Package 'TimeWarp'

July 22, 2016

Type Package

Title Date Calculations and Manipulation

Version 1.0.15

Date 2016-07-19

Author Tony Plate, Jeffrey Horner, Lars Hansen

Maintainer Tony Plate <tplate@acm.org>

Depends R (>= 2.6)

Suggests scriptests, Holidays

Imports methods

Description Date sequence, relative date calculations, and date manipulation with business days and holidays. Works with Date and POSIXt classes.

License GPL

NeedsCompilation no

Repository CRAN

Date/Publication 2016-07-22 23:21:38

R topics documented:

TimeWarp-package *Date sequence and manipulation with business days and holidays.*

Description

A package for manipulating vectors of class Date. Support for other vectors and classes may be added in the future.

The general idea with the behavior of functions from this package is that they should return an object of the same class as they are given, e.g., `dateWarp()` applied to a vector of dates in character format should return a vector of dates in character format, and `dateWarp()` applied to a `Date` class vector should return a `Date` class vector.

This functionality is currently implemented for `Date`, `character`, `POSIXct` and `POSIXlt` classes. For other classes, functions from this package will currently return a `Date` vector, but that may change as other classes are added.

Version 1.0 of `TimeWarp` does not handle times on `POSIXct` and `POSIXlt`: the functions in `TimeWarp` will return the same type of object stripped of times. This may change in the future.

Author(s)

Jeffrey Horner, Lars Hansen, Tony Plate

Maintainer: Tony Plate <tplate@acm.org>

See Also

dateWarp, dateAlign, dateSeq, dateMatch, and holidays.

The Holidays package loads a database of holidays into the `TimeWarp` package.

pitfalls describes some pitfalls with date class conversions.

On the use of `Date`, `chron` and `POSIXt` classes: Gabor Grothendieck and Thomas Petzoldt. R help desk: Date and time classes in R. http://cran.r-project.org/doc/Rnews/Rnews_2004-1.pdf (Has a helpful table of how to accomplish various tasks with the different classes.)

On the creation and use of the `POSIXt` classes (`POSIXct` and `POSIXlt`): Brian D. Ripley and Kurt Hornik. Date-time classes. http://cran.r-project.org/doc/Rnews/Rnews_2001-2.pdf

Examples

```
library(Holidays)
# View counts of registered holidays by year
sapply(as.character(1998:2012), function(y)
       sapply(allHolidays(), function(h) length(holidays(y, h, silent=TRUE))))

# Find US option expiration dates in 2011 (The pricing day, usually a Friday)

# Technically speaking, standardized options expire on the Saturday
# following the third Friday of the month. The reason that equity and
# index options expire on this day is due to the fact that this day offers
```

```
# the least number of scheduling conflicts, i.e. holidays.

# When an options expiration date falls on a holiday, all trading dates
# are moved earlier. For example, in 2008, options expiration date falls
# on Good Friday. In this situation, options will still expire on
# Saturday following Good Friday -- however the last trading day for
# Equity options will be the Thursday preceding the Good Friday trading
# holiday.

yy <- 2011
(d1 <- dateSeq(paste(yy, '-01-01', sep=''), len=12, by='months'))
(d2 <- dateAlign(d1, by='months', dir=-1))
(d3 <- dateAlign(d2, by='weeks', week.align=5))
(d4 <- dateWarp(d3, 14, by='days'))
(d5 <- dateAlign(d4, by='bizdays@NYSEC', dir=-1))

# Find option expiration dates that have been shifted because they would have
# occured on a holiday
yy <- 1990
d1 <- dateSeq(paste(yy, '-01-01', sep=''), len=288, by='months')
d2 <- dateAlign(d1, by='months', dir=-1)
d3 <- dateAlign(d2, by='weeks', week.align=5)
d4 <- dateWarp(d3, 14, by='days')
d5 <- dateAlign(d4, by='bizdays@NYSEC', dir=-1)
data.frame(holiday=d4, option.expiration=d5)[which(d4 != d5), ]
```

---

dateAlign *Date alignment*

Description

Align a date vector the a day, bizday, month, week or year boundary. `dateAlign()` is a generic, with methods for `character`, `Date`, `POSIXct`, and `POSIXlt`.

