

# Package ‘timeDate’

January 2, 2012

**Version** 2131.00

**Revision** 5145

**Date** 2011-10-24

**Title** Rmetrics - Chronological and Calendarical Objects

**Author** Diethelm Wuertz, Yohan Chalabi and Martin Maechler with contributions from Joe W. Byers, and others

**Depends** R (>= 2.10.0), graphics, utils, stats, methods

**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.

**LazyData** yes

**License** GPL (>= 2)

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

**Repository** CRAN

**Date/Publication** 2011-10-24 11:17:41

## R topics documented:

timeDate-package . . . . .	3
align . . . . .	13
as.timeDate . . . . .	14
blockStart . . . . .	15
c . . . . .	16

currentYear . . . . .	17
DaylightSavingTime . . . . .	17
dayOfWeek . . . . .	18
dayOfYear . . . . .	19
diff . . . . .	20
difftimeDate . . . . .	21
Easter . . . . .	21
finCenter . . . . .	22
firstDay . . . . .	23
format-methods . . . . .	25
holiday . . . . .	26
holidayDate . . . . .	28
holidayLONDON . . . . .	31
holidayNERC . . . . .	32
holidayNYSE . . . . .	32
holidayTSX . . . . .	33
holidayZURICH . . . . .	34
is.na-methods . . . . .	34
isBizday . . . . .	35
isRegular . . . . .	36
isWeekday . . . . .	37
julian . . . . .	38
kurtosis . . . . .	39
length . . . . .	40
listFinCenter . . . . .	41
listHolidays . . . . .	42
midnightStandard . . . . .	42
myFinCenter . . . . .	43
myUnits . . . . .	44
names-methods . . . . .	44
nDay . . . . .	45
onOrAfter . . . . .	46
periods . . . . .	47
plot-methods . . . . .	48
rep . . . . .	49
rev . . . . .	50
RmetricsOptions . . . . .	51
round . . . . .	51
rulesFinCenter . . . . .	52
sample . . . . .	53
show-methods . . . . .	53
skewness . . . . .	54
sort . . . . .	55
start . . . . .	56
subset . . . . .	57
summary-methods . . . . .	58
Sys.timeDate . . . . .	58
timeCalendar . . . . .	59

timeDate . . . . .	60
timeDate-class . . . . .	61
timeDateMathOps . . . . .	65
timeSequence . . . . .	67
unique . . . . .	68
whichFormat . . . . .	69
window . . . . .	70

**Index 71**

---

timeDate-package      *Utilities and Tools Package*

---

**Description**

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

**Overview of Topics**

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

The 'timeDate' class fulfils the conventions of the ISO 8601 standard as well as of the ANSI C and POSIX standards. Beyond these standards 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"
getRmetricsOptions("myFinCenter")

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

# Create a 'timeDate' object
timeDate(charvec)

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

# if the 'timeDate' was recorded in a different financial center, it
# will be automatically converted to your financial center,
# i.e. "Zurich".
timeDate(charvec, zone = "Tokyo")

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

```

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

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

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

Examples:

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

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

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

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

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

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

Examples:

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

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

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

## 2. Operations on 'timeDate' Objects

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

### Math Operations

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

Examples:

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

# One Hour Later:
now + 3600

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

### Lagging

You can generate suitable lagged and iterated differences:

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

Examples:

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

# Number of days in months and total 2008:
diff(tC)
sum(as.integer(diff(tC)))
```

### Rounding and Truncating

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

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

Examples:

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

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

### Subsetting

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

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

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

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

Examples:

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

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

```

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

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

```

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

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

Examples:

```

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

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

```

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

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

Examples:

```

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

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

```

### Coercions and Transformations

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

```

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

```

```

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

```

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

### Concatenations and Reorderings

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

```

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

```

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

financial center of the first time stamp used in the concatenation:

Examples:

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

# Rordering:
tC = timeCalendar(); tC
tS = sample(tC); tS
t0 = sort(tS); t0
tV = rev(t0); tV
tU = unique(c(tS, tS)); tU
```

### 3. Daylight Saving Time and Financial Centers

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

#### Setting Financial Centers

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

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

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

Examples:

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

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

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

### List of Financial Centers

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

Examples:

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

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

### DST Rules

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

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

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

Examples:

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

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

## 3. Holidays and Holiday Calendars

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

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

**Special Dates:**

The implemented functions can compute the last day in a given month and year, the dates in a month that is a n-day (e.g. n = Sun) on or after a given date, the dates in a month that is a n-day on or before a specified date, the n-th occurrences of a n-day for a specified year/month vectors, or the last n-day for a specified year/month value or vector.

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

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

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

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

**Holidays:**

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

Examples:

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

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

**Holiday Calendars:**

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

We would like to 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.

---

align	<i>Making a 'timeDate' object unique</i>
-------	--

---

## Description

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

## Usage

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

## Arguments

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

## Value

returns an object of class timeDate.

## Examples

```
## align -

# Align Bi-Weekly with a 3 Days Offset
tC = timeCalendar()
tC
# align(tC, by = "2w", offset = "3d")
```

---

as.timeDate                      *Any to 'timeDate' Coercion*

---

## Description

Coerce and transform objects of class 'timeDate'.

## Usage

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

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

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

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

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

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

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

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

## Arguments

FinCenter	a character with the the location of the financial center named as "continent/city".
method	a character string denoting the method how to determine the dates.
tz	inputs the time zone to POSIX objects, i.e. the time zone, zone, or financial center string, FinCenter, as used by timeDate objects.
units	a character string denoting the date/time units in which the results are desired.

x                    an object of class timeDate.  
 zone                the time zone or financial center where the data were recorded.  
 ...                 arguments passed to other methods.

**Value**

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

**Examples**

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

## Convert 'timeDate' to a character strings:
  as.character(tC)

## 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 timeDate.

**Details**

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

**Value**

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	<i>Current Year</i>
-------------	---------------------

---

**Description**

A variable with the current year.

**Note**

It is not allowed to change this variable.

**Examples**

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

---

DaylightSavingTime	<i>Daylight Saving Time Rules</i>
--------------------	-----------------------------------

---

**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 synonymes available "Pacific" for Los Angeles and "Eastern" for New York. specific time zones (AST, CET, CST, EET, EST, MST and PST) are also available.

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

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

**Examples**

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

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

---

dayOfWeek

*Day of the Week*

---

**Description**

returns the 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 = getRmetricsOptions("currentYear"), shift = 0)
```

**Arguments**

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

**Details**

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

The date of Easter is evaluated by a complex procedure whose detailed explanation goes beyond this description. The reason that the calculation is so complicate is, because the date of Easter is linked to (an inaccurate version of) the Hebrew calendar. But nevertheless a short answer to the question "When is Easter?" is the following: Easter Sunday is the first Sunday after the first full moon after vernal equinox. For the long answer we refer to Toendering (1998).

