Package: kwb.datetime (via r-universe)

September 2, 2024

Title Functions for date/time objects

Version 0.5.0

Description Functions for date/time objects, e.g. functions to convert timestamps between different time zones. Correctness for some functions still to be verified!

License MIT + file LICENSE

URL https://github.com/kwb-r/kwb.datetime

BugReports https://github.com/kwb-r/kwb.datetime/issues

Imports kwb.utils (>= 0.4.2)

Suggests knitr (>= 1.23), rmarkdown (>= 1.13), testthat (>= 2.0.1)

VignetteBuilder knitr

Remotes github::kwb-r/kwb.utils

Encoding UTF-8

RoxygenNote 7.1.1

Repository https://kwb-r.r-universe.dev

RemoteUrl https://github.com/KWB-R/kwb.datetime

RemoteRef HEAD

RemoteSha 5f2b2c4c258ab776bacc6b968ea86af55fbf40b2

Contents

`berlinNormalTimeToBerlinLocalTime` *berlinNormalTimeToBerlinLocalTime*

Description

berlinNormalTimeToBerlinLocalTime

Usage

```
berlinNormalTimeToBerlinLocalTime(x)
```

Arguments

`x`	character string representing a timestamp measured in Berlin without adjusting time during the summer period, i.e. keeping the normal (= winter) time (= UTC+1)

`berlinNormalTimeToUTC` *berlinNormalTimeToUTC*

Description

berlinNormalTimeToUTC

Usage

```
berlinNormalTimeToUTC(x)
```

Arguments

`x`	character string representing a timestamp measured in Berlin without adjusting time during the summer period, i.e. keeping the normal (= winter) time (= UTC+1)

`currentDateString` *String representing the current Date*

Description

String representing the current Date

Usage

```
currentDateString(format = "%d %B %Y")
```

Arguments

`format`	format string containing percentage-placeholders as defined in `strptime`

Value

character string representing the current date

Examples

```
currentDateString()
currentDateString("%d.%m.%Y")
currentDateString("%Y-%m-%d")
```

`currentQuarter` *Number of current Quarter*

Description

Number of current Quarter

Usage

```
currentQuarter()
```

Value

number of current quarter (1, 2, 3 or 4)

`currentYear` *Current Year (as numeric)*

Description

Current Year (as numeric)

Usage

```
currentYear()
```

Value

Current year as four digit number (numeric)

`date_range_CEST` *When does Summer Time start / end?*

Description

At what days does the summer time start /end in a given year?

Usage

```
date_range_CEST(year)
```

Arguments

`year`	Scalar year number between 1980 and 2100.

Examples

```
# At what days does summer time start and end, respectively, in 2010?
date_range_CEST(2010)

# Check if summer time really starts at 2010-03-28. Timestamps between
# 2:00 (inclusive) and 3:00 (exclusive) do not exist in Central European Time
# Note that in this case R removes the time information!
as.POSIXct("2010-03-28 01:59:59", tz = "Europe/Berlin") # CET
as.POSIXct("2010-03-28 02:00:00", tz = "Europe/Berlin") # Time removed!
as.POSIXct("2010-03-28 02:59:59", tz = "Europe/Berlin") # Time removed!
as.POSIXct("2010-03-28 03:00:00", tz = "Europe/Berlin") # CEST

# Check if summer time really ends at "2010-10-31. Timestamps between
# 2:00 (inclusive) and 3:00 (exclusive) exist twice, once in CEST and a
# second time in CET, so R does not know which one you mean!
as.POSIXct("2010-10-31 01:00:00", tz = "Europe/Berlin") # CEST
as.POSIXct("2010-10-31 02:00:00", tz = "Europe/Berlin") # CEST

# R seems to decide (on my computer!) that times before 02:01:50 belong to
# CEST and at or after that time belong to CET!
as.POSIXct("2010-10-31 02:01:49", tz = "Europe/Berlin") # CEST
as.POSIXct("2010-10-31 02:01:50", tz = "Europe/Berlin") # CET

as.POSIXct("2010-10-31 02:02:00", tz = "Europe/Berlin") # CET
as.POSIXct("2010-10-31 03:00:00", tz = "Europe/Berlin") # CET

# Get the starts and ends of CEST for a sequence of years
date_range_CEST(2017:2020)
```