Usage

```
dateAlign(x, by = "days", k.by = 1, direction = 1,
          week.align = NULL, holidays = NULL, silent = FALSE,
          optimize.dups=TRUE)
```

Arguments

x	`Date` vector, or a character vector that can be converted to `Date` by `dateParse`.
by	character string with the time unit of the time period. Can be one of `"days"`, `"bizdays"`,`"weeks"`, `"months"`, or `"years"`.
k.by	positive integer giving the number of the by units to align to. Ignored for `"bizdays"` and `"weeks"`.
direction	integer with either -1 or 1, to align to the previous or next time that is an integer number of the `k.by` * by units.

week.align	if not NULL, and by is "weeks", an integer, 0 to 6 with 0 being Sunday, to specify a weekday to align to.
holidays	character string naming the holiday series (see holidays).
silent	logical indicating whether or not to suppress warnings about arguments.
optimize.dups	If TRUE, internally optimize by not performing the same computation multiple times for duplicates. This does not change the return value.

Value

Date vector whose elements are moved up or down (according to direction) so that they lie on integer multiples of k.by * by units of time starting from the beginning of the next bigger time unit (e.g. if by = "days", then align to multiples of k.by days added to the first of the month. Note that for "weeks", and "bizdays", k.by is assumed to be 1 and ignored; "weeks" without week.align is equivalent to "days". Also note that k.by should be a divisor of the number of by units in the next larger time unit, or NA values will result. The class of the returned value is the same as the class of x for character, Date, POSIXct, and POSIXlt. For x of other classes, the class of the returned value is Date, but this may change in the future.

Note

Alignment of dates can be thought of as a partition on date sequences where an input date is aligned to the first date in a partition, if it is not already aligned. The direction of alignment determines which partition to use for the alignment. If the direction is -1 then alignment happens in the partition which the date falls in. If +1 then alignment happens in the partition just after the partition in which the dates falls.

Author(s)

Lars Hansen, Tony Plate

See Also

dateShift, dateWarp, dateMatch, dateParse, dateSeq

Examples

```
dateAlign("2007/12/06", by = "days", k.by = 4, direction = -1)
date <- as.Date("2009/1/1") + -5:5
dateAlign(date, by = "days", silent = FALSE)
dateAlign(date, by = "days", k.by = 3, direction = -1)
dateAlign(date, by = "bizdays", k.by = 1, direction = 1)
library(Holidays)
dateAlign(date, by = "bizdays", k.by = 1, direction = 1, holidays =
"NYSEC")
dateAlign(date, by = "months", k.by = 2, direction = -1)
dateAlign(date, by = "years", k.by = 3, direction = -1)
```

dateDow *Formats a date/time as character and appends the day of week*

Description

Formats a date/time as character and appends the day of week in the current locale. Is a generic with currently on a default method.

Usage

```
dateDow(date)
```

Arguments

date	A vector that can be interpreted as a date.

Details

For character values of `date`, parses the date using `dateParse(date, dross.remove=TRUE)` and converts to day of week using `weekdays()`. For `date` of other classes, calls `weekdays(date)` directly. Weekdays uses the locale `LC_TIME` to determine the language used.

Value

A character vector.

Note

To get the weekday expressed in English, do `Sys.setlocale("LC_TIME", "C")`. (This should affect all language-dependent time formatting.)

Author(s)

Tony Plate <tplate@acm.org>

See Also

`weekdays`

Examples

```
dateDow(Sys.time())
dateDow(Sys.Date())
dateDow('2011-01-01')
```

`dateFormat` *Converts a Date object to character data*

Description

Converts a Date object to character data

Usage

```
dateFormat(date, format = NULL, optimize.dups=TRUE)
```

Arguments

`date`	A vector of dates. Can be character data or some date object – anything that can be handled by `dateParse()`
`format`	A specification of the format. The default will print as "YYYY-MM-DD" under both Windows and Linux.
`optimize.dups`	If `TRUE`, internally optimize by not performing the same computation multiple times for duplicates. This does not change the return value.

Details

Unfortunately, 'format.POSIXct' does not have the same behavior on Linux and Windows. Here are a few for the difference found in 2008:

1. Windows version does not recognize "%y" format.
2. Windows version does not recognize width arguments like "%02d".
3. Windows and Linux does not agree on the meaning of "%Y". Under Windows it means "%04Y"; under Linux it prints with minimal width.

