

Package ‘fmdates’

January 5, 2018

Type Package

Title Financial Market Date Calculations

Version 0.1.4

Description Implements common date calculations relevant for specifying the economic nature of financial market contracts that are typically defined by International Swap Dealer Association (ISDA, <<http://www2.isda.org>>) legal documentation. This includes methods to check whether dates are business days in certain locales, functions to adjust and shift dates and time length (or day counter) calculations.

License GPL-2

URL <https://github.com/manuelcostigan/fmdates>,
<https://manuelcostigan.github.io/fmdates/>

BugReports <https://github.com/manuelcostigan/fmdates/issues>

Imports assertthat, lubridate (>= 1.7.0), methods, utils

Suggests covr, knitr, rmarkdown, testthat

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

NeedsCompilation no

Author Imanuel Costigan [aut, cre]

Maintainer Imanuel Costigan <i.costigan@me.com>

Repository CRAN

Date/Publication 2018-01-04 23:07:49 UTC

R topics documented:

adjust	2
Calendar	3

eom	4
fmdates	5
generate_schedule	5
is	6
is_eom	7
is_good	7
is_valid_bdc	8
is_valid_day_basis	9
JointCalendar	10
locale	11
shift	11
tz	12
year_frac	13
Index	15

adjust	<i>Adjust to good dates</i>
--------	-----------------------------

Description

One common financial markets date arithmetic requires a date needs to be rolled to the closest business day following some convention (see [is_valid_bdc\(\)](#) for further details). Such rolled dates can be determined by calling `adjust()`.

Usage

```
adjust(dates, bdc, calendar)
```

Arguments

dates	a vector of dates to adjust.
bdc	the business day convention used to roll the dates if necessary
calendar	an object that inherits from Calendar or JointCalendar which is used to determine the goodness of dates

Value

a vector of adjusted dates - good days are unadjusted

See Also

Other calendar methods: [generate_schedule](#), [is_good](#), [is_valid_bdc](#), [is](#), [locale](#), [shift](#), [tz](#)

Examples

```
ausy <- AUSYCalendar()
adjust(lubridate::ymd("20120102"), "u", ausy)
adjust(lubridate::ymd("20120102"), "f", ausy)
adjust(lubridate::ymd("20120102"), "mf", ausy)
adjust(lubridate::ymd("20120102"), "p", ausy)
adjust(lubridate::ymd("20120102"), "mp", ausy)
adjust(lubridate::ymd("20120102"), "ms", ausy)
```

Calendar

Build a calendar

Description

Calendars are necessary for two reasons: they define whether a calendar day is a good business day in a given locale and they are used to store the time zone for the locale. Calendars can correspond to a single locale (usually a city). These inherit from the `Calendar` class. The package implements a number of calendars for key financial market locales such as `AUSYCalendar`, `USNYCalendar` and `EUTACalendar` (`TARGET`). You can also define a joint locale using `JointCalendar()`.

Usage

`Calendar(locale, tz)`

`EmptyCalendar()`

`AUSYCalendar()`

`AUMECalendar()`

`CHZHCalendar()`

`EUTACalendar()`

`GBLOCalendar()`

`HKHKCalendar()`

`JPTOCalendar()`

`NOOSCalendar()`

`NZAUCalendar()`

`NZWECalendar()`

`USNYCalendar()`

Arguments

locale	a four letter string representing an abbreviation of the locale. The package uses locale representations loosely based on UN/LOCODE (e.g. Australia/Sydney is represented by AUSY rather than AU/SYD per the LOCODE specification). The locale is used as a prefix to the calendar's S3 class in the following manner: <locale>Calendar (e.g. AUSYCalendar).
tz	the time zone associated with the given locale using OlsonNames() (e.g. Australia/Sydney)

Value

Calendar() returns a function that constructs an object inheriting from the Calendar class. The calendar constructors provided by the package returns an object that inherits from Calendar.

See Also

Other calendar classes: [JointCalendar](#)

Examples

```
Calendar(NA, NA) # Defined: EmptyCalendar()
Calendar("AUSY", "Australia/Sydney") # Defined: AUSYCalendar()
```

eom

The end of month date

Description

The dates are rounded to the end of their respective months.

Usage

```
eom(dates)
```

Arguments

dates a vector of dates.

Value

a date vector with the same class as dates

Examples

```
library("lubridate")
eom(ymd(20120203, 20140203))
```

fmdates	<i>fmdates</i>
---------	----------------

Description

Implements common date calculations relevant for specifying the economic nature of financial market contracts that are typically defined by International Swap Dealer Association (ISDA) legal documentation.