`daysPerMonth` *Number of Days in the Month of the given Date*

Description

Number of Days in the Month of the given Date

Usage

```
daysPerMonth(date)
```

Arguments

`date`	date object

Value

(integer) number of days in the month of the given date

Examples

```
# You may either pass a Date object...
daysPerMonth(as.Date("2010-01-04"))

# ... or a date string in yyyy-mm-dd format
daysPerMonth("2010-01-04")

# Number of days in February 2010
daysPerMonth("2010-02-01")

# Number of days in February 2012
daysPerMonth("2012-02-15")
```

`getEqualStepRanges` *Sequences of Date Time Objects With Equal Time Step*

Description

Sequences of Date Time Objects With Equal Time Step

Usage

```
getEqualStepRanges(times)
```

Arguments

`times`	vector of `POSIXct` objects

Examples

```
# Generate a sequence of date and time objects
as_berlin_posix <- function(x) as.POSIXct(x, tz = "Europe/Berlin")
times <- seq(
  from = as_berlin_posix("2019-01-01"),
  to = as_berlin_posix("2020-01-01"),
  by = 3600
)

# As expected, exactly one sequence of equal time step is found:
getEqualStepRanges(times)

# Simulate the case that timestamps were read from a text file and converted
# with as.POSIXct()
timestamps <- as.character(times)
new_times <- as.POSIXct(timestamps, tz = "Europe/Berlin")

# Show the sequences of equal time steps again
getEqualStepRanges(new_times)
```

```
# What happened? The timestamp 2019-10-27 02:00 appears twice! Once in CEST
# and once in CET. Use a helper function that assigns CEST and CET as
# required:
good_times <- textToEuropeBerlinPosix(timestamps)

# Check if the original date and time objects could be reproduced
identical(good_times, times)
```

`getTimestampSummary` *Summary about a Sequence of (Text) Timestamps*

Description

Summary about a Sequence of (Text) Timestamps

Usage

```
getTimestampSummary(x, format = NULL, template_index = NULL)
```

Arguments

`x`	vector of character representing timestamps
`format`	format description, such as "%Y-%m-%d %H:%M:%S"
`template_index`	index in `x` from which to select a timestamps that is used as a template when looking for an appropriate timestamp format

Examples

```
x <- kwb.datetime::sequenceOfTimestamps("2019-10-31", "2019-11-01")
getTimestampSummary(x)
```

`getTimestepInSeconds` *Get Time Step in Seconds*

Description

Find the time step applied in a sequence of timestamps. Give a warning if more than one time step was found.

Usage

```
getTimestepInSeconds(timestamps, default = 60)
```

Arguments

`timestamps`	vector of POSIXt objects
`default`	default time step in seconds. Default: 60

Value

smallest, non-zero timestep in seconds found in the sequence of timestamps

---

`hasTimeFormat` *Do Timestamps have the expected Format?*

Description

Checks if timestamps are formatted according to timeformat

Usage

```
hasTimeFormat(timestamps, timeformat = NULL, method = 1L)
```

Arguments

`timestamps`	character strings representing timestamps.
`timeformat`	character string giving a date-time format as used by `strptime`.
`method`	Method used to do the check. 1: Check based on pattern matching 2: Check whether `as.POSIXct` returns valid time objects and if re-formatting these objects are identical to the original timestamps. The simple check should be much faster for many timestamps to be checked.

Examples

```
hasTimeFormat("1.1.2012", "%d.%m.%Y") # TRUE
hasTimeFormat("1.13.2012", "%d.%m.%Y") # FALSE
hasTimeFormat("1/31/2012", "%m/%d/%Y") # TRUE
hasTimeFormat("31/1/2012", "%m/%d/%Y") # FALSE

hasTimeFormat(c("1.1.", "1.13.", "12.12.", "32.1."), "%d.%m.")
#                TRUE     FALSE    TRUE      FALSE
```

hsDateStr — *Timestamp or Date Object to String in format yyyy-mm-dd*

Description

Timestamp or Date Object to String in format yyyy-mm-dd

Usage

```
hsDateStr(tstamp)
```

Arguments

`tstamp`	vector of date or time objects

hsToPosix — *Conversion to POSIXt*

Description

Converts an object representing a date (and if applicable a time) into an object of class POSIXct. Supported input classes are character, Date and POSIXt.