This function tries to provide identical behavior under Linux and Windows and Mac. This function formats any kind of data objects to character strings with the default format "%04Y-%02m-%02d" under both Linux and Windows. It does this by choosing a system-specific default format and then calling `format()`.

Additional format specifications are also provided:

- '%Q' for quarter, as 'Q1', etc (the value returned by `quarters()`
- '%C' for century, always 2 digits (e.g., '20' for 2013)

Value

The formatted dates as a vector of character data.

Author(s)

Tony Plate

Examples

```
dateFormat(as.Date('2001-02-13'), '%Y.%02m.%02d')
```

`dateMatch` *Match Dates in a Table*

Description

Return the indices of dates in a table that match, according to rules "before", "after", etc. `dateMatch()` is a generic, with methods for `character`, `Date`, `POSIXct`, and `POSIXlt`.

Usage

```
dateMatch(x, table, how=c("NA", "before", "after", "nearest", "interp"),
          error.how=c("NA", "drop", "nearest", "stop"),
          nomatch=NA, offset=NULL, value=FALSE, optimize.dups=TRUE)
```

Arguments

`x`	A `"Date"` vector, or a character vector that can be converted to `"Date"`
`table`	A `"Date"` vector, or a character vector that can be converted to `"Date"`. Must be strictly increasing.
`how`	A character string. Determines how values in `x` that do not have exact matches in `table` are handled. `"before"` the element in `table` that is just before `"after"` the element in `table` that is just after `"nearest"` the element in `table` that is nearest `"interp"` an interpolated index `"NA"` return the `nomatch` value For convenience, `how` can specify both `how` and `error.how` separated with a period, e.g., `how="before.nearest"` is equivalent to `how="before"` and `error.how="nearest"`
`error.how`	A character string. Determines how to handle values in `x` that do not have exact matches in `table` and for which the `how` rule fails (e.g. when `how` is one of `"before"`, `"after"`, or `"interp"`). `"NA"` return the "nomatch" value `"drop"` causes non-matched values to be `dropped` `"nearest"` pick the nearest value in `table`. `"stop"` stop with an error. See the note on argument `how` for another way of specifying `error.how`. A value for `error.how` is ignored if the value for `how` has a period in it.
`nomatch`	The value to return for `nomatch` cases. If `value=TRUE`, then nomatch must be a `Date` value, Otherwise it must be a numeric value. `NA` is the default.

offset	If an integer, this offset is added to the computed indices after matching. (Can be an integer value represented as a float.) Non-integer and non-numeric values cause an error. It is possible that later on, character values may be allowed to specify a computed offset to the values in x (e.g., something like "+1 bizdays@NYSEC"). If the result is outside the range of indices of table, NA is returned in those positions.
value	If TRUE, the matching value in table is returned instead of the index.
optimize.dups	If TRUE, internally optimize by not performing the same computation multiple times for duplicates. This does not change the return value.

Details

Uses match and findInterval to perform matching.

Value

The indices of the matches for the elements of x in table, or the actual matching values from table if value==TRUE. In the latter case, the class of the returned value is the same as the class of x for character, Date, POSIXct, and POSIXlt. For x of other classes, the class of the returned value is Date, but this may change in the future.

Examples

```
d1 <- dateParse(c("2001/01/10", "2001/03/12"))
d2 <- dateSeq(dateParse("2001/01/01"), by = "weeks", len = 20)
dateMatch(d1, dateParse(), how = "nearest", error.how = "drop")
dateMatch(d1, dateParse(), how = "nearest", error.how = "stop")
dateMatch(d1, dateParse(), how = "nearest.stop")
dateMatch(d1, d2, how = "after")
dateMatch(d1, d2, how = "after", offset = -3)
dateMatch(dateParse(c("2001/01/10", "2001/01/17", "2001/03/12")),
dateSeq(dateParse("2001/01/01"), by = "weeks", len = 20), how = "after",
offset = 10, value = TRUE)
```

---

dateParse *Date Construction from Character Vectors*

Description

Parse dates, automatically selecting one of three formats, returning a Date vector. The possible formats are:

- yyyymmddno delimiters, 4 digits year, 2 digit month, 2 digit day
- yyyy/[m]m/[d]dwith delimiters, 4 digit year, 1 or 2 digit month, 1 or 2 digit day
- [m]m/[d]d/yyyywith delimiters, 1 or 2 digit month, 1 or 2 digit day, 4 digit year

Delimiters are discovered automatically, but '/' and '-' are recommended.