Details

The key classes and methods introduced by this package are documented in [Calendar](#), [JointCalendar](#), [is_good\(\)](#), [adjust\(\)](#), [shift\(\)](#) and [year_frac\(\)](#).

generate_schedule	<i>Generate a date schedule</i>
-------------------	---------------------------------

Description

Generate a date schedule from `effective_date` to `termination_date`. This code was derived from the Quantlib method `Schedule::Schedule`. This can be used to generate the cash flow, fixing and projection dates of an interest rate swap according to certain conventions.

Usage

```
generate_schedule(effective_date, termination_date, tenor,
                 calendar = EmptyCalendar(), bdc = "u", stub = "short_front",
                 eom_rule = FALSE, first_date = effective_date,
                 last_date = termination_date)
```

Arguments

<code>effective_date</code>	the date at which the schedule begins. For example, the effective date of a swap. This should be POSIXct .
<code>termination_date</code>	the date at which the schedule ends. For example, the termination date of a swap. This should be POSIXct .
<code>tenor</code>	the frequency of the events for which dates are generated. For example, <code>month(3)</code> reflects events that occur quarterly. Should be an atomic Period-class of length one
<code>calendar</code>	a Calendar
<code>bdc</code>	a string representing one of the following business day conventions: "u", "f", "mf", "p", "mp", "ms" (unadjusted, following, modified following, preceding, modified preceding and modified succeeding, resp.)

stub	a string representing one of the following stub types: "short_front", "short_back", "long_front", "long_back".
eom_rule	a logical value defining whether the end-to-end convention applies.
first_date	date of first payment for example. This defaults to effective_date as is usually the case
last_date	date of last payment for example. This defaults to termination_date as is usually the case

Value

an Interval vector

See Also

Other calendar methods: [adjust](#), [is_good](#), [is_valid_bdc](#), [is](#), [locale](#), [shift](#), [tz](#)

Examples

```
library(lubridate)
effective_date <- ymd('20120103')
termination_date <- ymd('20121203')
tenor <- months(3)
stub <- 'short_front'
bdc <- 'mf'
calendar <- AUSYCalendar()
eom_rule <- FALSE
generate_schedule(effective_date, termination_date, tenor, calendar,
  bdc, stub, eom_rule)
```

 is

Calendar class checkers

Description

Calendar class checkers

Usage

```
is.Calendar(x)
```

```
is.JointCalendar(x)
```

Arguments

x object to be tested

Value

TRUE if x inherits from Calendar or JointCalendar (is.Calendar and is.JointCalendar respectively) and FALSE otherwise.

See Also

Other calendar methods: [adjust](#), [generate_schedule](#), [is_good](#), [is_valid_bdc](#), [locale](#), [shift](#), [tz](#)

is_eom	<i>Checks whether dates are last day of month</i>
--------	---

Description

This checks whether the dates provided are the last day of a month.

Usage

```
is_eom(dates)
```

Arguments

dates a vector of dates.

Value

a logical vector

Examples

```
library("lubridate")
is_eom(ymd(20110228)) # TRUE
is_eom(ymd(20120229)) # TRUE
```

is_good	<i>Good date checker</i>
---------	--------------------------

Description

Checks whether dates are business days (good days) in a given locale represented by a Calendar.

Usage

```
is_good(dates, calendar)
```

Arguments

dates	a vector of dates
calendar	an object inheriting from either Calendar or JointCalendar . Dispatch to methods occurs on this argument.

Details

An `is_good` method must be written for each calendar. The default method returns TRUE for all dates. Methods have been implemented for each of the calendars inheriting from the `Calendar` class - see the method's code for more details. The method implemented for the `JointCalendar` class checks whether the supplied dates are good in each or any of the locales represented by the joint calendar depending on the rule specified by the joint calendar.

Value

a logical vector with TRUE if the date is good and FALSE if the date is bad

See Also

[Calendar](#)

Other calendar methods: [adjust](#), [generate_schedule](#), [is_valid_bdc](#), [is](#), [locale](#), [shift](#), [tz](#)

Examples

```
is_good(lubridate::ymd(20160126, 20160411), AUSYCalendar())
is_good(lubridate::ymd(20160126), USNYCalendar())
```

is_valid_bdc	<i>Business day conventions</i>
--------------	---------------------------------

Description

Checks whether business day conventions are valid.