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

**Value**

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

**Examples**

```
## Easter -
# Current Year:
Easter()

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

---

 finCenter

*Financial Center of a timeDate object*


---

**Description**

Print or assign new financial center to a `timeDate` object.

**Usage**

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

**Arguments**

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

**See Also**

listFinCenter

**Examples**

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

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

---

 firstDay

*First and Last Days*


---

**Description**

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

**Usage**

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

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

**Arguments**

charvec a character vector of dates and times.  
 format the format specification of the input character vector.  
 zone the time zone or financial center where the data were recorded.  
 FinCenter a character with the 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"
myFinCenter = getRmetricsOptions("myFinCenter")

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

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

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

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

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

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

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

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

---

format-methods            *Format Methods*

---

### Description

Formats 'timeDate' objects as ISO conform character strings.

### Usage

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

### Arguments

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

### Value

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

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

```

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

```

### Arguments

`year` an integer value or vector of year numbers including the century. These are integers of the form CCYY, e.g. 2000.

### Details

Note that `GBBankHoliday()` returns GB Spring bank holiday. For GB holiday calendar see `holidayGB()`.

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

---

holidayLONDON	<i>London Bank Holidays</i>
---------------	-----------------------------

---

**Description**

Returns bank holidays in London.

**Usage**

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

**Arguments**

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

**Details**

There are 8 bank holidays in Britain every year: New Year's Day, Good Friday, Easter Monday, Spring (May), Last Monday of May, End of Summer (Last Monday) August, Christmas Eve, Christmas Day.

**Value**

returns an object of class `timeDate`.

**Author(s)**

Function contributed by Menon Murali

**Examples**

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

---

holidayNERC

*NERC Holiday Calendar*

---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

returns an object of class `timeDate`.

**Author(s)**

Joe W. Byers

**References**

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

**Examples**

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

---

holidayNYSE

*NYSE Holiday Calendar*

---

**Description**

Returns a holiday calendar for the New York Stock Exchange.

**Usage**

```
holidayNYSE(year = getRmetricsOptions("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	<i>TSX Holiday Calendar</i>
------------	-----------------------------

---

**Description**

Returns a holiday calendar for the Toronto Stock Exchange.

**Usage**

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

**Arguments**

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

**Value**

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 = getRmetricsOptions("currentYear"))
```

**Arguments**

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

**Details**

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

**Value**

returns an object of class `timeDate`.

**Examples**

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

---

is.na-methods	<i>is.na Methods</i>
---------------	----------------------

---

**Description**

is.na methods for 'timeDate' objects.

**Examples**

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

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

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

---

isBizday

*Business and Holidays*

---

**Description**

Tests if a date is a business day or not.

**Usage**

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

**Arguments**

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

**Value**

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

**Examples**

```
## Dates in April, currentYear:
currentYear = getRmetricsOptions("currentYear")
tS = timeSequence(
  from = paste(currentYear, "-03-01", sep = ""),
  to = paste(currentYear, "-04-30", sep = "")
)
tS

## Subset Business Days at NYSE:
holidayNYSE()
isBizday(tS, holidayNYSE())
tS[isBizday(tS, holidayNYSE())]
```

---

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

---

**Description**

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

**Usage**

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

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

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

**Arguments**

<code>x</code>	an R object of class <code>timeDate</code> .
<code>...</code>	arguments to be passed

**Details**

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

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

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

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

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

NOT yet implemented is the case of weekly vectors.

**Value**

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

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

**Examples**

```
## None
```

---

isWeekday	<i>Weekdays and Weekends</i>
-----------	------------------------------

---

**Description**

Tests if a date is a weekday or not.

**Usage**

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

**Arguments**

x	an object of class <code>timeDate</code> .
wday	Specify which days should be considered as weekdays. By default from Mondays to Fridays.

**Value**

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

**Examples**

```
## Dates in April, currentYear:  
currentYear = getRmetricsOptions("currentYear")  
tS = timeSequence(  
  from = paste(currentYear, "-03-01", sep = ""),  
  to = paste(currentYear, "-04-30", sep = "")  
  tS  
  
## Subset of Weekends:  
isWeekend(tS)  
tS[isWeekend(tS)]
```

---

julian *Julian Counts and Calendar Atoms*

---

### Description

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

### Usage

```
## S4 method for signature 'timeDate'
julian(x, origin = timeDate("1970-01-01"),
       units = c("auto", "secs", "mins", "hours", "days", "weeks"),
       zone = NULL, FinCenter = NULL, ...)

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

## S4 method for signature '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)
```

---

kurtosis

*Kurtosis*


---

## Description

Functions to compute kurtosis.

## Usage

```
kurtosis(x, ...)

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

## Arguments

na.rm	a logical. Should missing values be removed?
method	a character string which specifies the method of computation. These are either "moment", "fisher", or "excess". If "excess" is selected, then the value of the kurtosis is computed by the "moment" method and a value of 3 will be subtracted. The "moment" method is based on the definitions of kurtosis for distributions; these forms should be used when resampling (bootstrap or jackknife). The "fisher" method correspond to the usual "unbiased" definition of sample variance, although in the case of kurtosis exact unbiasedness is not possible.
x	a numeric vector or object.
...	arguments to be passed.

## Value

kurtosis  
returns the value of the statistics, a numeric value. An attribute which reports the used method is added.

## See Also

[link{skewness}](#).

**Examples**

```
## mean -  
## var -  
  # Mean, Variance:  
  r = rnorm(100)  
  mean(r)  
  var(r)  
  
## kurtosis -  
  kurtosis(r)
```

---

length	<i>Length of a 'timeDate' Object</i>
--------	--------------------------------------

---

**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:
getRmetricsOptions("myFinCenter")

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

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

---

listHolidays	<i>List of Holidayss</i>
--------------	--------------------------

---

**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	<i>Midnight Standard</i>
------------------	--------------------------

---

**Description**

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

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

**Usage**

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

**Arguments**

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

**Value**

midnightStandard returns a character and midnightStandard2 a [POSIXct](#) object.

**Examples**

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

---

myFinCenter

*myFinCenter Variable*

---

**Description**

A character string with the name of my financial center.

**Note**

Can be modified by the user to his 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:
getRmetricsOptions("myFinCenter")

## Change to another Financial Center:
# setRmetricsOptions(myFinCenter = "Zurich")

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

---

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

---

**Description**

A variable with the frequency of date/units.

