Package 'bizdays'

June 25, 2018

Title Business Days Calculations and Utilities

Description Business days calculations based on a list of holidays and
nonworking weekdays. Quite useful for fixed income and derivatives pricing.

Version 1.0.6

Author Wilson Freitas <wilson.freitas@gmail.com>

Maintainer Wilson Freitas <wilson.freitas@gmail.com>

URL https://github.com/wilsonfreitas/R-bizdays

VignetteBuilder knitr

Suggests RQuantLib, timeDate, knitr, testthat, covr

Imports methods, utils, jsonlite

Collate 'R-bizdays-package.r' 'datasets.R' 'calendar.R'
'adjust.date.R' 'bizdays.options.R' 'bizseq.R' 'is.bizday.R'
'offset.R' 'bizdiff.R' 'bizdays.R' 'create-calendars.R'
'calendar-export.R' 'getdate.R'

Depends R (>= 2.15)

License MIT + file LICENSE

LazyData true

RoxygenNote 6.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2018-06-25 15:25:00 UTC

R topics documented:

---

bizdays-package   *Business Days Calculations and Utilities*

Description

In many countries the standard approach to price derivatives and fixed income instruments involves
the use of business days. In Brazil, for example, the great majority of financial instruments are
priced on business days counting rules. Given that the use of business days is somehow vital to
handle many tasks. That's the reason why bizdays came up, to make these tasks easier. Excel's
NETWORKDAYS is fairly at hand and once you have a list of holidays it is quite easy to put your
data into a spreadsheet and make things happen. bizdays brings that ease to R.

Although R's users have similar feature in packages like RQuantLib and timeDate it doesn't come
for free. Users have to do some stackoverflow in order to get this task accomplished. bizdays is a
tiny package dramatically focused on that simple task: support calculations involving business days
for a given list of holidays.

bizdays was designed to work with all common date types and ISO formatted character strings and
all methods have support for vectorized operations and handle the recycle rule.

Author(s)

Wilson Freitas

---

adjust.date   *Adjusts the given dates to the next/previous business day*

Description

Rolls the given date to the next or previous business day, unless it is a business day.

Usage

```
adjust.next(dates, cal)

following(dates, cal)

adjust.none(dates, cal)

modified.following(dates, cal)

adjust.previous(dates, cal)

preceding(dates, cal)

modified.preceding(dates, cal)
```

Arguments

dates	dates to be adjusted
cal	an instance of Calendar

Details

`adjust.next` and `following` return the next business day if the given date is not a business day. `adjust.previous` and `preceding` are similar, but return the previous business day. `modified.following` rolls the given date to the next business day, unless it happens in the next month, in this case it returns the previous business day. `modified.preceding` is similar to `modified.following`, but rolls the given date to the previous business day.

Value

`Date` objects adjusted accordingly.

Date types accepted

The argument `dates` accepts `Date` objects and any object that returns a valid `Date` object when passed through `as.Date`, which include all `POSIX*` classes and `character` objects with ISO formatted dates.

Examples

```
cal <- create.calendar("Brazil/ANBIMA", holidaysANBIMA, weekdays=c("saturday", "sunday"))
adjust.next("2013-01-01", "Brazil/ANBIMA")
following("2013-01-01", cal)
modified.following("2016-01-31", cal)
adjust.previous("2013-01-01", cal)
preceding("2013-01-01", cal)
modified.preceding("2016-01-01", cal)
```

---

bizdays	*Computes business days between two dates.*

Description

Returns the amount of business days between 2 dates taking into account the provided Calendar (or bizdays.options$get("default.calendar")).

Usage

```
bizdays(from, to, cal)
```

Arguments

from	the initial dates
to	the final dates
cal	the calendar's name

Value

integer objects representing the amount of business days.

Date types accepted

The arguments from and to accept Date objects and any object that returns a valid Date object when passed through as.Date, which include all POSIX* classes and character objects with ISO formatted dates.

Recycle rule

These arguments handle the recycle rule so vectors of dates can be provided and once those vectors differs in length the recycle rule is applied.

Date adjustment

from and to are adjusted when nonworking dates are provided. Since bizdays function returns the amount of business days between 2 dates, it must start and end in business days. The default behavior, that is defined in Calendar's instantiation with adjust.from and adjust.to, reproduces the Excel's NETWORKDAYS. A common and useful setting is adjust.to=adjust.next which moves expiring maturities to the next business day, once it is not.