Usage

```
hsToPosix(datetime, keepTZ = is.null(tzone), tzone = NULL, lt = FALSE, ...)
```

Arguments

`datetime`	object of class POSIXt or Date or character representing date (and time) information to be converted to class POSIXct.
`keepTZ`	if `TRUE` and if the given object is already of POSIX-type, the returned POSIXct object will be in the same time zone as the original object. Otherwise POSIX-objects will be returned in the time zone *tzone*.
`tzone`	time zone. Will be set to "UTC" if missing. UTC it the preferred time zone as it seems that only UTC prevents the POSIXt-classes from applying daylight-savings time.
`lt`	if TRUE a POSIXlt object is returned instead of a POSIXct object.
`...`	further arguments to be passed to as.POSIXct/as.POSIXlt, e.g. format, help for as.POSIXct/as.POSIXlt.

Details

If *datetime* is already of class POSIXlt or POSIXct the time zone is preserved unless *keepTZ* is FALSE. If *datetime* is a character string it is expected to be in ISO format: "yyyy-mm-dd [HH:MM:SS]" where the time-part in brackets is optional.

Examples

```
# Start with a string representing a timestamp
datetime <- "2011-01-02 12:34:56"

# By default hsToPosix creates a POSIXct object:
ct <- hsToPosix(datetime)
class(ct) # "POSIXct" "POSIXt"

# You may decide to create a POSIXlt object instead:
lt <- hsToPosix(datetime, lt = TRUE)
class(lt) # "POSIXlt" "POSIXt"

# With a POSIXlt object you can access the different parts of the timestamp
sprintf("%d hours, %d minutes, %d seconds", lt$hour, lt$min, lt$sec)

# These are all available pieces of information
# (isdst = is daylight savings time in effect)
sapply(attr(lt, "names"), function(name) try(lt[[name]]))

# You may use hsToPosix to convert between lt and ct
identical(hsToPosix(ct, lt = TRUE), lt)
identical(hsToPosix(lt, lt = FALSE), ct)

# The following time does not exist in CET/CEST but in UTC
# as it is the time when daylight-savings time switched.
hsToPosix("2011-03-27 02:00:00") # "2011-03-27 02:00:00 UTC"

# Compare with as.POSIXct: between 02:00:00 and 02:59:59 the
# time information gets lost and is only recognized again
# from 03:00:00 on. Similar results with as.POSIXlt.
as.POSIXlt("2011-03-27 01:59:59") # "2011-03-27 01:59:59"
as.POSIXlt("2011-03-27 02:00:00") # "2011-03-27"
as.POSIXlt("2011-03-27 02:59:59") # "2011-03-27"
as.POSIXlt("2011-03-27 03:00:00") # "2011-03-27 03:00:00"

# When loading data from an Access table it will be of class
# POSIXct:
#dat <- hsGetTable(xmdb(), "tbl_Hyd")
#class(dat$Zeitst) # "POSIXct" "POSIXt"

# In order to prevent R from considering daylight savings time
# we should convert to UTC time zone. But then we have to keep
# in mind that the indication "UTC" is not correct as the time
# stamps in fact represent the time zone "UTC+1"!
#head(dat$Zeitst)
# "2011-08-23 00:00:00 CEST" "2011-08-23 00:01:00 CEST" ...

#head(hsToPosix(dat$Zeitst))
# "2011-08-23 00:00:00 UTC" "2011-08-23 00:01:00 UTC" ...
```

hsTsIn *Deprecated use timestampIn instead*

Description

Deprecated use timestampIn instead

Usage

```
hsTsIn(...)
```

Arguments

...	passed to timestampIn

intervalKey *Representative String for Part of Timestamp*

Description

Representative String for Part of Timestamp

Usage

```
intervalKey(tstamps, itype)
```

Arguments

tstamps	(vector of) timestamp(s) of type POSXIXt
itype	one of 'y' (year), 'q' (quarter in year), 'm' (month in year), 'd' (day in month in year), 'w' (week in year), 'D' (weekday in month in year), 'qo' (quarter only), 'mo' (month only), 'do' (day only), 'dm' (day in month), 'wo' (week only), 'Do' (weekday only), 'Dy' (weekday in year)