**Value**

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

**Examples**

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

---

names-methods	<i>The Names of a timeDate Object</i>
---------------	---------------------------------------

---

**Description**

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

**Usage**

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

**Arguments**

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

**Examples**

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

---

nDay	<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 = "", FinCenter = "")
```

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

### Arguments

charvec	a character vector of dates and times.
nday	an integer vector with entries ranging from 0 (Sunday) to 6 (Saturday).
nth	an integer vector numbering the n-th 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 ( $nth = 1, \dots, 5$ ) 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 = "", FinCenter = "")
```

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

**Arguments**

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

---

 periods

*Rolling periods*


---

**Description**

Returns start and end dates for a rolling periods

**Usage**

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

**Arguments**

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

**Details**

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

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

**Examples**

```
## Create Time Sequence -
x <- timeSequence(from = "2001-01-01", to = "2009-01-01", by = "day")

## Generate Periods -
periods(x, "12m", "1m")
periods(x, "52w", "4w")

## Roll Periodically -
periodicallyRolling(x)

## Roll Monthly -
monthlyRolling(x)
```

**Description**

Plot methods for timeDate objects.

**Usage**

```
## S4 method for signature 'timeDate'  
plot(x, y, ...)  
## S4 method for signature 'timeDate'  
lines(x, y, ...)  
## S4 method for signature 'timeDate'  
points(x, y, ...)  
axis.timeDate(side, x, at, format = NULL, labels = TRUE, ...)
```

**Arguments**

x, y, at	an object of class timeDate.
side	an integer specifying which side of the plot the axis is to be drawn on. The axis is placed as follows: 1=below, 2=left, 3=above and 4=right.
format	format - format string.
labels	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.
...	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)
```

---

RmetricsOptions	<i>Rmetrics Option Settings</i>
-----------------	---------------------------------

---

**Description**

Allow the user to set and examine a variety of global options which affect the way in which Rmetrics functions computes and displays its results.

**Usage**

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

**Arguments**

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

---

round	<i>Rounding and Truncating 'timeDate' Objects</i>
-------	---

---

**Description**

Rounds and truncates objects of class 'timeDate'.

**Usage**

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

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

**Arguments**

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

**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	<i>Financial Centers DST Rules</i>
----------------	------------------------------------

---

**Description**

Returns DST rules for a financial center.

**Usage**

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

**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	<i>Resampling 'timeDate' Objects</i>
--------	--------------------------------------

---

**Description**

Resamples a 'timeDate' object.

**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	<i>Show Methods</i>
--------------	---------------------

---

**Description**

Show methods for 'timeDate' objects.

**Methods**

**object = "ANY"** Generic function.

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

**Examples**

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

---

skewness	<i>Skewness</i>
----------	-----------------

---

**Description**

Functions to compute skewness.

**Usage**

```
skewness(x, ...)

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

**Arguments**

na.rm	a logical. Should missing values be removed?
method	a character string which specifies the method of computation. These are either "moment" or "fisher" The "moment" method is based on the definitions of skewness for distributions; these forms should be used when resampling (bootstrap or jackknife). The "fisher" method correspond to the usual "unbiased" definition of sample variance, although in the case of skewness exact unbiasedness is not possible.
x	a numeric vector or object.
...	arguments to be passed.

**Value**

skewness

returns the value of the statistics, a numeric value. An attribute which reports the used method is added.

**See Also**

link{kurtosis}.

**Examples**

```
## mean -
## var -
# Mean, Variance:
r = rnorm(100)
mean(r)
var(r)

## skewness -
skewness(r)
```

---

 sort

*Sorting 'timeDate' Objects*


---

**Description**

Sorts a 'timeDate' object.

**Usage**

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

**Arguments**

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

**Value**

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 timeDate.
...	[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.

**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 = "")
```

**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	<i>'timeDate' from Calendar Atoms</i>
--------------	---------------------------------------

---

**Description**

Create a 'timeDate' object from calendar atoms.

**Usage**

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

**Arguments**

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

**Value**

returns a S4 object of class "timeDate".

**Examples**

```
## timeCalendar -

# Current Year:
getRmetricsOptions("currentYear")

# 12 months of current year
timeCalendar()

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

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

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

---

timeDate

*'timeDate' Objects from Scratch*


---

**Description**

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

**Usage**

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

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

**Arguments**

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

## 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. Furthermore, 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 the timeDate functions which can also take timeDate and POSIXt objects as inputs, while transforming them between financial centers and/or time zones specified by the arguments zone and FinCenter. Finally the

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

### Tests and Representation of timeDate Objects:

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

### Mathematical Operations:

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

### Transformation of Objects:

Rmetrics has also functions to transform date/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 -

getRmetricsOptions("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)

```

**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**

```
## S4 method for signature 'timeDate,timeDate'
Ops(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	<i>Regularly spaced 'timeDate' objects</i>
--------------	--

---

### Description

Create a regularly spaced object of class `timeDate`.

### Usage

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

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

### Arguments

<code>from, to</code>	starting date, required, and end date, optional. If supplied, <code>to</code> must be after (later than) <code>from</code> .
<code>by</code>	<ul style="list-style-type: none"> <li>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".</li> <li>character string "quarter" that is equivalent to "3 months".</li> <li>A number, taken to be in seconds.</li> <li>A object of class 'difftime'</li> </ul>
<code>length.out</code>	<code>length.out</code> integer, optional. Desired length of the sequence, if specified "to" will be ignored.
<code>along.with</code>	Take the length from the length of this argument.
<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".
<code>...</code>	arguments passed to other methods.

### Value

an object of class `"timeDate"`.

### Note

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

**Examples**

```
## timeSequence -

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

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

## observe "switch to summer time":
timeSequence(from = "2004-03-12", to = "2004-04-11",
  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)
```

---

whichFormat

*Format Recognition*

---

## Description

Trys to recognize the date/time format.

## Usage

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

## Arguments

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

## Value

returns a format string.