Usage

```
is_valid_bdc(bdc)
```

Arguments

bdc	a character vector
-----	--------------------

Details

The supported day conventions are:

- u - unadjusted. No adjustments made to a date.
- f - following. The date is adjusted to the following business day.
- mf - modified following. As per following convention. However, if the following business day is in the month following the date, then the date is adjusted to the preceding business day.
- p - preceding. The date is adjusted to the preceding business day.
- mp - modified preceding. As per preceding convention. However, if the preceding business day is in the month prior to the date, then the date is adjusted to the following business day.
- ms - modified succeeding. This convention applies to Australian bank bills. Australian bank bills' maturities defined as either early (prior to the 15th) or late month (after the 15th). If the maturity date calculated straight from a bill's term crosses either the end of the month or the 15th of the month, the bill's maturity is adjusted to the preceding business day.

Value

a flag (TRUE or FALSE) if all the supplied business day conventions are supported.

See Also

Other calendar methods: [adjust](#), [generate_schedule](#), [is_good](#), [is](#), [locale](#), [shift](#), [tz](#)

is_valid_day_basis *Day basis conventions*

Description

Checks whether day basis conventions are valid. Supported day basis conventions are documented in [year_frac\(\)](#)

Usage

```
is_valid_day_basis(day_basis)
```

Arguments

day_basis A character vector of day basis conventions.

Value

will return TRUE for day_basis elements that are any of the following: 30/360, 30/360us, 30e/360, 30e/360isda, 30e+/360, act/360, act/365 and act/actisda. Otherwise will return FALSE

See Also

Other counter methods: [actual_360](#), [actual_365](#), [actual_actual_isda](#), [thirty_360_eu_isda](#), [thirty_360_eu_plus](#), [thirty_360_eu](#), [thirty_360_us](#), [thirty_360](#), [year_frac](#)

Examples

```
is_valid_day_basis(c("act/360", "act/365f"))
```

JointCalendar	<i>Joint calendars</i>
---------------	------------------------

Description

Sometimes the calendar governing a financial contract is defined by multiple single locales. These joint calendars are represented by the JointCalendar class.

Usage

```
JointCalendar(calendars, rule = all)
```

Arguments

calendars	a list of at least one Calendar() objects
rule	either all or any corresponding to a date being good if it is good in all or any of the calendars supplied.

Value

an object of class JointCalendar when using JointCalendar()

See Also

Other calendar classes: [Calendar](#)

Examples

```
JointCalendar(list(AUSYCalendar(), AUMECalendar()), all)
JointCalendar(list(AUSYCalendar(), AUMECalendar()), any)
```

locale	<i>Extract locale from calendars</i>
--------	--------------------------------------

Description

Extract locale from calendars

Usage

```
locale(x)
```

Arguments

x an instance of a [Calendar](#) or [JointCalendar](#) object

Value

a string representing the locale (e.g. "AUSY")

See Also

Other calendar methods: [adjust](#), [generate_schedule](#), [is_good](#), [is_valid_bdc](#), [is](#), [shift](#), [tz](#)

Examples

```
locale(AUSYCalendar())  
locale(c(AUSYCalendar(), AUMECalendar()))
```

shift	<i>Shifting dates to good dates</i>
-------	-------------------------------------

Description

The [adjust\(\)](#) function rolls dates to the closest good dates. This function shifts dates by a given [period](#) and adjusting the resulting dates to a closest good dates following the given business day convention.

Usage

```
shift(dates, period, bdc = "u", calendar = EmptyCalendar(),  
      eom_rule = TRUE)
```

Arguments

dates	a vector of dates to shift and adjust
period	an atomic instance of the period class in the sense that only one of its slots should be non-zero. It must also only be a day, month or year period type.
bdc	the business day convention used to roll the dates if necessary (default: "u" - unadjusted)
calendar	an object that inherits from Calendar or JointCalendar which is used to determine the goodness of dates (default: <code>EmptyCalendar()</code>)
eom_rule	if one of the dates is the last business day of the month, is being shifted by a month or year period and <code>eom_rule</code> is TRUE then the shifted date is also the last business day of the month (default: TRUE)

Value

a vector of shifted dates

See Also

Other calendar methods: [adjust](#), [generate_schedule](#), [is_good](#), [is_valid_bdc](#), [is](#), [locale](#), [tz](#)

Examples

```
library(lubridate)
ausy <- AUSYCalendar()
shift(ymd("20120229"), months(1), "u", ausy, FALSE)
shift(ymd("20120229"), months(1), "u", ausy, TRUE)
```

tz	<i>Extract time zone from calendars</i>
----	---

Description

Extract time zone from calendars

Usage

```
## S3 method for class 'Calendar'
tz(x)

## S3 method for class 'JointCalendar'
tz(x)
```

Arguments

x an instance of a [Calendar](#) or [JointCalendar](#) object

Value

a string representing the time zone (e.g. "Australia/Sydney") or vector of time zones in the case of joint calendars

See Also

Other calendar methods: [adjust](#), [generate_schedule](#), [is_good](#), [is_valid_bdc](#), [is](#), [locale](#), [shift](#)

Examples

```
lubridate::tz(AUSYCalendar())
lubridate::tz(c(AUSYCalendar(), AUMCalendar()))
```

year_frac

The years between two dates for a given day basis convention

Description

This calculates the years between two dates using the given day basis convention.