Examples

```
create.calendar("Brazil/ANBIMA", holidaysANBIMA, weekdays=c("saturday", "sunday"))
bizdays("2013-01-02", "2013-01-31", "Brazil/ANBIMA")

# Once you have a default calendar set, cal does not need to be provided
bizdays.options$set(default.calendar="Brazil/ANBIMA")
bizdays("2013-01-02", "2013-01-31")

dates <- bizseq("2013-01-01", "2013-01-10")
bizdays(dates, "2014-01-31")
```

---

bizdays.options   *bizdays' options*

Description

bizdays.options defines option parameters used internally in bizdays.

Usage

```
bizdays.options
```

Format

A list object with *methods* get and set attached to.

Details

Parameters are stored in bizdays.options using get and set

```
bizdays.options$set(option.key=value)
bizdays.options$get("option.key")
```

bizdays supports the following parameter:

- default.calendar: the default calendar to be used with the functions: bizdays, bizdayse,
  adjust.next, adjust.previous, is.bizday, bizseq, offset.

Examples

```
create.calendar(name="actual")
bizdays.options$set(default.calendar="actual")
bizdays("2013-07-12", "2013-07-22")
```

---

bizdayse   *Business days and current days equivalence*

Description

`bizdayse` stands for business days equivalent, it returns the amount of business days equivalent to a given number of current days.

Usage

```
bizdayse(dates, curd, cal)
```

Arguments

dates	the reference dates
curd	the amount of current days
cal	the calendar's name

Details

Let us suppose I have a reference date `dates` and I offset that date by `curd` current days. `bizdayse` returns the business days between the reference date and the new date offset by `curd` current days.

This is equivalent to

```
refdate <- Sys.Date()
curd <- 10
newdate <- refdate + 10 # offset refdate by 10 days
# this is equals to bizdayse(refdate, 10)
bizdays(refdate, newdate)
```

Value

An `integer` representing an amount of business days.

Date types accepted

The argument `dates` accepts `Date` objects and any object that returns a valid `Date` object when passed through `as.Date`, which include all `POSIX*` classes and `character` objects with ISO formatted dates.

Recycle rule

These arguments handle the recycle rule so a vector of dates and a vector of numbers can be provided and once those vectors differs in length the recycle rule is applied.

Examples

```
create.calendar("Brazil/ANBIMA", holidaysANBIMA, weekdays=c("saturday", "sunday"))
bizdayse("2013-01-02", 3, "Brazil/ANBIMA")
```

---

bizdiff	*Compute the amount of business days between dates*

Description

Returns the number of business days between dates in a given vector of dates.

Usage

```
bizdiff(dates, cal)
```

Arguments

dates	a vector containing the dates to be differenced
cal	the calendar's name

Value

A 'numeric' vector of length 'n-1' (where 'n' is the input vector length), containing the business days computed between pairs of dates.

Date types accepted

The arguments `from` and `to` accept `Date` objects and any object that returns a valid `Date` object when passed through `as.Date`, which include all `POSIX*` classes and `character` objects with ISO formatted dates.

Examples

```
dates <- c("2017-05-10", "2017-05-12", "2017-05-17")
bizdiff(dates, "Brazil/ANBIMA")
```

---

bizseq   *Create a sequence of business days*

Description

Returns a sequence of dates with business days only.

Usage

```
bizseq(from, to, cal)
```

Arguments

from	the initial date
to	the final date (must be greater than `from`)
cal	the calendar's name

Value

A vector of `Date` objects that are business days according to the provided `Calendar`.

Date types accepted

The arguments `from` and `to` accept `Date` objects and any object that returns a valid `Date` object when passed through `as.Date`, which include all `POSIX*` classes and `character` objects with ISO formatted dates.

Examples

```
create.calendar("Brazil/ANBIMA", holidaysANBIMA, weekdays=c("saturday", "sunday"))
bizseq("2013-01-02", "2013-01-31", "Brazil/ANBIMA")
```

---

calendar-holidays-weekdays
                           Calendar's holidays and weekdays

Description

Returns calendar's list of holidays and weekdays

Usage

```
holidays(cal)

## Default S3 method:
holidays(cal)

## S3 method for class 'Calendar'
holidays(cal)

## S3 method for class 'character'
holidays(cal)

## Default S3 method:
weekdays(x, ...)

## S3 method for class 'Calendar'
weekdays(x, ...)

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

Arguments

`cal`	character with calendar name or the calendar object
`x`	character with calendar name or the calendar object
`...`	unused argument (this exists to keep compliance with weekdays generic)

Examples

```
holidays("actual")
weekdays("actual")
# empty calls return the default calendar attributes
holidays()
weekdays()
```

---

calendar-import-export

Import and export calendars

Description

The calendars can be specified in JSON files and these functions helps with importing and exporting calendars to text files.

Usage

```
save_calendar(cal, con)

load_calendar(con)
```

Arguments

cal   the calendar's name

con   a connection object or a character string.

Details

save_calendar exports a calendar to a JSON file and load_calendar imports.

In load_calenadar, the con argument can be a connection object or a character string specifying either the file or the JSON text.

JSON calendar's specification

Here's an example of a calendar's specification.