## Examples

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

---

window

*Time Windows*

---

### **Description**

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

### **Usage**

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

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

### **Arguments**

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

### **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**

- align, 13
- as.timeDate, 14
- blockStart, 15
- c, 16
- currentYear, 17
- dayOfWeek, 18
- dayOfYear, 19
- diff, 20
- difftimeDate, 21
- Easter, 21
- firstDay, 23
- format-methods, 25
- holiday, 26
- holidayDate, 28
- holidayLONDON, 31
- holidayNERC, 32
- holidayNYSE, 32
- holidayTSX, 33
- holidayZURICH, 34
- is.na-methods, 34
- isBizday, 35
- isRegular, 36
- isWeekday, 37
- julian, 38
- length, 40
- listFinCenter, 41
- listHolidays, 42
- midnightStandard, 42
- myFinCenter, 43
- myUnits, 44
- nDay, 45
- onOrAfter, 46
- plot-methods, 48
- rep, 49
- rev, 50
- round, 51
- rulesFinCenter, 52
- sample, 53

- show-methods, 53
- sort, 55
- start, 56
- subset, 57
- summary-methods, 58
- Sys.timeDate, 58
- timeCalendar, 59
- timeDate, 60
- timeDate-class, 61
- timeDateMathOps, 65
- timeSequence, 67
- unique, 68
- whichFormat, 69
- window, 70

## \*Topic **data**

- DaylightSavingTime, 17

## \*Topic **hplot**

- plot-methods, 48

## \*Topic **package**

- timeDate-package, 3

## \*Topic **univar**

- kurtosis, 39
- skewness, 54

- +, numeric, timeDate-method (timeDateMathOps), 65
- +, timeDate, numeric-method (timeDateMathOps), 65
- +, timeDate, timeDate-method (timeDateMathOps), 65
- , numeric, timeDate-method (timeDateMathOps), 65
- , timeDate, numeric-method (timeDateMathOps), 65
- , timeDate, timeDate-method (timeDateMathOps), 65
- [, timeDate, ANY, missing-method (subset), 57
- [, timeDate, character, missing-method (subset), 57

- [, timeDate, logical, missing-method (subset), 57
- [, timeDate, missing, missing-method (subset), 57
- [, timeDate, numeric, missing-method (subset), 57
- [<- .timeDate (subset), 57
- Abidjan (DaylightSavingTime), 17
- abline, ANY, ANY, ANY, timeDate-method (plot-methods), 48
- Accra (DaylightSavingTime), 17
- Adak (DaylightSavingTime), 17
- Addis\_Ababa (DaylightSavingTime), 17
- Adelaide (DaylightSavingTime), 17
- Aden (DaylightSavingTime), 17
- Advent1st (holidayDate), 28
- Advent2nd (holidayDate), 28
- Advent3rd (holidayDate), 28
- Advent4th (holidayDate), 28
- Algiers (DaylightSavingTime), 17
- align, 13
- align, ANY-method (align), 13
- align, timeDate-method (align), 13
- AllSaints (holidayDate), 28
- AllSouls (holidayDate), 28
- Almaty (DaylightSavingTime), 17
- Amman (DaylightSavingTime), 17
- Amsterdam (DaylightSavingTime), 17
- Anadyr (DaylightSavingTime), 17
- Anchorage (DaylightSavingTime), 17
- Andorra (DaylightSavingTime), 17
- Anguilla (DaylightSavingTime), 17
- Annunciation (holidayDate), 28
- Antananarivo (DaylightSavingTime), 17
- Antigua (DaylightSavingTime), 17
- Any to 'timeDate' Coercion (as.timeDate), 14
- Apia (DaylightSavingTime), 17
- Aqtau (DaylightSavingTime), 17
- Aqtobe (DaylightSavingTime), 17
- Araguaina (DaylightSavingTime), 17
- Aruba (DaylightSavingTime), 17
- as.character.timeDate (as.timeDate), 14
- as.data.frame.timeDate (as.timeDate), 14
- as.Date.timeDate (as.timeDate), 14
- as.double.timeDate (as.timeDate), 14
- as.list.timeDate (as.timeDate), 14
- as.POSIXct, 42
- as.POSIXct.timeDate (as.timeDate), 14
- as.POSIXlt.timeDate (as.timeDate), 14
- as.timeDate, 14
- Ascension (holidayDate), 28
- Ashgabat (DaylightSavingTime), 17
- AshWednesday (holidayDate), 28
- Asmara (DaylightSavingTime), 17
- AssumptionOfMary (holidayDate), 28
- AST (DaylightSavingTime), 17
- Asuncion (DaylightSavingTime), 17
- Athens (DaylightSavingTime), 17
- Atikokan (DaylightSavingTime), 17
- atoms (julian), 38
- atoms, ANY-method (julian), 38
- atoms, timeDate-method (julian), 38
- Auckland (DaylightSavingTime), 17
- axis.timeDate (plot-methods), 48
- Azores (DaylightSavingTime), 17
- Baghdad (DaylightSavingTime), 17
- Bahia (DaylightSavingTime), 17
- Bahrain (DaylightSavingTime), 17
- Baku (DaylightSavingTime), 17
- Bamako (DaylightSavingTime), 17
- Bangkok (DaylightSavingTime), 17
- Bangui (DaylightSavingTime), 17
- Banjul (DaylightSavingTime), 17
- Barbados (DaylightSavingTime), 17
- Beirut (DaylightSavingTime), 17
- Belem (DaylightSavingTime), 17
- Belgrade (DaylightSavingTime), 17
- Belize (DaylightSavingTime), 17
- Berlin (DaylightSavingTime), 17
- Bermuda (DaylightSavingTime), 17
- BirthOfVirginMary (holidayDate), 28
- Bishkek (DaylightSavingTime), 17
- Bissau (DaylightSavingTime), 17
- Blanc-Sablon (DaylightSavingTime), 17
- Blantyre (DaylightSavingTime), 17
- blockEnd (blockStart), 15
- blockStart, 15
- Boa\_Vista (DaylightSavingTime), 17
- Bogota (DaylightSavingTime), 17
- Boise (DaylightSavingTime), 17
- BoxingDay (holidayDate), 28
- Bratislava (DaylightSavingTime), 17
- Brazzaville (DaylightSavingTime), 17
- Brisbane (DaylightSavingTime), 17
- Broken\_Hill (DaylightSavingTime), 17

