Package: kwb.base (via r-universe)

August 27, 2024

Title Functions supporting data analysis with R at KWB

Version 0.5.0

Description This package originally contained all the different helper functions developed and used at KWB. As it grew, functions were extracted into new packages that are dedicated to certain topics. The database access functions, for example, that were originally contained here, are now in the package kwb.db. Functions that help to create event information from time series data have been moved to kwb.event and date/time related functions are now in kwb.datetime. This package still contains some functions that are used in different scripts of different KWB projects. Currently, when loading this package, the packages kwb.datetime, kwb.db, kwb.event, kwb.plot and kwb.utils are automatically loaded. I plan to change this behaviour with the next release so that you need to load the other packages explicitly, i.e. with library(kwb.datetime), library(kwb.db), etc.

License MIT + file LICENSE

URL https://github.com/KWB-R/kwb.base/

BugReports https://github.com/KWB-R/kwb.base/issues

Imports kwb.datetime, kwb.db, kwb.event, kwb.plot, kwb.utils, xts

Suggests testthat

Remotes github::kwb-r/kwb.datetime, github::kwb-r/kwb.db, github::kwb-r/kwb.event, github::kwb-r/kwb.plot, github::kwb-r/kwb.utils

Encoding UTF-8

RoxygenNote 7.1.2

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

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

RemoteRef HEAD

RemoteSha 0a8f37eec4ebee01e591fdade5ae60e97108270b

Contents

`artificialHydrograph` *Artificial Hydrograph*

Description

Generates an artificial hydrograph

Usage

```
artificialHydrograph(step.s = 3600, from = "2015-01-01", to = "2015-01-10")
```

Arguments

step.s	time step in seconds
from	first day as character string in format "yyyy-mm-dd"
to	last day as character string in format "yyyy-mm-dd"

Value

data frame with columns *DateTime* and *values*

---

checkForOverlappingTimeSequences *Check for Overlapping Time Sequences*

Description

Check for Overlapping Time Sequences

Usage

```
checkForOverlappingTimeSequences(
  dataFrame,
  main = "Overlapping time sequences in hydraulic data"
)
```

Arguments

dataFrame	frame with timestamps in first column
main	plot title, passed to plotSequenceEvents

Value

TRUE if there are overlapping sequences, else FALSE

---

dataFrameToXts *Convert Time Series Data Frame to xts Object*

Description

Convert Time Series Data Frame to xts Object

Usage

```
dataFrameToXts(
  dataFrame,
  timeColumn = names(dataFrame)[kwb.utils::posixColumnAtPosition(dataFrame)[1]]
)
```

Arguments

dataFrame	data frame containing a timestamp column
timeColumn	name or number of time column. Default: first POSIXt-column in *dataFrame*

Value

xts object with timestamps taken from timestamp column. Non-numeric columns are removed.

---

demoGroupByInterval *Demonstrate hsGroupByInterval*

Description

Creates a plot demonstrating the effect of offset1 and offset2 in function hsGroupByInterval

Usage

```
demoGroupByInterval(
  df = hsExampleTSeries(60),
  step = kwb.datetime::minTimeStep(df[, 1]),
  to_pdf = TRUE
)
```

Arguments

df	data frame containing data to be used for the demonstration
step	time step in seconds
to_pdf	if TRUE the output goes into a PDF file

---

demo_hsFillUp *Create Plot Demonstrating hsFillUp()*

Description

Create Plot Demonstrating hsFillUp()

Usage

```
demo_hsFillUp()
```

Value

demo plot hsFillUp

`fillup` *Fill-up with Missing Timestamps*

Description

Based on the time series given in "tseries" a new time series is generated in which timestamps that are lacking in tseries are included. Optionally missing values are generated by interpolation between existing values.

Usage

```
fillup(
  tseries,
  tsField,
  step_s,
  forceStep,
  interpol,
  includeOrig,
  default = NA,
  dbg = FALSE
)
```

Arguments

`tseries`	data frame representing a time-series of values
`tsField`	Name of timestamp column in tseries. Default: name of first available POSIXt-column
`step_s`	Time step in seconds that shall lie between consecutive timestamps. Non-existing timestamps are generated.
`forceStep`	If TRUE, only timestamps that are multiples of the time step given in step_s are put into the result data frame
`interpol`	if TRUE, the returned data frame will for each value column contain a corresponding column containing interpolated values
`includeOrig`	if TRUE and interpol = TRUE the original columns will remain in the output in addition to the columns containing all (original plus interpolated) values.
`default`	value to be used if there are not enough (at least to) non-NA values to be used for interpolation. Default: NA
`dbg`	If TRUE, debug messages are shown