```
{
 "name": "Brazil/ANBIMA",
 "weekdays": ["saturday", "sunday"],
 "holidays": ["2001-01-01", "2001-02-26", "2001-02-27", "2001-04-13"],
 "adjust.from": "following",
 "adjust.to": "preceding"
 "financial": true,
}
```

Examples

```
con <- tempfile(fileext = ".json")
save_calendar("actual", con)
load_calendar(con)
```

---

calendar-register   *Calendars register*

Description

Every calendar created with create.calendar is stored in the calendar register. The idea behind this register is allowing calendars to be accessed by its names.

Usage

```
calendars()

remove.calendars(cals)

has.calendars(cals)
```

Arguments

cals   character vector of calendars names

Details

`calendars` returns the object which represents the calendars register. Since the register inherits from `environment`, the calendars are retrieved with the `[[` operator. But the register object has its own `print` generic which helps listing all registered calendars.

`remove.calendars` remove calendars from the register.

Examples

```
# ACTUAL calendar
cal <- create.calendar("Actual")
cal <- calendars()[["Actual"]]
remove.calendars("Actual")
# lists registered calendars
calendars()
has.calendars(c("actual", "weekends"))
```

---

create.calendar   *Creates calendars*

Description

`create.calendar` creates calendars and stores them in the calendar register.

Usage

```
create.calendar(name, holidays = integer(0), weekdays = NULL,
  start.date = NULL, end.date = NULL, adjust.from = adjust.none,
  adjust.to = adjust.none, financial = TRUE)
```

Arguments

name   calendar's name. This is used to retrieve calendars from register.

holidays   a vector of Dates which contains the holidays

weekdays	a character vector which defines the weekdays to be used as non-working days (defaults to NULL which represents an actual calendar). It accepts: sunday, monday, thuesday, wednesday, thursday, friday, saturday. Defining the weekend as nonworking days is weekdays=c("saturday", "sunday").
start.date	the date which the calendar starts
end.date	the date which the calendar ends
adjust.from	is a function to be used with the bizdays's from argument. That function adjusts the argument if it is a nonworking day according to calendar.
adjust.to	is a function to be used with the bizdays's to argument. See also adjust.from.
financial	is a logical argument that defaults to TRUE. This argument defines the calendar as a financial or a non financial calendar. Financial calendars don't consider the ending business day when counting working days in bizdays. bizdays calls for non financial calendars are greater than financial calendars calls by one day.

Details

The arguments start.date and end.date can be set but once they aren't and holidays is set, start.date is defined to min(holidays) and end.date to max(holidays). If holidays isn't set start.date is set to '1970-01-01' and end.date to '2071-01-01'.

weekdays is controversial but it is only a sequence of nonworking weekdays. In the great majority of situations it refers to the weekend but it is also possible defining it differently. weekdays accepts a character sequence with lower case weekdays (sunday, monday, thuesday, wednesday, thursday, friday, saturday). This argument defaults to NULL because the default intended behavior for create.calendar returns an *actual* calendar, so calling create.calendar(name="xxx") returns a *actual* calendar named *xxx*. (for more calendars see Day Count Convention) To define the weekend as the nonworking weekdays one could simply use weekdays=c("saturday", "sunday").

The arguments adjust.from and adjust.to are used to adjust bizdays' arguments from and to, respectively. These arguments need to be adjusted when nonworking days are provided. The default behavior, setting adjust.from=adjust.previous and adjust.to=adjust.next, works like Excel's function NETWORKDAYS, since that is fairly used by a great number of practitioners.

Calendars register

Every named calendar is stored in a register so that it can be retrieved by its name (in calendars). bizdays' methods also accept the calendar's name on their cal argument. Given that, naming calendars is strongly recommended.

See Also

calendars, bizdays

Examples

```
# ANBIMA's calendar (from Brazil)
cal <- create.calendar("Brazil/ANBIMA", holidays=holidaysANBIMA, weekdays=c("saturday", "sunday"))

# ACTUAL calendar
cal <- create.calendar("Actual")
```

```
# named calendars can be accessed by its name
create.calendar(name="Actual")
bizdays('2016-01-01', '2016-03-14', 'Actual')
```

---

getdate   *Obtaining dates using other dates (or month or year) as reference*

Description

Imagine you have one date and want the first or last day of this date's month. For example, you have
the date 2018-02-01 and want the last day of its month. You have to check whether or not its year
is a leap year, and this sounds a tough task. getdate helps with returning specific dates according
to a reference than can be another date, a month or an year.

Usage