Examples

```
# Define a sequence of times
times <- as.POSIXct(kwb.datetime::sequenceOfTimestamps(
  "2017-11-04 22:00:00", "2017-11-05 02:00:00", step.s = 3000
))

# Apply all different defined types and print the result
for (type in rownames(kwb.datetime:::.timestamp_type_info())) {
  kwb.utils::printIf(TRUE, intervalKey(times, type), paste("\ntype:", type))
}
```

isoToLocaltime *Text Timestamps to POSIXct*

Description

Convert text timestamps in a format according to ISO 8601 to POSIXct objects

Usage

```
isoToLocaltime(timestamps, dbg = TRUE)
```

Arguments

timestamps	vector of character timestamps of format `yyyy-mm-dd HH:MM:SS+[01|02]`, i.e. ending either in '+0100' (UTC offset in Berlin in winter) or '+0200' (UTC offset in Berlin in summer)
dbg	if `TRUE` debug messages are shown

Examples

```
times <- isoToLocaltime(c(
  "2017-10-29 01:00:00+0200",
  "2017-10-29 01:30:00+0200",
  "2017-10-29 02:00:00+0200",
  "2017-10-29 02:30:00+0200",
  "2017-10-29 02:00:00+0100",
  "2017-10-29 02:30:00+0100",
  "2017-10-29 03:00:00+0100",
  "2017-10-29 03:30:00+0100"
))

class(times)
plot(times, rep(1, length(times)), ylab = "", xlab = "LocalTime")
```

isValidTimestampSequence *Check Sequence of Timestamps for Validity*

Description

Different validation checks for sequence of timestamps

Usage

```
isValidTimestampSequence(
  timestamps,
  checks = c("sorted", "duplicates", "timestep"),
  dbg = FALSE
)
```

Arguments

`timestamps`	vector of timestamps of class POSIXt
`checks`	Vector of character indicating the checks to be performed. Available checks: "sorted", "duplicates", "timestep". Default: perform all tests
`dbg`	shall debug messages be shown?

Examples

```
timestamps <- sequenceOfTimestamps("2017-11-03", "2017-11-04", 3600)
times <- as.POSIXct(timestamps)
isValidTimestampSequence(times)
isValidTimestampSequence(rev(times))
isValidTimestampSequence(timestamps = c(times[1], times))
```

`matchingTimeFormat` *Find Time Format matching a Timestamp*

Description

Find Time Format matching a Timestamp

Usage

```
matchingTimeFormat(
  timestamp,
  timeFormats = getTimeFormats(),
  method = 1L,
  warn = TRUE,
  failureValue = NULL
)
```

Arguments

`timestamp`	character timestamp to be checked against different possible timstamp formats
`timeFormats`	vector of possible time formats with placeholders (year), described for `format.POSIXct`
`method`	passed to `hasTimeFormat`
`warn`	if `TRUE` an R warning is issued if no matching format was found. Otherwise a message is given.
`failureValue`	value returned in case that no matching format was found. Default: `NULL`

Value

first time format in *timeformats* that matches the given *timestamp*. NULL is returned if none of the given *timeformats* matches.

Examples

```
# Convert a character timestamp of which the format can be one of two
# possible formats into a POSIXct-object
possibleFormats <- c("%d.%m.%Y", "%d/%m/%Y")

x <- "14.01.2015"
t1 <- hsToPosix(x, format = matchingTimeFormat(x, possibleFormats))

# In fact this is what stringToPosix does (for only one timestamp)
t2 <- stringToPosix(x, formats = possibleFormats)
stopifnot(identical(t1, t2))

# You get a warning if none of the possible formats matches
matchingTimeFormat("01.14.2015", possibleFormats)
```

---

minTimeStep *Minimum Time Step in Sequence of Timestamps*

Description

Minimum Time Step in Sequence of Timestamps

Usage

```
minTimeStep(tstamps, dbg = FALSE)
```

Arguments

`tstamps`	vector of POSIX-type timestamps or any other vector that can be converted to integer
`dbg`	should debug messages be shown?

Examples

```
tstamps <- seq(as.POSIXct("2017-11-03"), as.POSIXct("2017-11-04"), 3600)
minTimeStep(tstamps)

# No need for timestamps!
minTimeStep(c(10, 20, 30, 40, 45, 50, 60))
minTimeStep(c(10, 20, 30, 40, 45, 50, 60), dbg = TRUE)
```

`reformatTimestamp` *Convert Timstamp String from one Format to another*

Description

Convert Timstamp String from one Format to another

Usage

```
reformatTimestamp(x, old.format = NULL, new.format = NULL, ...)
```

Arguments

`x`	vector of timestamps (character)
`old.format`	format in which timestamps in `x` are given. Default: `"%Y-%m-%d %H:%M:%S"`
`new.format`	format to be applied to timestamps. Default: `"%Y-%m-%d %H:%M:%S"`
`...`	further arguments passed to `hasTimeFormat`, such as `method`

`roundTime` *Timestamp complying with Time Step*

Description

Returns for (each of) the given timestamp(s) the timestamp(s) itself if it represents a multiple of the given time step or the nearest smaller or nearest greater timestamp that represents a multiple of the time step.