Value

Returns a data.frame

`firstTimestamp` *First Timestamp in Data Frame*

Description

First Timestamp in Data Frame

Usage

```
firstTimestamp(x)
```

Arguments

`x`	data frame containing a date/time column

`getOverlappingTimeSequences` *Get Overlapping Time Sequences*

Description

Get Overlapping Time Sequences

Usage

```
getOverlappingTimeSequences(timestamps)
```

Arguments

`timestamps`	vector of date and time objects, inheriting from `POSIXt`

Value

data frame with columns *sequenceNo*, *tBeg*, *tEnd*. In the attribute "sequenceNumber" the vector of sequenceNumbers, each of which corresponds to one timestamp is returned.

getSequenceNumber *Get Sequence Number*

Description

Get Sequence Number

Usage

```
getSequenceNumber(timestamps)
```

Arguments

timestamps	vector of timestamps

Value

vector as long as *timestamps* containing numbers 1, 2, 3, ... indicating the belonging to a sequence of increasing timestamps. If the vector contains only ones, this means that *timestamps* is increasingly sorted.

hsDaylightSaving *Find Days of Daylight Saving*

Description

Find Days of Daylight Saving

Usage

```
hsDaylightSaving(year.first = NULL, year.last = NULL)
```

Arguments

year.first	first year of which daylight saving dates are to be calculated
year.last	last year of which daylight saving dates are to be calculated

Value

data frame with columns begST (begin of summer time) and endST (end of summer time)

hsDbTablePlotXY *X-Y Plot of Two Database Table Fields*

Description

Plots the values of two database fields (= columns) against each other and prints the plot into a PDF file.

Usage

```
hsDbTablePlotXY(strDb, strTable, strX, strY, strPdfFile = NULL)
```

Arguments

strDb	full path to MS Access database file (*.mdb)
strTable	name of table in database
strX	name of table field containing the "x" values
strY	name of table field containing the "y" values
strPdfFile	full path to PDF file to be generated. If omitted, the plot is shown on the screen.

Value

If the output device is a pdf file the result of the dev.off() command is returned.

Examples

```
## Not run:
## Plot CSB vs. timestamp values from table "tbl_STA_CAL" in the
## example database into a window on the screen
hsDbTablePlotXY(kwb.db::xmdb(), "tbl_Qua", "myDateTime", "CSB")

## Set paths to a test PDF file
pdf_file <- file.path(tempdir(), "ex_hsDbTablePlotXY.pdf")

## Plot CSBf vs. timestamp values from the same table into a PDF file.
hsDbTablePlotXY(kwb.db::xmdb(), "tbl_Qua", "myDateTime", "CSBf", pdf_file)

## Open PDF file in PDF viewer
kwb.utils::hsShowPdf(pdf_file)

## End(Not run)
```

hsDelNaRowsOrCols *Delete Rows or Columns Containing only NA*

Description

Delete Rows or Columns Containing only NA

Usage

```
hsDelNaRowsOrCols(df, rows = TRUE, drop = FALSE)
```

Arguments

df	data frame
rows	if TRUE, rows that only contain NAs are deleted, else columns.
drop	if TRUE and only one row/column remains this row/column is returned in forms of a vector instead as a data frame.

hsExampleTSeries *Data Frame with Example Time Series*

Description

Example time series

Usage

```
hsExampleTSeries(step)
```

Arguments

step	time step in seconds

Value

data frame with columns *t* (timestamp) and *y* (sinus values)

hsFillUp *Fill-up with Missing Timestamps*

Description

Based on the time series given in "tseries" a new time series is generated in which timestamps that are lacking in tseries are included. Optionally missing values are generated by interpolation between existing values.

Usage

```
hsFillUp(
  tseries,
  tsField = names(tseries)[kwb.utils::posixColumnAtPosition(tseries)[1]],
  step_s = 60,
  forceStep = TRUE,
  limits = NULL,
  interpol = TRUE,
  includeOrig = TRUE,
  default = NA,
  dbg = FALSE
)
```

Arguments

tseries	data frame representing a time-series of values
tsField	Name of timestamp column in tseries. Default: name of first available POSIXt-column
step_s	Time step in seconds that shall lie between consecutive timestamps. Non-existing timestamps are generated.
forceStep	If TRUE, only timestamps that are multiples of the time step given in step_s are put into the result data frame
limits	data frame or matrix with exactly two columns containing first and last timestamp of intervals for which timestamps are to be generated.
interpol	if TRUE, the returned data frame will for each value column contain a corresponding column containing interpolated values
includeOrig	if TRUE and interpol = TRUE the original columns will remain in the output in addition to the columns containing all (original plus interpolated) values.
default	value to be used if there are not enough (at least to) non-NA values to be used for interpolation. Default: NA
dbg	If TRUE, debug messages are shown

Value

Returns a data.frame

`hsFilterPeriod` *Filter Rows Within Time Period*

Description

Filters a data.frame containing time series data for the time interval between *minDate* and *maxDate*.