- Brunei (DaylightSavingTime), 17
- Brussels (DaylightSavingTime), 17
- Bucharest (DaylightSavingTime), 17
- Budapest (DaylightSavingTime), 17
- Buenos\_Aires (DaylightSavingTime), 17
- BuenosAires (DaylightSavingTime), 17
- Bujumbura (DaylightSavingTime), 17
- c, 16
- CACanadaDay (holidayDate), 28
- CACivicProvincialHoliday (holidayDate), 28
- CAFamilyDay (holidayDate), 28
- Cairo (DaylightSavingTime), 17
- CALabourDay (holidayDate), 28
- Calcutta (DaylightSavingTime), 17
- Cambridge\_Bay (DaylightSavingTime), 17
- Campo\_Grande (DaylightSavingTime), 17
- Canary (DaylightSavingTime), 17
- Cancun (DaylightSavingTime), 17
- Cape\_Verde (DaylightSavingTime), 17
- Caracas (DaylightSavingTime), 17
- CaRemembranceDay (holidayDate), 28
- Casablanca (DaylightSavingTime), 17
- Casey (DaylightSavingTime), 17
- Catamarca (DaylightSavingTime), 17
- CAThanksgivingDay (holidayDate), 28
- CAVictoriaDay (holidayDate), 28
- Cayenne (DaylightSavingTime), 17
- Cayman (DaylightSavingTime), 17
- CelebrationOfHolyCross (holidayDate), 28
- Center (DaylightSavingTime), 17
- CET (DaylightSavingTime), 17
- Ceuta (DaylightSavingTime), 17
- Chagos (DaylightSavingTime), 17
- CHAscension (holidayDate), 28
- Chatham (DaylightSavingTime), 17
- CHBerchtoldsDay (holidayDate), 28
- CHConfederationDay (holidayDate), 28
- Chicago (DaylightSavingTime), 17
- Chihuahua (DaylightSavingTime), 17
- Chisinau (DaylightSavingTime), 17
- CHKnabenschiessen (holidayDate), 28
- Choibalsan (DaylightSavingTime), 17
- Chongqing (DaylightSavingTime), 17
- Christmas (DaylightSavingTime), 17
- ChristmasDay (holidayDate), 28
- ChristmasEve (holidayDate), 28
- ChristTheKing (holidayDate), 28
- CHSechselaeuten (holidayDate), 28
- class, 59, 67
- Cocos (DaylightSavingTime), 17
- coerce, ANY, timeDate-method (as.timeDate), 14
- coerce, Date, timeDate-method (as.timeDate), 14
- coerce, POSIXt, timeDate-method (as.timeDate), 14
- coerce, timeDate, character-method (as.timeDate), 14
- coerce, timeDate, data.frame-method (as.timeDate), 14
- coerce, timeDate, Date-method (as.timeDate), 14
- coerce, timeDate, list-method (as.timeDate), 14
- coerce, timeDate, numeric-method (as.timeDate), 14
- coerce, timeDate, POSIXct-method (as.timeDate), 14
- coerce, timeDate, POSIXlt-method (as.timeDate), 14
- Colombo (DaylightSavingTime), 17
- Comoro (DaylightSavingTime), 17
- Conakry (DaylightSavingTime), 17
- Copenhagen (DaylightSavingTime), 17
- Cordoba (DaylightSavingTime), 17
- CorpusChristi (holidayDate), 28
- Costa\_Rica (DaylightSavingTime), 17
- CST (DaylightSavingTime), 17
- Cuiaba (DaylightSavingTime), 17
- Curacao (DaylightSavingTime), 17
- currentYear, 17
- Currie (DaylightSavingTime), 17
- cut.timeDate (window), 70
- Dakar (DaylightSavingTime), 17
- Damascus (DaylightSavingTime), 17
- Danmarkshavn (DaylightSavingTime), 17
- Dar\_es\_Salaam (DaylightSavingTime), 17
- Darwin (DaylightSavingTime), 17
- Davis (DaylightSavingTime), 17
- Dawson (DaylightSavingTime), 17
- Dawson\_Creek (DaylightSavingTime), 17
- DaylightSavingTime, 17
- dayOfWeek, 18, 19
- dayOfYear, 18, 19
- DEAscension (holidayDate), 28

- DEChristmasEve (holidayDate), 28
- DECorpusChristi (holidayDate), 28
- DEGermanUnity (holidayDate), 28
- DENewYearsEve (holidayDate), 28
- Denver (DaylightSavingTime), 17
- Detroit (DaylightSavingTime), 17
- Dhaka (DaylightSavingTime), 17
- diff, 20, 63
- difftimeDate, 21, 63
- Dili (DaylightSavingTime), 17
- Djibouti (DaylightSavingTime), 17
- Dominica (DaylightSavingTime), 17
- Douala (DaylightSavingTime), 17
- Dubai (DaylightSavingTime), 17
- Dublin (DaylightSavingTime), 17
- DumontDUrville (DaylightSavingTime), 17
- Dushanbe (DaylightSavingTime), 17
- Easter, 21
- EasterMonday (holidayDate), 28
- Eastern (DaylightSavingTime), 17
- EasterSunday (holidayDate), 28
- Edmonton (DaylightSavingTime), 17
- EET (DaylightSavingTime), 17
- Efate (DaylightSavingTime), 17
- Eirunepe (DaylightSavingTime), 17
- El\_Aaiun (DaylightSavingTime), 17
- El\_Salvador (DaylightSavingTime), 17
- end (start), 56
- Enderbury (DaylightSavingTime), 17
- Epiphany (holidayDate), 28
- EST (DaylightSavingTime), 17
- Eucla (DaylightSavingTime), 17
- Fakaofu (DaylightSavingTime), 17
- Faroe (DaylightSavingTime), 17
- Fiji (DaylightSavingTime), 17
- finCenter, 22
- finCenter, timeDate-method (finCenter), 22
- finCenter<- (finCenter), 22
- finCenter<- , timeDate-method (finCenter), 22
- firstDay, 23
- format (format-methods), 25
- format-methods, 25
- Fortaleza (DaylightSavingTime), 17
- FRAllSaints (holidayDate), 28
- Frankfurt (DaylightSavingTime), 17
- FRArmisticeDay (holidayDate), 28
- FRAscension (holidayDate), 28
- FRAssumptionVirginMary (holidayDate), 28
- FRBastilleDay (holidayDate), 28
- Freetown (DaylightSavingTime), 17
- frequency (isRegular), 36
- frequency, timeDate-method (isRegular), 36
- FRFetDeLaVictoire1945 (holidayDate), 28
- Funafuti (DaylightSavingTime), 17
- Gaborone (DaylightSavingTime), 17
- Galapagos (DaylightSavingTime), 17
- Gambier (DaylightSavingTime), 17
- Gaza (DaylightSavingTime), 17
- GBBankHoliday (holidayDate), 28
- GBMayDay (holidayDate), 28
- GBMilleniumDay (holidayDate), 28
- GBSummerBankHoliday (holidayDate), 28
- getDataPart, timeDate-method (timeDate), 60
- getRmetricsOption (RmetricsOptions), 51
- getRmetricsOptions (RmetricsOptions), 51
- Gibraltar (DaylightSavingTime), 17
- Glace\_Bay (DaylightSavingTime), 17
- Godthab (DaylightSavingTime), 17
- GoodFriday (holidayDate), 28
- Goose\_Bay (DaylightSavingTime), 17
- Grand\_Turk (DaylightSavingTime), 17
- Grenada (DaylightSavingTime), 17
- grep, 41, 42
- Guadalcanal (DaylightSavingTime), 17
- Guadeloupe (DaylightSavingTime), 17
- Guam (DaylightSavingTime), 17
- Guatemala (DaylightSavingTime), 17
- Guayaquil (DaylightSavingTime), 17
- Guernsey (DaylightSavingTime), 17
- Guyana (DaylightSavingTime), 17
- Halifax (DaylightSavingTime), 17
- Harare (DaylightSavingTime), 17
- Harbin (DaylightSavingTime), 17
- Havana (DaylightSavingTime), 17
- Helsinki (DaylightSavingTime), 17
- Hermosillo (DaylightSavingTime), 17
- Hobart (DaylightSavingTime), 17
- holiday, 26
- holidayDate, 28
- holidayLONDON, 31