```
getdate(expr, ref, cal = bizdays.options$get("default.calendar"))
```

Arguments

expr   a character string specifying the date to be returned (see Details)

ref   a ref object (see Details)

cal   the calendar's name

        expr represents the day has to be returned, here it follows a few examples:

- "second day"
- "10th bizday"
- "3rd wed"
- "last bizday"
- "first fri"

        expr is a character string with two terms: "<position> <day>"

- positions: first or 1st, second or 2nd, third or 3rd, last and XXth
  (examples 6th or 11th)
- days: day, bizday, or weekdays (sun, mon, tue, wed, thu, fri, sat)

        getdate returns dates according to a reference that can be a month or an year.
This reference is build with the ref object. The ref object specifies a month or
an year based on a date or the month and year can be directly specified.

See Also

ref

Examples

```
getdate("first day", ref("2018-01-01", ym = "month"), "actual")
getdate("10th wed", ref(2018), "actual")
getdate("last bizday", ref(2010:2018), "Brazil/ANBIMA")
dts <- seq(as.Date("2018-01-01"), as.Date("2018-12-01"), "month")
getdate("first bizday", ref(dts, ym = "month"), "Brazil/ANBIMA")
```

---

holidaysANBIMA   *ANBIMA's holidays list*

Description

A dataset containing the list of holidays delivered by ANBIMA (www.anbima.com.br).

Format

a vector with `Date` objects that represent holidays

---

is.bizday   *Checks if the given dates are business days.*

Description

Returns `TRUE` if the given date is a business day and `FALSE` otherwise.

Usage

```
is.bizday(dates, cal)
```

Arguments

dates	dates to be checked
cal	the calendar's name

Value

`logical` objects informing that given dates are or are not business days.

Date types accepted

The argument dates accepts `Date` objects and any object that returns a valid `Date` object when passed through `as.Date`, which include all `POSIX*` classes and `character` objects with ISO formatted dates.

Examples

```
create.calendar("Brazil/ANBIMA", holidaysANBIMA, weekdays=c("saturday", "sunday"))
is.bizday("2013-01-02", "Brazil/ANBIMA")

# Once you have a default calendar set, cal does not need to be provided
bizdays.options$set(default.calendar="Brazil/ANBIMA")

dates <- seq(as.Date("2013-01-01"), as.Date("2013-01-05"), by="day")
is.bizday(dates)
```

---

offset   *Offsets the given* dates *by* n *business days*

Description

Returns the given dates offset by the given amount of n business days.

Usage

```
offset(dates, n, cal)

add.bizdays(dates, n, cal)
```

Arguments

dates	dates to be offset
n	the amount of business days to offset
cal	the calendar's name

Details

The argument n accepts a sequence of integers and if its length differs from dates' length, the recycle rule is applied to fulfill the gap.

Value

Date objects offset by the amount of days defined.

Date types accepted

The argument dates accepts Date objects and any object that returns a valid Date object when passed through as.Date, which include all POSIX* classes and character objects with ISO formatted dates.

Recycle rule

These arguments handle the recycle rule so a vector of dates and a vector of numbers can be provided
and once those vectors differs in length the recycle rule is applied.

Examples

```
create.calendar("Brazil/ANBIMA", holidaysANBIMA, weekdays=c("saturday", "sunday"),
                adjust.from=adjust.next, adjust.to=adjust.previous)
offset("2013-01-02", 5, "Brazil/ANBIMA")

# Once you have a default calendar set, cal does not need to be provided
bizdays.options$set(default.calendar="Brazil/ANBIMA")

dates <- seq(as.Date("2013-01-01"), as.Date("2013-01-05"), by="day")
is.bizday(dates)
offset(dates, 1)
```

---

other-calendars   *Calendars from other packages*

Description

The packages RQuantLib and timeDate (Rmetrics) have functions to compute business days be-
tween 2 dates according to a predefined calendar. bizdays creates calendars based on these func-
tions.

Usage