Usage

```
hsFilterPeriod(
  tSeries,
  minDate,
  maxDate,
  tsField,
  maxIncluded = FALSE,
  dbg = FALSE
)
```

Arguments

`tSeries`	data.frame containing time-series data.
`minDate`	lower boundary of time interval to be selected, as either of POSIXt-object, Date object or string in ISO format: yyyy-mm-dd
`maxDate`	upper boundary of time interval to be selected, as either of POSIXt-object, Date object or string in ISO format: yyyy-mm-dd
`tsField`	name of time stamp column in *tseries*.
`maxIncluded`	if TRUE, *maxDate* is included in the time interval to be filtered for, otherwise it is not included (see Details).
`dbg`	if `TRUE` debug messages are shown

Details

minDate and *maxDate* must be given as character string in ISO format: "yyyy-mm-dd [HH:MM:SS]", where the brackets indicate that time information is optional. The lower boundary is always included in the time interval to be selected, whereas the upper boundary is only included if *maxIncluded* is TRUE and otherwise excluded. With *maxIncluded* = FALSE it is easy to select whole months or whole years by setting *maxDate* to the first day of the next month/year.

Value

Returns *tSeries*, reduced to rows representing a time within the selected time interval between *minDate* and *maxDate*

hsFilterRowsWithValuesInColumns *Filter for Rows with Given Values in Given Columns*

Description

Filter for rows that have specified values in specified columns

Usage

```
hsFilterRowsWithValuesInColumns(dframe, columnValuePairs)
```

Arguments

dframe	data frame
columnValuePairs	list of elements each of which defines a filter criterion in the form <column-name> = <value>

Value

data frame containing those rows of *dframe* that comply with all of the filter criteria defined in *columnValuePairs*

hsGroupBy2Fields *Two-field Grouping of a Data Frame*

Description

Groups data by values in two columns creating a matrix with as many rows as there are distinct values in field1 and as many columns as there are distinct values in field2. The matrix contains the sum of values in the specified value field that belong to the corresponding value combination of field1/field2

Usage

```
hsGroupBy2Fields(
  frmData,
  strValField,
  strField1,
  strField2,
  boolDesc1 = FALSE,
  boolDesc2 = FALSE
)
```

Arguments

frmData	data.frame containing data to be grouped.
strValField	name of value field (= column in frmData)
strField1	name of first field to be grouped by
strField2	name of second field to be grouped by
boolDesc1	if TRUE, columns in result matrix will be ordered according to decreasing values of field1
boolDesc2	if TRUE, rows in result matrix will be ordered according to decreasing values of field2#'

Value

matrix with as many rows as there are distinct values in field1 and as many columns as there are distinct values in field2 of the input data.frame. The matrix contains the sum of values in the specified value field of the input data.frame that belong to the corresponding combination of values in field1/field2

---

hsGroupByInterval *Group Data in Time Intervals*

Description

Builds groups of rows belonging to the same time interval and aggregates the values within the group by using a given function (e.g. sum, mean, min, max)

Usage

```
hsGroupByInterval(
  data,
  interval,
  FUN,
  tsField = names(data)[1],
  offset1 = 0,
  offset2 = interval/2,
  limits = FALSE,
  ...,
  dbg = FALSE
)
```

Arguments

data	data frame containing a timestamp field and data fields to be aggregated over time.
interval	length of time interval in seconds

FUN	function used to aggregate the values within one and the same interval, e.g. sum, mean, min, max
tsField	name of timestamp column, default: name of first column
offset1	number of seconds by which all timestamps are shifted before they are grouped into intervals. The grouping to intervals is done by dividing the timestamps (converted to number of seconds since 1970-01-01) by the interval length and taking the integer part of the division as interval number. Thus, with offset1 = 0 and an interval length of e.g. 60 seconds, the first interval is from 00:00:00 to 00:00:59, the second from 00:01:00 to 00:01:59 etc., whereas offset1 = 30 in this case would lead to intervals 00:00:30 to 00:01:29, 00:01:30 to 00:02:29 etc..
offset2	value given in seconds determining which of the timestamps in an interval represents the interval in the output. If 0, each time interval is represented by the smallest timestamp belonging to the interval. By default, offset2 is half the interval length, meaning that each time interval is represented by the timestamp in the middle of the interval.
limits	if TRUE, two additional columns will be added showing the minimum and maximum value of the interval
...	further arguments passed to aggregate, the internally called function
dbg	if TRUE, debug messages are shown