Usage

```
roundTime(tstamp, tstep, direction = -1)
```

Arguments

`tstamp`	(vector of) timestamp(s) of class "POSIXlt" or "POSIXct"
`tstep`	time step in seconds of which timestamps in *tstamp* shall represent multiples
`direction`	one of -1, 0, 1. If -1, the nearest timestamp (either smaller or greater) complying with the timestamp is returned. If 0, always the nearest greater timestamp and if 1, always the nearest smaller timestamp is returned.

Value

(Vector of) timestamp(s) corresponding to timestamp(s) given in *tstamp* being “rounded” to the nearest — greater or smaller (direction == -1), always smaller (direction == 1) or always greater (direction == 0) — timestamp representing a multiple of the given time step *tstep*.

Examples

```
# Generate a timestamp to be "rounded"
t0 <- hsToPosix("2011-12-24 18:22:05")

# Round to nearest (default) full minute
roundTime(t0, 60) ## = 2011-12-24 18:22:00 UTC

# Round to nearest full greater minute
roundTime(t0, 60, 0) ## = 2011-12-24 18:23:00 UTC

# Round to nearest multiple of 15 minutes (-1 could be omitted)
roundTime(t0, 15*60, -1) ## 2011-12-24 18:15:00 UTC

# Round to nearest smaller multiple of four hours
roundTime(t0, 4*60*60, 1) ## 2011-12-24 16:00:00 UTC
```

`sequenceOfTimestamps` *Create a Sequence of Timestamps*

Description

Creates timestamps of mode character between first timestamp *from* and *to* with a distance of *step.s* seconds between the timestamps.

Usage

```
sequenceOfTimestamps(from, to, step.s = 60)
```

Arguments

`from`	first timestamp of mode character in ISO-Syntax: yyyy-mm-dd [HH:MM:SS] where the part in brackets is optional.
`to`	last timestamp of mode character in ISO-Syntax: yyyy-mm-dd [HH:MM:SS] where the part in brackets is optional.
`step.s`	time step between the timestamps in seconds.

Value

Vector of character timestamps

Examples

```
# Create timestamps of January 2011 with five minutes step
sequenceOfTimestamps("2011-01-01 19:00:00", "2011-01-02", 300)
```

stringToPosix — *Convert a Time String to a POSIXct Object*

Description

Convert a time string to a POSIXct object. Allow for different possible timestamp formats.

Usage

```
stringToPosix(
  x,
  formats = c("%Y-%m-%d %H:%M:%S", "%Y-%m-%d %H:%M", "%Y-%m-%d"),
  ...
)
```

Arguments

`x`	character vector of length one representing a timestamp
`formats`	vector of allowed time formats (using %-placeholders)
`...`	arguments passed to `hsToPosix`

Examples

```
stringToPosix("2016-05-26")
stringToPosix("2016-05-26 12:00")

# additional arguments passed to hsToPosix
stringToPosix("2016-05-26 12:00:33", tzone = "ETC/Gmt-1")

# lt = TRUE -> create POSIXlt instead of POSIXct
lt1 <- stringToPosix("2016-05-26 17:00", lt = TRUE)
lt2 <- stringToPosix("2016-05-26 17:00", lt = TRUE, tz = "Europe/Berlin")

lt1$hour
lt1$isdst # normal time (is daylight saving time = FALSE)
lt2$isdst # summer time (is daylight saving time = TRUE)
```

test_TimeConversionFunctions — *Test Time Conversion Functions*

Description

Test Time Conversion Functions

Usage

```
test_TimeConversionFunctions(year = 2000, normalToSummer = TRUE)
```

Arguments

`year`	year for which to demonstrate the switch between Central European Time (CET) and Central European Summer Time (CEST)
`normalToSummer`	`TRUE`: CET to CEST, `FALSE`: CEST to CET

`textToEuropeBerlinPosix` *Convert Text Timestamps to POSIXct Objects*

Description

This function tries to overcome some problems that may arise when using `as.POSIXct`. It can handle timestamps that originate from a clock that switches between standard time and summer time as well as those originating from a clock that stays in standard time over the whole year. See `vignette("text_to_posixct", package = "kwb.datetime")` for details. It also tries to find a convenient format description string.