```
load_quantlib_calendars(ql_calendars = NULL, from, to)

load_rmetrics_calendars(year)
```

Arguments

ql_calendars	(QuantLib only) A character vector with the names of QuantLib's calendars. This parameter defaults to NULL, which loads all calendars.
from	(QuantLib only) the start date
to	(QuantLib only) the end date
year	(timeDate Rmetrics only) a vector with years to create the calendars.

Details

To load QuantLib's calendars use `load_quantlib_calendars` defining which calendar has to be
loaded by its name and the range of dates the calendar has to handle. All QuantLib calendars have
the `QuantLib` prefix.

To load Rmetrics' calendars use `load_rmetrics_calendars` defining the years the calendar has to
handle. All Rmetrics calendars have the `Rmetrics` prefix.

List of calendars

QuantLib Calendars:

- QuantLib/Argentina
- QuantLib/Australia
- QuantLib/Brazil
- QuantLib/Canada
- QuantLib/Canada/Settlement
- QuantLib/Canada/TSX
- QuantLib/China
- QuantLib/CzechRepublic
- QuantLib/Denmark
- QuantLib/Finland
- QuantLib/Germany
- QuantLib/Germany/FrankfurtStockExchange
- QuantLib/Germany/Settlement
- QuantLib/Germany/Xetra
- QuantLib/Germany/Eurex
- QuantLib/HongKong
- QuantLib/Hungary
- QuantLib/Iceland
- QuantLib/India
- QuantLib/Indonesia
- QuantLib/Italy
- QuantLib/Italy/Settlement
- QuantLib/Italy/Exchange
- QuantLib/Japan
- QuantLib/Mexico
- QuantLib/NewZealand
- QuantLib/Norway
- QuantLib/Poland
- QuantLib/Russia
- QuantLib/SaudiArabia
- QuantLib/Singapore
- QuantLib/Slovakia
- QuantLib/SouthAfrica
- QuantLib/SouthKorea
- QuantLib/SouthKorea/KRX

- QuantLib/Sweden
- QuantLib/Switzerland
- QuantLib/Taiwan
- QuantLib/Turkey
- QuantLib/Ukraine
- QuantLib/UnitedKingdom
- QuantLib/UnitedKingdom/Settlement
- QuantLib/UnitedKingdom/Exchange
- QuantLib/UnitedKingdom/Metals
- QuantLib/UnitedStates
- QuantLib/UnitedStates/Settlement
- QuantLib/UnitedStates/NYSE
- QuantLib/UnitedStates/GovernmentBond
- QuantLib/UnitedStates/NERC

Rmetrics Calendars:

- Calendar Rmetrics/LONDON
- Calendar Rmetrics/NERC
- Calendar Rmetrics/NYSE
- Calendar Rmetrics/TSX
- Calendar Rmetrics/ZURICH

Examples

```
if (require("RQuantLib")) {
 # loading Argentina calendar
 load_quantlib_calendars('Argentina', from='2016-01-01', to='2016-12-31')
 bizdays('2016-01-01', '2016-03-14', 'QuantLib/Argentina')
 # loading 2 calendars
 load_quantlib_calendars(c('UnitedStates/NYSE', 'UnitedKingdom/Settlement'),
                         from='2016-01-01', to='2016-12-31')
 bizdays('2016-01-01', '2016-03-14', 'QuantLib/UnitedStates/NYSE')
 # loading all QuantLib's 49 calendars
 load_quantlib_calendars(from='2016-01-01', to='2016-12-31')
 bizdays('2016-01-01', '2016-03-14', 'QuantLib/Brazil')
}

if (require("timeDate")) {
 # loading all Rmetrics calendar
 load_rmetrics_calendars(2016)
 bizdays('2016-01-01', '2016-03-14', 'Rmetrics/NERC')
 bizdays('2016-01-01', '2016-03-14', 'Rmetrics/NYSE')
}
```

---

ref *Creates date references to be used in* getdate

Description

Date references are specifically months or years to be used in getdate. Months and years can be specified directly or can be base on a given date. getdate returns a date that is in the reference passed.

Usage

```
ref(x, ...)

## S3 method for class 'Date'
ref(x, ym = c("month", "year"), ...)

## S3 method for class 'character'
ref(x, ...)

## S3 method for class 'numeric'
ref(x, ...)
```

Arguments

x	a Date vector, a character vector (specifying dates, months or years) or a numeric vector (specifying years)
...	additional arguments
	If a date (character or Date) is passed to ref it has to specified whether the reference is to the month or the year of the given date. This is set in the argument ym that accepts month (default) or year.
ym	a character string with the values month or year (see Details)

Examples

```
ref(as.Date("2018-01-01"), "month") # refers to 2018-01
ref("2018-01-01", "month")          # refers to 2018-01
ref("2018-01-01", "year")           # refers to 2018

ref(c("2018-01", "2018-02")) # refers to 2018-01 and 2018-02
ref("2018") # refers to 2018
ref(2010:2018) # refers to all years from 2010 to 2018
```

Index