Examples

```
## Get an example time-series with values every one minute
step <- 60
df <- hsExampleTSeries(step)

## Calculate 5-min-means with
## offset1 = 0 (default), offset2 = interval/2 (default)
df.agg1 <- hsGroupByInterval(df, interval = 5*step, mean, limits = TRUE)
df.agg1

## Shift the interval limits with
## offset1 = 2.5*60, offset2 = interval/2 (default)
df.agg2 <- hsGroupByInterval(df, interval = 5*step, mean, limits = TRUE,
                             offset1 = 2.5*step)
df.agg2

## Shift the timestamps representing the intervals with
## offset1 = 0, offset2 = 0
df.agg3 <- hsGroupByInterval(df, interval = 5*step, mean, limits = TRUE,
                             offset1 = 0, offset2 = 0)
df.agg3

## Show a plot demonstrating the effect of offset1 and offset2
## Not run:
demoGroupByInterval(df)

## End(Not run)
## Handling NA values...
```

```
## Set y to NA at 2 random positions
df[sample(nrow(df), 2), 2] <- NA
df ## Let' have a look at df

## Count NA values per group
hsGroupByInterval(df, interval = 300, function(x){sum(is.na(x))})

## default behaviour: mean(values containing at least one NA) = NA
hsGroupByInterval(df, interval = 300, mean)

## ignore NA values by passing na.rm = TRUE to the aggregate function
hsGroupByInterval(df, interval = 300, mean, na.rm = TRUE)
```

`hsLabValToVal` *Laboratory Value to Numeric Value*

Description

conversion of text representing a number, but possibly starting with "<" or ">" to indicate that the number exceeds the detection limit of an analysis method, to a valid number

Usage

```
hsLabValToVal(
  x,
  country,
  detLimFactorBelow = 0.5,
  detLimFactorAbove = 2,
  factors = c(`<` = detLimFactorBelow, `<<` = detLimFactorBelow, `>` =
    detLimFactorAbove, `>>` = detLimFactorAbove),
  stopOnError = TRUE
)
```

Arguments

`x`	(vector of) character(s) representing values, possibly starting with "<" or ">" to indicate detection limit exceedance
`country`	"en" if value is given in English format (decimal point ".", thousands separator ",") or "de" if value is given in German format (decimal point ",", thousands separator ".").
`detLimFactorBelow`	factor by which detection limit is multiplied in order to get a valid value when the value was below the detection limit. Default value: 0.5
`detLimFactorAbove`	factor by which detection limit is multiplied in order to get a valid value when the value was above the detection limit. Default value: 2

factors	named vector of conversion factors to be multiplied with the numeric values if the name of the factor matches the prefix (e.g. "<", "«", ">", "»") found in front of the value. Set to NULL if not factors are to be applied
stopOnError	if TRUE, the program stops on conversion errors, otherwise shows a warning

Value

data frame with columns *outOfLimit* being one of "" (value within detection limits), "<" (value below detection limit) or ">" (value above detection limit) and *numericValue* containing the value which, in case of detection limit exceedance, may be a substitute value. If there were conversion errors, the column *numericValue* containing the indices of the wrongly formatted values in its attribute "errorIndices".

---

hsLabValToVal_old *Laboratory Value to Numeric Value (Old Version)*

Description

Conversion of text representing a number, but possibly starting with "<" or ">" to indicate that the number exceeds the detection limit of an analysis method, to a valid number

Usage

```
hsLabValToVal_old(
  x,
  country,
  detLimFactorBelow = 0.5,
  detLimFactorAbove = 2,
  stopOnError = TRUE
)
```

Arguments

x	(vector of) character(s) representing values, possibly starting with "<" or ">" to indicate detection limit exceedance
country	"en" if value is given in English format (decimal point ".", thousands separator ",") or "de" if value is given in German format (decimal point ",", thousands separator ".").
detLimFactorBelow	factor by which detection limit is multiplied in order to get a valid value when the value was below the detection limit. Default value: 0.5
detLimFactorAbove	factor by which detection limit is multiplied in order to get a valid value when the value was above the detection limit. Default value: 2
stopOnError	if TRUE, the program stops on conversion errors, otherwise shows a warning

Value

data frame with columns *outOfLimit* being one of "" (value within detection limits), "<" (value below detection limit) or ">" (value above detection limit) and *numericValue* containing the value which, in case of detection limit exceedance, may be a substitute value. If there were conversion errors, the column *numericValue* containing the indices of the wrongly formatted values in its attribute "errorIndices".