Differs from Splus timeDate in that it automatically chooses the format and in that it stops or returns NULL if any elements cannot be parsed. (timeDate silently returns NA for elements that cannot be parsed.)

Usage

```
dateParse(x, format = NULL, stop.on.error = TRUE, quick.try = TRUE,
          dross.remove = FALSE, na.strings = c("NA", ""), ymd8 = TRUE,
          use.cache = TRUE, optimize.dups=TRUE)
```

Arguments

x	A character, factor, timeDate or numeric vector.
format	Force the use of this date format for parsing x.
stop.on.error	Should this function stop with an error when x cannot be parse consistently, or should it return NULL?
quick.try	Should this function do a quick try on parsing just few elements of x (with the goal of failing fast)?
dross.remove	Should extra characters around the date be allowed and automatically removed? The extracted date is the first substring that can be found consistently in all elements of x.
na.strings	Strings that should be treated as NA values.
ymd8	Should an 8-digit format with no separators be tried? Default is TRUE (there is potential for confusion with numeric security identifiers, but this is likely to be a problem, supply ymd8 in the particular case).
use.cache	Try matching against cached values instead of using strptime? When this works, it is 10 to 15 times faster.
optimize.dups	If TRUE, internally optimize by not performing the same computation multiple times for duplicates. This does not change the return value.

Details

If any elements of x cannot be interpreted as a valid date this function either returns NULL or stops with an error (depending on the value supplied for the arugment stop.on.error). This is different from the behavior of timeDate() and timeCalandar which return NA elements in their results. This behavior is not appropriate for dateParse() because of its ability to guess the format, and its assumption that all elements have the same format – if different elements had different formats there would not be a unique way of saying which dates were invalid.

Numeric vectors are interpreted as having the date spelled out in digits, e.g., the integer 20010228 is interpreted as the date "2001/02/28".

Value

A Date vector, or NULL.

Examples

```
dateParse("2001-02-14")
dateParse("2/14/2002")
dateParse(c("1962/06/20", "1962/10/30","NA"))
dateParse(c("19620620", "19621030", "NA"), ymd8 = TRUE)
```

```
dateParse(factor(c("2001/01/01", "2001/01/03", "2001/01/01")))
# Possibly unexpected values in conversion from POSIXct to Date
Sys.setenv('TZ'='EST')
x <- as.POSIXct('2011-12-10 16:55:26 EST')+(0:9)*3600
# Date rolls to the next day after 19:00 hours for EST
# (because that it the time the next day is dawning in UTC)
data.frame(x, as.Date(x))
# This is the way to get as.Date() to do the sensible thing
data.frame(x, as.Date(x, tz='EST'))
```

`dateSeq` *Create a sequence of Dates*

Description

Generate a sequence of dates. Based on `seq.Date`, but adds support for business day and holiday sequencing. `dateSeq()` is a generic, with methods for `character`, `Date`, `POSIXct`, and `POSIXlt`.

Usage

```
dateSeq(from = NULL, to = NULL, year = NULL,
        by = "days", k.by = 1, length.out = NULL,
        holidays = NULL, align.by = TRUE, extend = FALSE,
        range = NULL, week.align = NULL)
```

Arguments

`from`	starting value of the sequence, a `Date` object, or number or character string recognized by `dateParse`.
`to`	ending value of the sequence, a `Date` object, or number or character string recognized by `dateParse`.
`year`	an alternative to supplying `from` and `to`, create a sequence of dates from the given year.
`by`	spacing between successive values in the sequence. Can be one of `"days"`, `"bizdays"`, `"weeks"`, `"months"`, or `"years"`. An alternative way to specify by is with a character string that encodes the `k.by`, by, and the named `holidays`, e.g "+1 bizdays@NYSEC" says to create a sequence whose elements are 1 business day apart and exclude NYSEC holidays.
`k.by`	non-zero integer giving the width of the interval between consecutive values in the sequence in terms of the units given in by.
`length.out`	the length of the sequence, before additions and exceptions are included.
`holidays`	character string describing the holidays to exclude from the sequence when `by="bizdays"` (see `holidays`).
`align.by`	if TRUE, adjust the sequence so that each element is on a whole number of the `by * k.by` units.

extend	if TRUE and align.by is also TRUE, instead of making the entire sequence lie between from and to, make it extend just past from and to to the next aligned values.
range	a two-element character or Date vector: an alternate way to specify from and to.
week.align	if by is "weeks", specify the weekday to align to, given as number, 0 to 6 with 0 being Sunday.