Usage

```
textToEuropeBerlinPosix(x, format = NULL, switches = TRUE, dbg = TRUE, ...)
```

Arguments

`x`	vector of text (i.e. character) timestamps
`format`	format string describing the format of a timestamp, such as " placeholders. If not given or `NULL`, the function tries to guess the format from the first timestamp given in `x`.
`switches`	if `TRUE` (the default), the timestamps are assumed to originate from a clock that switches between standard time and summer time. Otherwise (`switches = FALSE`) timestamps are assumed to originate from a clock that stays in standard time over the whole year.
`dbg`	if `TRUE` debug messages are shown
`...`	further arguments passed to `reformatTimestamp` (only relevant if `switches = TRUE`)

Details

When reading timestamps that observe Daylight Saving, it is required that the timestamps in `x` are ordered by time, which should be the case if they were recorded by a measuring device.

Value

vector of POSIXct objects

Examples

```
# Test the functions with the following "switch" days
kwb.datetime::date_range_CEST(2019)

t1 <- textToEuropeBerlinPosix(c("31.03.2019 01:00", "31.03.2019 03:00"))
t2 <- textToEuropeBerlinPosix(c("31.03.2019 01:00", "31.03.2019 02:00"),
                              switches = FALSE)
identical(t1, t2)

t3 <- textToEuropeBerlinPosix(c("27.10.2019 02:00", "27.10.2019 02:00"))
t4 <- textToEuropeBerlinPosix(c("27.10.2019 01:00", "27.10.2019 02:00"),
                              switches = FALSE)
identical(t3, t4)

kwb.datetime::textToEuropeBerlinPosix(c(
  "2017-10-29 01:30:00", # 1: CEST
  "2017-10-29 02:00:00", # 2: CEST
  "2017-10-29 02:30:00", # 3: CEST
  "2017-10-29 02:00:00", # 4: CET
  "2017-10-29 02:30:00", # 5: CET
  "2017-10-29 03:00:00"  # 6: CET
))
```

`timesAroundClockChange` *Times Around Clock Change in Central Europe*

Description

Sequence of Times Around Clock Change in Central Europe

Usage

```
timesAroundClockChange(
  year = 2000,
  normalToSummer = TRUE,
  step_s = 1800,
  length.out = 5
)
```

Arguments

`year`	year for which to demonstrate the switch between Central European Time (CET) and Central European Summer Time (CEST)
`normalToSummer`	`TRUE`: CET to CEST, `FALSE`: CEST to CET
`step_s`	time step in seconds
`length.out`	number of time objects in returned vector

Value

vector of POSIXct objects with length `length.out`

Examples

```
timesAroundClockChange(2019, normalToSummer = TRUE)
timesAroundClockChange(2019, normalToSummer = FALSE)
timesAroundClockChange(2019, TRUE, step_s = 1, length.out = 3)
timesAroundClockChange(2019, FALSE, step_s = 1, length.out = 3)
```

`timestampIn` *Timestamps within Time Interval?*

Description

Returns vector of booleans indicating whether the timestamps at corresponding positions in a vector of timestamps lie within a time interval, specified by first and/or last timestamp of the interval.

Usage

```
timestampIn(
  tstamps,
  tsFirst = NULL,
  tsLast = NULL,
  firstIncluded = TRUE,
  lastIncluded = FALSE,
  dbg = FALSE
)
```

Arguments

`tstamps`	vector of timestamps, either as character strings in ISO-format (yyyy-mm-dd HH:MM:SS), as Date objects or as POSIXt-objects
`tsFirst`	first timestamp (`firstIncluded` == TRUE) or timestamp directly before first timestamp (`firstIncluded` == FALSE) to be considered
`tsLast`	last timestamp (`lastIncluded` == TRUE) or timestamp directly after last timestamp (`lastIncluded` == FALSE) to be considered

`firstIncluded`	if `TRUE`, `tsFirst` represents the first, otherwise the timestamp direcly before the first timestamp to be considered.
`lastIncluded`	if `TRUE`, `tsLast` represents the last, otherwise the timestamp direcly after the last timestamp to be considered.
`dbg`	should debug messages be shown?

