16
- Tokyo (*DaylightSavingTime*), 16
- Tongatapu (*DaylightSavingTime*), 16
- Toronto (*DaylightSavingTime*), 16
- Tortola (*DaylightSavingTime*), 16
- TransfigurationOfLord
 - (*holidayDate*), 26
- TrinitySunday (*holidayDate*), 26
- Tripoli (*DaylightSavingTime*), 16
- Truk (*DaylightSavingTime*), 16
- trunc, 53
- trunc (round), 42
- Tucuman (*DaylightSavingTime*), 16
- Tunis (*DaylightSavingTime*), 16
- Ulaanbaatar (*DaylightSavingTime*), 16
- unique, 58
- Urumqi (*DaylightSavingTime*), 16
- USChristmasDay (*holidayDate*), 26
- USColumbusDay (*holidayDate*), 26
- USCPulaskisBirthday
 - (*holidayDate*), 26

- USDecorationMemorialDay
(*holidayDate*), 26
- USElectionDay (*holidayDate*), 26
- USGoodFriday (*holidayDate*), 26
- Ushuaia (*DaylightSavingTime*), 16
- USInaugurationDay (*holidayDate*),
26
- USIndependenceDay (*holidayDate*),
26
- USLaborDay (*holidayDate*), 26
- USLincolnsBirthday (*holidayDate*),
26
- USMemorialDay (*holidayDate*), 26
- USMLKingsBirthday (*holidayDate*),
26
- USNewYearsDay (*holidayDate*), 26
- USPresidentsDay (*holidayDate*), 26
- USThanksgivingDay (*holidayDate*),
26
- USVeteransDay (*holidayDate*), 26
- USWashingtonsBirthday
(*holidayDate*), 26
- Uzhgorod (*DaylightSavingTime*), 16

- Vaduz (*DaylightSavingTime*), 16
- Vancouver (*DaylightSavingTime*), 16
- Vatican (*DaylightSavingTime*), 16
- Vevay (*DaylightSavingTime*), 16
- Vienna (*DaylightSavingTime*), 16
- Vientiane (*DaylightSavingTime*), 16
- Vilnius (*DaylightSavingTime*), 16
- Vincennes (*DaylightSavingTime*), 16
- Vladivostok (*DaylightSavingTime*),
16
- Volgograd (*DaylightSavingTime*), 16
- Vostok (*DaylightSavingTime*), 16

- Wake (*DaylightSavingTime*), 16
- Wallis (*DaylightSavingTime*), 16
- Warsaw (*DaylightSavingTime*), 16
- whichFormat, 59
- Whitehorse (*DaylightSavingTime*),
16
- Winamac (*DaylightSavingTime*), 16
- Windhoek (*DaylightSavingTime*), 16
- window, 59
- Winnipeg (*DaylightSavingTime*), 16

- Yakutat (*DaylightSavingTime*), 16
- Yakutsk (*DaylightSavingTime*), 16
- Yekaterinburg
(*DaylightSavingTime*), 16
- Yellowknife (*DaylightSavingTime*),
16
- Yerevan (*DaylightSavingTime*), 16

- Zagreb (*DaylightSavingTime*), 16
- Zaporozhye (*DaylightSavingTime*),
16
- Zurich (*DaylightSavingTime*), 16