Value

A vector of dates. The class of the returned value is the same as the class of from for character, Date, POSIXct, and POSIXlt. For from of other classes, the class of the returned value is Date, but this may change in the future.

Examples

```
dateSeq("2008-12-20", "2009-1-10")
dateSeq("2008-12-20", "2009-1-10", by = "days", k.by = 2)
library(Holidays)
dateSeq("2008-12-20", "2009-1-10", by = "bizdays", holidays = "NYSEC")
dateSeq(from = "1960-01-01", to = "1960-01-20", by = "weeks", week.align
= 0, extend = TRUE)
dateSeq(from = "2000/01/14", length.out = 5, by = "bizdays", holidays = 'NYSEC')
```

---

dateShift *Date shifting*

Description

Shift a date vector a number of days, bizdays, months, weeks or years. dateShift() is a generic, with methods for character, Date, POSIXct, and POSIXlt.

Usage

```
dateShift(x, by = "days", k.by = 1, direction = 1, holidays = NULL,
          silent = FALSE, optimize.dups=TRUE)
```

Arguments

x	Date vector, or a character vector that can be converted to Date by dateParse.
by	character string with the time unit of the shifts. Can be one of "days", "bizdays","weeks", "months", or "years".
k.by	positive integer with the number of by time units to shift.
direction	integer with the direction to shift. A value of 1 for the future, and -1 for the past.
holidays	character string naming the holiday series (see holidays).
silent	logical indicating whether or not to suppress warnings about arguments.
optimize.dups	If TRUE, internally optimize by not performing the same computation multiple times for duplicates. This does not change the return value.

Value

Date vector that is a time shifted version of the input dates. If shifting by "bizdays", weekends and holidays will be skipped. The class of the returned value is the same as the class of x for character, Date, POSIXct, and POSIXlt. For x of other classes, the class of the returned value is Date, but this may change in the future.

Author(s)

Lars Hansen, Tony Plate

See Also

dateAlign, dateWarp, dateMatch, dateParse, dateSeq

Examples

```
dateShift("2007/12/06", by = "days", k.by = 7, direction = -1)
date <- as.Date("2009/1/1") + -5:5
dateShift(date, by = "days", silent = TRUE)
library(Holidays)
dateShift(date, by = "bizdays", k.by = 5, holidays = "NYSEC")
dateShift(date, by = "weeks", k.by = 2)
dateShift(date, by = "months", k.by = "3", direction = "-1")
dateShift(date, by = "years", k.by = 1, direction = 1)
```

---

dateWarp *Date vector shifting and alignment*

Description

Perform multiple shifts and alignments on Date vectors. dateWarp() is a generic, with methods for character, factor, Date, POSIXct, and POSIXlt.

Usage

```
dateWarp(date, spec, holidays = NULL, by = NULL, direction = 1,
         duplicates.keep = TRUE, optimize.dups=TRUE)
dateWarpAppend(date, ..., where = c("sorted", "start", "end"), empty.ok = FALSE,
               duplicates.ok = FALSE)
```

Arguments

date	a Date vector, or a character vector that can be converted to Date by dateParse.
spec	a specification of shifts and alignment tranformations. See 'Details'.
holidays	a character string naming the holiday series (see holidays).
by	how to warp. Can be one of "days", "bizdays", "weeks", "months", or "years". "bizdays" can contain a holiday specification like: "bizdays@NYSEC"

direction	which direction to warp. a numeric value of 1 for the future, and -1 for the past (be careful about using variables for this value, for if it's value is negative and you place a minus sign in front of the variable, you'll go back to the future).
duplicates.keep	logical indicating whether or not to keep duplicate dates.
...	arguments to pass to dateWarp.
where	character string. can be "sorted" to sort the result instead of append, "start" to prepend, or "end" to append.
empty.ok	is it okay for the dates argument to be empty?
duplicates.ok	logical indicating whether or not to keep duplicate dates.
optimize.dups	If TRUE, internally optimize by not performing the same computation multiple times for duplicates. This does not change the return value.