- holidayNERC, 32
- holidayNYSE, 32
- holidayTSX, 33
- holidayZURICH, 34
- Hong\_Kong (DaylightSavingTime), 17
- HongKong (DaylightSavingTime), 17
- Honolulu (DaylightSavingTime), 17
- Hovd (DaylightSavingTime), 17
- Indianapolis (DaylightSavingTime), 17
- initialize, timeDate-method (timeDate), 60
- Inuvik (DaylightSavingTime), 17
- Iqaluit (DaylightSavingTime), 17
- Irkutsk (DaylightSavingTime), 17
- is.na, timeDate-method (is.na-methods), 34
- is.na-methods, 34
- isBizday, 35
- isDaily (isRegular), 36
- isDaily, timeDate-method (isRegular), 36
- isHoliday (isBizday), 35
- Isle\_of\_Man (DaylightSavingTime), 17
- isMonthly (isRegular), 36
- isMonthly, timeDate-method (isRegular), 36
- isQuarterly (isRegular), 36
- isQuarterly, timeDate-method (isRegular), 36
- isRegular, 36
- isRegular, timeDate-method (isRegular), 36
- Istanbul (DaylightSavingTime), 17
- isWeekday, 37
- isWeekend (isWeekday), 37
- ITAllSaints (holidayDate), 28
- ITAssumptionOfVirginMary (holidayDate), 28
- ITEpiphany (holidayDate), 28
- ITImmaculateConception (holidayDate), 28
- ITLiberationDay (holidayDate), 28
- ITStAmrose (holidayDate), 28
- Jakarta (DaylightSavingTime), 17
- Jamaica (DaylightSavingTime), 17
- Jayapura (DaylightSavingTime), 17
- Jersey (DaylightSavingTime), 17
- Jerusalem (DaylightSavingTime), 17
- Johannesburg (DaylightSavingTime), 17
- Johnston (DaylightSavingTime), 17
- JPAutumnalEquinox (holidayDate), 28
- JPBankHolidayDec31 (holidayDate), 28
- JPBankHolidayJan2 (holidayDate), 28
- JPBankHolidayJan3 (holidayDate), 28
- JPBunkaNoHi (holidayDate), 28
- JPChildrensDay (holidayDate), 28
- JPComingOfAgeDay (holidayDate), 28
- JPConstitutionDay (holidayDate), 28
- JPEmperorsBirthday (holidayDate), 28
- JPGantan (holidayDate), 28
- JPGreeneryDay (holidayDate), 28
- JPHealthandSportsDay (holidayDate), 28
- JPKeirouNOhi (holidayDate), 28
- JPKenkokuKinenNoHi (holidayDate), 28
- JPKenpouKinenBi (holidayDate), 28
- JPKinrouKanshaNoHi (holidayDate), 28
- JKodomoNoHi (holidayDate), 28
- JKokuminNoKyujitu (holidayDate), 28
- JPMarineDay (holidayDate), 28
- JPMidoriNoHi (holidayDate), 28
- JPNatFoundationDay (holidayDate), 28
- JPNationalCultureDay (holidayDate), 28
- JPNationHoliday (holidayDate), 28
- JPNewYearsDay (holidayDate), 28
- JPRespectForTheAgedDay (holidayDate), 28
- JPSeijinNoHi (holidayDate), 28
- JPShuubunNoHi (holidayDate), 28
- JPTaiikuNoHi (holidayDate), 28
- JPTennouTanjyouBi (holidayDate), 28
- JPThanksgivingDay (holidayDate), 28
- JPUmiNoHi (holidayDate), 28
- JPVernalEquinox (holidayDate), 28
- Jujuy (DaylightSavingTime), 17
- julian, 38, 63
- julian, timeDate-method (julian), 38
- Juneau (DaylightSavingTime), 17
- Kabul (DaylightSavingTime), 17
- Kaliningrad (DaylightSavingTime), 17
- Kamchatka (DaylightSavingTime), 17
- Kampala (DaylightSavingTime), 17
- Karachi (DaylightSavingTime), 17
- Kashgar (DaylightSavingTime), 17
- Katmandu (DaylightSavingTime), 17
- Kerguelen (DaylightSavingTime), 17
- Khartoum (DaylightSavingTime), 17
- Kiev (DaylightSavingTime), 17
- Kigali (DaylightSavingTime), 17

- Kinshasa (DaylightSavingTime), 17
- Kiritimati (DaylightSavingTime), 17
- Knox (DaylightSavingTime), 17
- Kosrae (DaylightSavingTime), 17
- Krasnoyarsk (DaylightSavingTime), 17
- Kuala\_Lumpur (DaylightSavingTime), 17
- KualaLumpur (DaylightSavingTime), 17
- Kuching (DaylightSavingTime), 17
- kurtosis, 39
- Kuwait (DaylightSavingTime), 17
- Kwajalein (DaylightSavingTime), 17
  
- La\_Paz (DaylightSavingTime), 17
- La\_Rioja (DaylightSavingTime), 17
- LaborDay (holidayDate), 28
- Lagos (DaylightSavingTime), 17
- lastDay (firstDay), 23
- length, 40
- Libreville (DaylightSavingTime), 17
- Lima (DaylightSavingTime), 17
- Lindeman (DaylightSavingTime), 17
- lines, timeDate-method (plot-methods), 48
- Lisbon (DaylightSavingTime), 17
- listFinCenter, 41, 43
- listHolidays, 42
- Ljubljana (DaylightSavingTime), 17
- Lome (DaylightSavingTime), 17
- London (DaylightSavingTime), 17
- Longyearbyen (DaylightSavingTime), 17
- Lord\_Howe (DaylightSavingTime), 17
- Los\_Angeles (DaylightSavingTime), 17
- LosAngeles (DaylightSavingTime), 17
- Louisville (DaylightSavingTime), 17
- Luanda (DaylightSavingTime), 17
- Lubumbashi (DaylightSavingTime), 17
- Lusaka (DaylightSavingTime), 17
- Luxembourg (DaylightSavingTime), 17
  
- Macau (DaylightSavingTime), 17
- Maceio (DaylightSavingTime), 17
- Madeira (DaylightSavingTime), 17
- Madrid (DaylightSavingTime), 17
- Magadan (DaylightSavingTime), 17
- Mahe (DaylightSavingTime), 17
- Majuro (DaylightSavingTime), 17
- Makassar (DaylightSavingTime), 17
- Malabo (DaylightSavingTime), 17
- Maldives (DaylightSavingTime), 17
- Malta (DaylightSavingTime), 17
  