Usage

```
year_frac(date1, date2, day_basis, maturity_date = NULL)
```

Arguments

date1	A vector of dates. This will be coerced to a Date class.
date2	A vector of dates. This will be coerced to a Date class.
day_basis	The basis on which the year fraction is calculated. See is_valid_day_basis()
maturity_date	a vector of dates representing the maturity date of the instrument. Only used for 30E/360 ISDA day basis.

Details

The order of date1 and date2 is not important. If date1 is less than date2 then the result will be non-negative. Otherwise, the result will be negative. The parameters will be repeated with recycling such that each parameter's length is equal to maximum length of any of the parameters.

Value

a numeric vector representing the number of years between date1 and date2.

References

http://en.wikipedia.org/wiki/Day_count_convention

See Also

Other counter methods: [actual_360](#), [actual_365](#), [actual_actual_isda](#), [is_valid_day_basis](#), [thirty_360_eu_isda](#), [thirty_360_eu_plus](#), [thirty_360_eu](#), [thirty_360_us](#), [thirty_360](#)

Examples

```
require(lubridate)
year_frac(ymd("2010-03-31"), ymd("2012-03-31"), "30/360us") # 2
year_frac(ymd("2010-02-28"), ymd("2012-03-31"), "act/360") # 2.116667
year_frac(ymd("2010-02-28"), ymd("2012-03-31"), "act/365") # 2.087671
year_frac(ymd("2010-02-28"), ymd("2012-03-31"), "act/actisda") # 2.086998
```

Index

actual_360, [10](#), [14](#)
actual_365, [10](#), [14](#)
actual_actual_isda, [10](#), [14](#)
adjust, [2](#), [6–9](#), [11–13](#)
adjust(), [5](#), [11](#)
AUMCalendar (Calendar), [3](#)
AUSYCalendar (Calendar), [3](#)

businessdayconventions (is_valid_bdc), [8](#)

Calendar, [2](#), [3](#), [5](#), [8](#), [10–12](#)
CHZHCalendar (Calendar), [3](#)

Date, [13](#)
daybasisconventions
 (is_valid_day_basis), [9](#)

EmptyCalendar (Calendar), [3](#)
eom, [4](#)
EUTACalendar (Calendar), [3](#)

fmdates, [5](#)
fmdates-package (fmdates), [5](#)

GBLOCalendar (Calendar), [3](#)
generate_schedule, [2](#), [5](#), [7–9](#), [11–13](#)

HKHKCalendar (Calendar), [3](#)

is, [2](#), [6](#), [6](#), [8](#), [9](#), [11–13](#)
is_eom, [7](#)
is_good, [2](#), [6](#), [7](#), [7](#), [9](#), [11–13](#)
is_good(), [5](#)
is_valid_bdc, [2](#), [6–8](#), [8](#), [11–13](#)
is_valid_bdc(), [2](#)
is_valid_day_basis, [9](#), [14](#)
is_valid_day_basis(), [13](#)

JointCalendar, [2](#), [4](#), [5](#), [8](#), [10](#), [11](#), [12](#)
JointCalendar(), [3](#)
JPTOCalendar (Calendar), [3](#)

locale, [2](#), [6–9](#), [11](#), [12](#), [13](#)

NOOSCalendar (Calendar), [3](#)
NZAUCalendar (Calendar), [3](#)
NZWECalendar (Calendar), [3](#)

OlsonNames(), [4](#)

period, [11](#)
period class, [12](#)
POSIXct, [5](#)

shift, [2](#), [6–9](#), [11](#), [11](#), [13](#)
shift(), [5](#)

thirty_360, [10](#), [14](#)
thirty_360_eu, [10](#), [14](#)
thirty_360_eu_isda, [10](#), [14](#)
thirty_360_eu_plus, [10](#), [14](#)
thirty_360_us, [10](#), [14](#)
tz, [2](#), [6–9](#), [11](#), [12](#), [12](#)

USNYCalendar (Calendar), [3](#)

year_frac, [10](#), [13](#)
year_frac(), [5](#), [9](#)