Details

spec specify tranformations in several ways:

- integer vector by which to shift the current Date object. The units of the shift are specified by the by argument. If the shift contains more than one element, this will transform a single Date object into a multiple-element Date object. It is an error to apply a multiple-element shift to a multiple element Date object.
- character data of the form "+3 bizdays@NYSEC", "+3 bizdays", or "+3". 'by' and 'holidays' specifications are extracted from the string as appropriate, and override any previously specified or given in arguments.
- a named list of elements. The elements can be lists or vectors. The names on the elements specify instructions:

  unique the actual value of the list element is ignored. The action is to remove duplicates from the results, i.e., make the dates unique.

  latest the value of the list element is a Date or a character that can be converted to a Date with dateParse. The dates will be clipped to be no later than the specified Date.

  earliest the value of the list element is a Date or a character that can be converted to a Date with dateParse. The dates will be clipped to be no earlier than the specified Date.

  shift the spec list element is a list that will be used as arguments to a call of dateShift. If any are not specified in the list, the values of the by, holidays, and direction arguments given to dateWarp() are passed to dateShift.

  align the spec list element is a list that will be used as arguments to a call of dateAlign. If any are not specified in the list, the values of the by, holidays, and direction arguments given to dateWarp() are passed to dateShift.

  If not all arguments are not provided, the dateWarp arguments will be used instead.

Value

A date vector that is a transformed version of the input dates. Multiple shift or alignment transformations can be specified. If more that one is given, each vill be applied in turn to the result of the previous one. The class of the returned value is the same as the class of date for character, Date, POSIXct, and POSIXlt. For date of other classes, the class of the returned value is Date, but this may change in the future.

Author(s)

Lars Hansen, Tony Plate

See Also

dateAlign, dateShift, dateMatch, dateParse, and dateSeq

The Holidays package contains holidays data, which is registered with the TimeWarp package when the Holidays package is loaded.

Examples

```
library(Holidays)
dates <- dateSeq("2001/12/20", by = 'bizdays', len = 9, holidays = "NYSEC")
dateWarp(dates, -1:1, by = "bizdays", holidays = "NYSEC", duplicates.keep = FALSE)
dateWarp(dates, "+1 bizdays@NYSEC")
dateWarp(dates, list(0:6), by = "bizdays@NYSEC")
dateWarp(dates, list(-1:1, unique = TRUE), by = "bizdays")
dateWarp(dates, list(latest = "2001/12/25"))
x <- dateSeq("2001/01/01", len = 4, by = "weeks")
dateWarp(dates, list(align = list(to = x, how = "after")))
dateWarp(dates, list(shift = list(by = "bizdays", k.by = 2, direction = 1, holidays = "NYSEC"),
                     shift = 2),
         by = "days", direction = -1,
holidays = "NONE")
dateWarp(dates, hol = "NYSEC",
         list(shift = list(k.by = 1, by = "months"),
         shift=list(k.by = -1, by = "bizdays")))
dateWarp(dates, list(align = list(by = "bizdays"),
                     shift = list(by = "months", k.by = 2), holidays = "JPNEX"))

# Options expirations dates in 2008
(d1 <- dateSeq('2008-01-01', len=12, by='months'))
(d2 <- dateAlign(d1, by='months', dir=-1))
(d3 <- dateAlign(d2, by='weeks', week.align=5))
(d4 <- dateWarp(d3, 14, by='days'))
(d5 <- dateAlign(d4, by='bizdays@NYSEC', dir=-1))
# Version that uses a list 'spec' to dateWarp
dateWarp(d1, list(align=list(by='months', dir=-1),
                  align=list(by='weeks', week.align=5),
                  shift=list(by='days', 14),
                  align=list(by='bizdays@NYSEC', dir=-1)))
# In 2008 the March options expiration is a Thursday because Friday was a holiday
dateDow(d5)
```

---

holidays *TimeWarp Holiday Database*

Description

Functions for querying and manipulating the TimeWarp holiday database

Usage

```
holidays(years, type, silent = FALSE)
addToHolidays(type,dates)
registerHolidays(type,dates)
unregisterHolidays(type,dates)
allHolidays()
isHoliday(dates, type)
```

Arguments

years	numeric vector of years for which to return holiday dates.
type	character string, name of the holiday.
dates	a Date vector, or a character vector that can be converted to Date by dateParse.
silent	do not display warnings.