- Managua (DaylightSavingTime), 17
- Manaus (DaylightSavingTime), 17
- Manila (DaylightSavingTime), 17
- Maputo (DaylightSavingTime), 17
- Marengo (DaylightSavingTime), 17
- Mariehamn (DaylightSavingTime), 17
- Marigot (DaylightSavingTime), 17
- Marquesas (DaylightSavingTime), 17
- Martinique (DaylightSavingTime), 17
- Maseru (DaylightSavingTime), 17
- MassOfArchangels (holidayDate), 28
- Mauritius (DaylightSavingTime), 17
- Mawson (DaylightSavingTime), 17
- max.timeDate (start), 56
- Mayotte (DaylightSavingTime), 17
- Mazatlan (DaylightSavingTime), 17
- Mbabane (DaylightSavingTime), 17
- McMurdo (DaylightSavingTime), 17
- Melbourne (DaylightSavingTime), 17
- Mendoza (DaylightSavingTime), 17
- Menominee (DaylightSavingTime), 17
- Merida (DaylightSavingTime), 17
- Mexico\_City (DaylightSavingTime), 17
- MexicoCity (DaylightSavingTime), 17
- midnightStandard, 42
- midnightStandard2 (midnightStandard), 42
- Midway (DaylightSavingTime), 17
- min.timeDate (start), 56
- Minsk (DaylightSavingTime), 17
- Miquelon (DaylightSavingTime), 17
- Mogadishu (DaylightSavingTime), 17
- Monaco (DaylightSavingTime), 17
- Moncton (DaylightSavingTime), 17
- Monrovia (DaylightSavingTime), 17
- Monterrey (DaylightSavingTime), 17
- Montevideo (DaylightSavingTime), 17
- monthlyRolling (periods), 47
- months, 63
- months, timeDate-method (julian), 38
- Monticello (DaylightSavingTime), 17
- Montreal (DaylightSavingTime), 17
- Montserrat (DaylightSavingTime), 17
- Moscow (DaylightSavingTime), 17
- MST (DaylightSavingTime), 17
- Muscat (DaylightSavingTime), 17
- myFinCenter, 43
- myUnits, 44
  
- Nairobi (DaylightSavingTime), 17

- names, timeDate-method (names-methods), 44
- names-methods, 44
- names<-, timeDate-method (names-methods), 44
- Nassau (DaylightSavingTime), 17
- Nauru (DaylightSavingTime), 17
- nDay, 45
- Ndjamena (DaylightSavingTime), 17
- New\_Salem (DaylightSavingTime), 17
- New\_York (DaylightSavingTime), 17
- NewYearsDay (holidayDate), 28
- NewYork (DaylightSavingTime), 17
- Niamey (DaylightSavingTime), 17
- Nicosia (DaylightSavingTime), 17
- Nipigon (DaylightSavingTime), 17
- Niue (DaylightSavingTime), 17
- Nome (DaylightSavingTime), 17
- Norfolk (DaylightSavingTime), 17
- Noronha (DaylightSavingTime), 17
- Nouakchott (DaylightSavingTime), 17
- Noumea (DaylightSavingTime), 17
- Novosibirsk (DaylightSavingTime), 17
- Omsk (DaylightSavingTime), 17
- onOrAfter, 46
- onOrBefore (onOrAfter), 46
- Ops, 63
- Ops, timeDate, timeDate-method (timeDateMathOps), 65
- Oral (DaylightSavingTime), 17
- Oslo (DaylightSavingTime), 17
- Ouagadougou (DaylightSavingTime), 17
- Pacific (DaylightSavingTime), 17
- Pago\_Pago (DaylightSavingTime), 17
- Palau (DaylightSavingTime), 17
- Palmer (DaylightSavingTime), 17
- PalmSunday (holidayDate), 28
- Panama (DaylightSavingTime), 17
- Pangnirtung (DaylightSavingTime), 17
- Paramaribo (DaylightSavingTime), 17
- Paris (DaylightSavingTime), 17
- Pentecost (holidayDate), 28
- PentecostMonday (holidayDate), 28
- periodicallyRolling (periods), 47
- periods, 47
- Perth (DaylightSavingTime), 17
- Petersburg (DaylightSavingTime), 17
- Phnom\_Penh (DaylightSavingTime), 17
- Phoenix (DaylightSavingTime), 17
- Pitcairn (DaylightSavingTime), 17
- plot, timeDate-method (plot-methods), 48
- plot-methods, 48
- Podgorica (DaylightSavingTime), 17
- points, timeDate-method (plot-methods), 48
- Ponape (DaylightSavingTime), 17
- Pontianak (DaylightSavingTime), 17
- Port-au-Prince (DaylightSavingTime), 17
- Port\_Moresby (DaylightSavingTime), 17
- Port\_of\_Spain (DaylightSavingTime), 17
- Porto-Novo (DaylightSavingTime), 17
- Porto\_Velho (DaylightSavingTime), 17
- POSIXct, 43
- Prague (DaylightSavingTime), 17
- PresentationOfLord (holidayDate), 28
- PST (DaylightSavingTime), 17
- Puerto\_Rico (DaylightSavingTime), 17
- Pyongyang (DaylightSavingTime), 17
- Qatar (DaylightSavingTime), 17
- Quinquagesima (holidayDate), 28
- Qyzylorda (DaylightSavingTime), 17
- Rainy\_River (DaylightSavingTime), 17
- range.timeDate (start), 56
- Rangoon (DaylightSavingTime), 17
- Rankin\_Inlet (DaylightSavingTime), 17
- Rarotonga (DaylightSavingTime), 17
- Recife (DaylightSavingTime), 17
- Regina (DaylightSavingTime), 17
- rep, 49
- Resolute (DaylightSavingTime), 17
- Reunion (DaylightSavingTime), 17
- rev, 50
- Reykjavik (DaylightSavingTime), 17
- Riga (DaylightSavingTime), 17
- Rio\_Branco (DaylightSavingTime), 17
- Rio\_Gallegos (DaylightSavingTime), 17
- Riyadh (DaylightSavingTime), 17
- RmetricsOptions, 51
- RogationSunday (holidayDate), 28
- Rome (DaylightSavingTime), 17
- Rothera (DaylightSavingTime), 17
- round, 51, 63
- rulesFinCenter, 52