Examples

```
tstamps <- sequenceOfTimestamps("2017-11-03", "2017-11-04")
table(timestampIn(tstamps, "2017-11-03 12:00:00", "2017-11-03 13:00:00"))
```

`to.GMT.plus.1` *Character Timestamps to POSIXct Objects (GMT+1)*

Description

Character Timestamps to POSIXct Objects (GMT+1)

Usage

```
to.GMT.plus.1(timestamp)
```

Arguments

`timestamp`	character timestamp(s) to be converted to POSIXct in timezone "Etc/GMT+1"

Value

vector of POSIXct in timezone “Etc/GMT+1”

`toGmtRelativePosix` *Character Timestamps to POSIXct Objects (GMT*+offset*)*

Description

Convert character timestamps to POSIXct objects in time zont GMT+*offset*

Usage

```
toGmtRelativePosix(timestamp, GMT.offset = 1, format = NULL)
```

Arguments

`timestamp`	vector of timestamps (character)
`GMT.offset`	offset to GMT time. Default: 1 = Berlin Normal Time
`format`	format string describing the format of *timstamp*, see help for `strptime`. Default: "%Y-%m-%d %H:%M:%S"

toUTC *Convert POSIXt Object to UTC Time Zone*

Description

Convert POSIXt Object to UTC Time Zone

Usage

```
toUTC(x)
```

Arguments

`x`	object of class POSIXt (either POSIXct or POSIXlt)

Value

POSIXt object in UTC timezone (hopefully!)

Examples

```
# Create a timestamp in the time zone that is set on the local machine
time <- as.POSIXct("2017-11-01 01:16")

# Convert time zone to UTC
time_utc <- toUTC(time)

# The new time zone "UTC" is set in the attribute "tz"
attr(time_utc, "tz")

# The times mean the same, just expressed in another time zone!
time_utc == time
```

utcOffset *Get UTC Offset from Local and UTC Timestamp (Character)*

Description

Get UTC Offset from Local and UTC Timestamp (Character)

Usage

```
utcOffset(LocalDateTime, DateTimeUTC)
```

Arguments

`LocalDateTime`	character string representing a local timestamp
`DateTimeUTC`	character string representing a time stamp in UTC

`utcOffsetBerlinTime` *UTC Offsets of Berlin Local Timestamps*

Description

For local timestamps (character) in the format `"yyyy-mm-dd HH:MM:SS"`, of which is known that they are recorded in time zone Europe/Berlin, i.e. CET in winter and CEST in summer, the UTC offset (i.e. `"+1"` in winter and `"+2"` in summer) is determined. Therefore, it is required that the `timestamps` are ordered by time, which should be the case if they were recorded by a measuring device. Use this function to create unique timestamps by adding their UTC offset.

Usage

```
utcOffsetBerlinTime(timestamps)
```

Arguments

`timestamps`	vector of character representing timestamps in format `"yyyy-mm-dd HH:MM:SS"`

Value

vector of elements `"+0100"` or `"+0200"`, depending on whether the timestamps at corresponding positions in `timestamps` are in CET or CEST, respectively.

Examples

```
# Change from CET to CEST
utcOffsetBerlinTime(c(
  "2017-03-26 01:58:00",
  "2017-03-26 01:59:00",
  "2017-03-26 03:00:00", # jump from 02:00 to 03:00
  "2017-03-26 03:01:00",
  "2017-03-26 03:02:00"
))

#> "+0200" "+0200" "+0100" "+0100" "+0100"

# Note that the following timestamps do not exist in Europe/Berlin timezone
# and would result in an error
## Not run:
utcOffsetBerlinTime(c(
  "2017-03-26 02:00:00",
  "2017-03-26 02:15:00",
  "2017-03-26 02:30:00",
  "2017-03-26 02:45:00"
))
## End(Not run)

#> "+0200" "+0200" "+0200" "+0200"

# Change from CEST to CET
utcOffsetBerlinTime(c(
  "2017-10-29 01:30:00", # CEST
  "2017-10-29 02:00:00", # first time: CEST
  "2017-10-29 02:30:00", # first time: CEST
  "2017-10-29 02:00:00", # second time: CET
  "2017-10-29 02:30:00"  # second time: CET
))

 #> "+0200" "+0200" "+0200" "+0100" "+0100"
```

Index