Details

The TimeWarp holidays database is implemented as an internal named, or type'd, list of data.frame's.

To create a new type of holiday, use registerHolidays. unregisterHolidays will delete the holiday named by type, and addToHolidays will add new days to an existing type of holiday.

allHolidays returns a character vector of all the known holiday types.

Value

holidays returns a Date vector of holidays that fall within the years argument for the given holiday type.

addToHolidays and registerHolidays invisibly return a copy of the data.frame for the given type.

unregisterHolidays invisibly returns NULL.

allHolidays returns a character vector of all holiday type's known to the database.

isHoliday returns a logical vector with TRUE of all holidays in input.

Examples

```
# Create a holiday type of New Year days for the 20th century.
registerHolidays('NEWYEAR', as.Date(ISOdate(1900:2000,1,1)))

# Return all New Year days for the 1990's
holidays(1990:2000,'NEWYEAR')

# View counts of registered holidays by year
sapply(as.character(1998:2012), function(y)
       sapply(allHolidays(), function(h) length(holidays(y, h, silent=TRUE))))
```

pitfalls *Pitfalls in date class conversions*

Description

Direct conversion between Date and POSIXt classes using the as.* can give probably undesired results unless great care is taken with supplying appropriate time zones where needed.

It is generally easier to convert dates between Date and POSIXt using character formatted dates as an intermediate representation.

Note that what is described as "not sensible" behavior here is NOT INCONSISTENT with the documentation for as.POSIXct and as.POSIXlt.