- Saigon (DaylightSavingTime), 17
- Saipan (DaylightSavingTime), 17
- Sakhalin (DaylightSavingTime), 17
- Samara (DaylightSavingTime), 17
- Samarkand (DaylightSavingTime), 17
- sample, 53
- sample, timeDate-method (sample), 53
- San\_Juan (DaylightSavingTime), 17
- San\_Marino (DaylightSavingTime), 17
- Santiago (DaylightSavingTime), 17
- Santo\_Domingo (DaylightSavingTime), 17
- Sao\_Paulo (DaylightSavingTime), 17
- Sao\_Tome (DaylightSavingTime), 17
- Sarajevo (DaylightSavingTime), 17
- Scoresbysund (DaylightSavingTime), 17
- Seoul (DaylightSavingTime), 17
- Septuagesima (holidayDate), 28
- seq, 67
- seq.POSIXt, 67
- seq.timeDate (timeSequence), 67
- setRmetricsOptions (RmetricsOptions), 51
- Shanghai (DaylightSavingTime), 17
- Shiprock (DaylightSavingTime), 17
- show, ANY-method (show-methods), 53
- show, timeDate-method (show-methods), 53
- show-methods, 53
- show.timeDate (show-methods), 53
- Simferopol (DaylightSavingTime), 17
- Singapore (DaylightSavingTime), 17
- skewness, 54
- Skopje (DaylightSavingTime), 17
- Sofia (DaylightSavingTime), 17
- SolemnityOfMary (holidayDate), 28
- sort, 55, 63
- South\_Georgia (DaylightSavingTime), 17
- South\_Pole (DaylightSavingTime), 17
- St\_Barthelemy (DaylightSavingTime), 17
- St\_Helena (DaylightSavingTime), 17
- St\_Johns (DaylightSavingTime), 17
- St\_Kitts (DaylightSavingTime), 17
- St\_Lucia (DaylightSavingTime), 17
- St\_Thomas (DaylightSavingTime), 17
- St\_Vincent (DaylightSavingTime), 17
- Stanley (DaylightSavingTime), 17
- start, 56
- Stockholm (DaylightSavingTime), 17
- strptime, 42
- strptimeDate (timeDate), 60
- subset, 57
- summary-methods, 58
- summary.timeDate (summary-methods), 58
- Swift\_Current (DaylightSavingTime), 17
- Sydney (DaylightSavingTime), 17
- Syowa (DaylightSavingTime), 17
- Sys.timeDate, 58
- Tahiti (DaylightSavingTime), 17
- Taipei (DaylightSavingTime), 17
- Tallinn (DaylightSavingTime), 17
- Tarawa (DaylightSavingTime), 17
- Tashkent (DaylightSavingTime), 17
- Tbilisi (DaylightSavingTime), 17
- Tegucigalpa (DaylightSavingTime), 17
- Tehran (DaylightSavingTime), 17
- Tell\_City (DaylightSavingTime), 17
- Thimphu (DaylightSavingTime), 17
- Thule (DaylightSavingTime), 17
- Thunder\_Bay (DaylightSavingTime), 17
- Tijuana (DaylightSavingTime), 17
- timeCalendar, 59
- timeDate, 60, 67
- timeDate, ANY-method (timeDate), 60
- timeDate, character-method (timeDate), 60
- timeDate, Date-method (timeDate), 60
- timeDate, missing-method (timeDate), 60
- timeDate, numeric-method (timeDate), 60
- timeDate, POSIXt-method (timeDate), 60
- timeDate, timeDate-method (timeDate), 60
- timeDate-class, 61
- timeDate-package, 3
- timeDateMathOps, 65
- timeFirstDayInMonth (firstDay), 23
- timeFirstDayInQuarter (firstDay), 23
- timeLastDayInMonth (firstDay), 23
- timeLastDayInQuarter (firstDay), 23
- timeLastNdayInMonth (nDay), 45
- timeNdayOnOrAfter (onOrAfter), 46
- timeNdayOnOrBefore (onOrAfter), 46
- timeNthNdayInMonth (nDay), 45
- timeSequence, 67
- Tirane (DaylightSavingTime), 17
- Tokyo (DaylightSavingTime), 17
- Tongatapu (DaylightSavingTime), 17
- Toronto (DaylightSavingTime), 17
- Tortola (DaylightSavingTime), 17
- TransfigurationOfLord (holidayDate), 28
- TrinitySunday (holidayDate), 28

Tripoli (DaylightSavingTime), 17  
Truk (DaylightSavingTime), 17  
trunc, 63  
trunc (round), 51  
Tucuman (DaylightSavingTime), 17  
Tunis (DaylightSavingTime), 17  
  
Ulaanbaatar (DaylightSavingTime), 17  
unique, 68  
Urumqi (DaylightSavingTime), 17  
USChristmasDay (holidayDate), 28  
USColumbusDay (holidayDate), 28  
USCPulaskisBirthday (holidayDate), 28  
USDecorationMemorialDay (holidayDate),  
28  
USElectionDay (holidayDate), 28  
USGoodFriday (holidayDate), 28  
Ushuaia (DaylightSavingTime), 17  
USInaugurationDay (holidayDate), 28  
USIndependenceDay (holidayDate), 28  
USLaborDay (holidayDate), 28  
USLincolnsBirthday (holidayDate), 28  
USMemorialDay (holidayDate), 28  
USMLKingsBirthday (holidayDate), 28  
USNewYearsDay (holidayDate), 28  
USPresidentsDay (holidayDate), 28  
USThanksgivingDay (holidayDate), 28  
USVeteransDay (holidayDate), 28  
USWashingtonsBirthday (holidayDate), 28  
Uzhgorod (DaylightSavingTime), 17  
  
Vaduz (DaylightSavingTime), 17  
Vancouver (DaylightSavingTime), 17  
Vatican (DaylightSavingTime), 17  
Vevay (DaylightSavingTime), 17  
Vienna (DaylightSavingTime), 17  
Vientiane (DaylightSavingTime), 17  
Vilnius (DaylightSavingTime), 17  
Vincennes (DaylightSavingTime), 17  
Vladivostok (DaylightSavingTime), 17  
Volgograd (DaylightSavingTime), 17  
Vostok (DaylightSavingTime), 17  
  
Wake (DaylightSavingTime), 17  
Wallis (DaylightSavingTime), 17  
Warsaw (DaylightSavingTime), 17  
whichFormat, 69  
Whitehorse (DaylightSavingTime), 17  
Winamac (DaylightSavingTime), 17  
  
Windhoek (DaylightSavingTime), 17  
window, 70  
Winnipeg (DaylightSavingTime), 17  
  
Yakutat (DaylightSavingTime), 17  
Yakutsk (DaylightSavingTime), 17  
Yekaterinburg (DaylightSavingTime), 17  
Yellowknife (DaylightSavingTime), 17  
Yerevan (DaylightSavingTime), 17  
  
Zagreb (DaylightSavingTime), 17  
Zaporozhye (DaylightSavingTime), 17  
Zurich (DaylightSavingTime), 17