Details

```
> # Behavior depends on the timezone of the system
> Sys.setenv(TZ='EST')
> Sys.timezone()
[1] "EST"
>
> # Get some POSIXct times that span 9 hours over an evening and
> # the following morning.
> x <- as.POSIXct('2011-12-10 16:55:26 EST', tz='EST')+(0:9)*3600
>
> # The first 8 date/times are in the evening of Dec 10, the last 2 in the
> # morning of Dec 11 (in EST).
>
> as.character(x)
 [1] "2011-12-10 16:55:26" "2011-12-10 17:55:26" "2011-12-10 18:55:26"
 [4] "2011-12-10 19:55:26" "2011-12-10 20:55:26" "2011-12-10 21:55:26"
 [7] "2011-12-10 22:55:26" "2011-12-10 23:55:26" "2011-12-11 00:55:26"
 [10] "2011-12-11 01:55:26"
>
> # Not sensible direction conversion POSIXct->Date.
> # Times after 7pm in the POSIXct object turn up as the next day in
> # the Date Object. (They are interpreted as a time-of-day in GMT
> # and 7pm EST is 12 midnight GMT.)
> as.Date(x)
 [1] "2011-12-10" "2011-12-10" "2011-12-10" "2011-12-11" "2011-12-11"
 [6] "2011-12-11" "2011-12-11" "2011-12-11" "2011-12-11" "2011-12-11"
>
> # Another way of looking at the as.Date(POSIXct)
>
> data.frame(as.character(x), as.Date(as.character(x)), as.Date(x), check.names=F)
       as.character(x) as.Date(as.character(x)) as.Date(x)
1  2011-12-10 16:55:26              2011-12-10 2011-12-10
2  2011-12-10 17:55:26              2011-12-10 2011-12-10
```

```
3  2011-12-10 18:55:26              2011-12-10 2011-12-10
4  2011-12-10 19:55:26              2011-12-10 2011-12-11
5  2011-12-10 20:55:26              2011-12-10 2011-12-11
6  2011-12-10 21:55:26              2011-12-10 2011-12-11
7  2011-12-10 22:55:26              2011-12-10 2011-12-11
8  2011-12-10 23:55:26              2011-12-10 2011-12-11
9  2011-12-11 00:55:26              2011-12-11 2011-12-11
10 2011-12-11 01:55:26              2011-12-11 2011-12-11
>
> # as.Date(POSIXlt) works differently from as.Date(POSIXct)
>
> data.frame(as.character(x), as.Date(x), as.Date(as.POSIXlt(x)), check.names=F)
       as.character(x) as.Date(x) as.Date(as.POSIXlt(x))
1  2011-12-10 16:55:26 2011-12-10             2011-12-10
2  2011-12-10 17:55:26 2011-12-10             2011-12-10
3  2011-12-10 18:55:26 2011-12-10             2011-12-10
4  2011-12-10 19:55:26 2011-12-11             2011-12-10
5  2011-12-10 20:55:26 2011-12-11             2011-12-10
6  2011-12-10 21:55:26 2011-12-11             2011-12-10
7  2011-12-10 22:55:26 2011-12-11             2011-12-10
8  2011-12-10 23:55:26 2011-12-11             2011-12-10
9  2011-12-11 00:55:26 2011-12-11             2011-12-11
10 2011-12-11 01:55:26 2011-12-11             2011-12-11
>
> # Sensible conversion POSIXct->character->Date.
> as.Date(as.character(x))
 [1] "2011-12-10" "2011-12-10" "2011-12-10" "2011-12-10" "2011-12-10"
 [6] "2011-12-10" "2011-12-10" "2011-12-10" "2011-12-11" "2011-12-11"
>
> # Can do this correctly multiple ways, direct POSIXct->Date works if
> # we supply tz='EST' to as.Date()
> all.equal(as.Date(as.character(x)), as.Date(x, tz='EST'))
[1] TRUE
> all.equal(as.Date(as.character(x)), as.Date(as.character(x), tz='EST'))
[1] TRUE
>
> (x.Date <- as.Date(as.character(x)))
 [1] "2011-12-10" "2011-12-10" "2011-12-10" "2011-12-10" "2011-12-10"
 [6] "2011-12-10" "2011-12-10" "2011-12-10" "2011-12-11" "2011-12-11"
>
> # Sensible conversion Date->character->POSIXct
> as.POSIXct(as.character(x.Date))
 [1] "2011-12-10 EST" "2011-12-10 EST" "2011-12-10 EST" "2011-12-10 EST"
 [5] "2011-12-10 EST" "2011-12-10 EST" "2011-12-10 EST" "2011-12-10 EST"
 [9] "2011-12-11 EST" "2011-12-11 EST"
>
> # Not sensible direct conversion Date->POSIXct.  (Unless we really want
> # the time in our zone corresponding to when it is midnight in GMT.)
```

```
> as.POSIXct(x.Date)
 [1] "2011-12-09 19:00:00 EST" "2011-12-09 19:00:00 EST"
 [3] "2011-12-09 19:00:00 EST" "2011-12-09 19:00:00 EST"
 [5] "2011-12-09 19:00:00 EST" "2011-12-09 19:00:00 EST"
 [7] "2011-12-09 19:00:00 EST" "2011-12-09 19:00:00 EST"
 [9] "2011-12-10 19:00:00 EST" "2011-12-10 19:00:00 EST"
>
> # Probably sensible direction conversion Date->POSIXlt -- if this stops
> # being sensible, check code in date*.POSIXlt methods
> as.POSIXlt(x.Date)
 [1] "2011-12-10 UTC" "2011-12-10 UTC" "2011-12-10 UTC" "2011-12-10 UTC"
 [5] "2011-12-10 UTC" "2011-12-10 UTC" "2011-12-10 UTC" "2011-12-10 UTC"
 [9] "2011-12-11 UTC" "2011-12-11 UTC"
>
> # 'tz' argument on as.POSIXlt() is ignored for conversion from Date
> attr(as.POSIXlt(x.Date, tz='UTC'), 'tzone')
[1] "UTC"
> attr(as.POSIXlt(x.Date, tz='GMT'), 'tzone')
[1] "UTC"
> attr(as.POSIXlt(x.Date, tz='EST'), 'tzone')
[1] "UTC"
> all.equal(as.POSIXlt(x.Date), as.POSIXlt(x.Date, tz='UTC'))
[1] TRUE
> all.equal(as.POSIXlt(x.Date), as.POSIXlt(x.Date, tz='GMT'))
[1] TRUE
> all.equal(as.POSIXlt(x.Date), as.POSIXlt(x.Date, tz='EST'))
[1] TRUE
>
```

Index