---

hsMfRows *Needed Rows for mfrow*

Description

Number of rows needed to plot <nPlots> in a grid with <plotCols> plots per row.

Usage

```
hsMfRows(nPlots, nPlotsPerRow)
```

Arguments

nPlots	number of total plots
nPlotsPerRow	number of plots per row

Value

Number of rows needed to place all the plots.

---

hsMkTimestamps *Sequence of Timestamps*

Description

Creates timestamps between first timestamp *from* and *to* with a distance of *step.s* seconds between the timestamps. If *mdb* is given, the timestamps are written to a database table in which the timestamp field is the primary key.

Usage

```
hsMkTimestamps(from, to, step.s = 60, mdb, tbl, dbg = FALSE)
```

Arguments

from	first timestamp in ISO-Syntax: yyyy-mm-dd [HH:MM:SS] where the part in brackets is optional.
to	last timestamp 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.
mdb	Optional. Full path to MS Access database file (*.mdb).
tbl	Optional. Name of table to be created in *mdb*. If no name is given a name of the type tbl*from*_to_*to_step.s*_s is created. If a table of given name exists, a non-existing name is generated first, so existing tables will not be overwritten.
dbg	if TRUE, debug messages are shown.

Value

Returns vecotor of timestamps if *mdb* is missing or nothing if timestamp table has been generated in database given in *mdb*.

Examples

```
## Not run:
## Write timestamps of January 2011 with five minutes step into example db.
hsMkTimestamps("2011-01-01", "2011-02-01", 300, kwb.db::xmdb())

## End(Not run)

## Output:
# Timestamps have been written to table
#   'tblTimestamps_2011_01_01_to_2011_02_01_300s' in
# 'C:/Users/hsonne/Documents/R/win-library/2.14/
#   kwb.base/extdata/RExKwbBase.mdb'.
# Timestamp field has been set as primary key.
```

---

hsNiceLabels *Nice Labels*

Description

Generates a nice vector of labels by suppressing labels at certain positions. Please use kwb.plot::niceLabels instead

Usage

```
hsNiceLabels(label, labelstep = NULL, labelpos = NULL, mindist = 1, offset = 0)
```

Arguments

label	see niceLabels
labelstep	see niceLabels
labelpos	see niceLabels
mindist	see niceLabels
offset	see niceLabels

---

hsPdfDev *ID(s) of PDF Device(s)*

Description

Returns the IDs of the first or all opened pdf device(s).

Usage

```
hsPdfDev(all = FALSE)
```

Arguments

all	if TRUE, the ids of all pdf devices are returned, if FALSE, only the id of the first pdf device

Value

ID of first (*all* == FALSE) or IDs of all opened pdf devices, as e.g. returned by dev.list

---

hsPlot *Plot to Specific Device*

Description

Generic function for redirecting a plot-command to a specific device.

Usage

```
hsPlot(dev, plotFun = graphics::plot, args)
```

Arguments

dev	device id
plotFun	plot-function to be used, e.g. plot, barplot, points, lines, ...
args	list of arguments to be passed to the function given in *plotFun*

Value

This function returns whtat the plot function given in plotFun returns

hsST2WT *Convert Summer Time to Winter Time*

Description

Conversion of time series in summer time to time series in winter time

Usage

```
hsST2WT(tstamps, dbg = FALSE)
```

Arguments

tstamps	timestamps in summer time
dbg	if TRUE, debug messages are shown

Value

time series, shifted to winter time (timezone is set to "UTC")

hsWait *Wait for Specified Time*

Description

Waits for the specified number of seconds.

Usage

```
hsWait(secs = 1)
```

Arguments

secs	number of seconds to wait

intervalWidthToSeconds *Interval Width to Seconds*

Description

Interval Width to Seconds

Usage

```
intervalWidthToSeconds(intervalWidth)
```

Arguments

intervalWidth	character string starting with numeric characters 0-9 and ending with either of "d" (days), "h" (hours) or "m" (minutes)

lastTimestamp *Last Timestamp in Data Frame*

Description

Last Timestamp in Data Frame

Usage

```
lastTimestamp(x)
```

Arguments

x	data frame containing a date/time column

plotSequenceEvents *Plot Sequence Events*

Description

Plot Sequence Events

Usage

```
plotSequenceEvents(
  timestamps,
  sequences,
  main = "Overlapping time sequences in hydraulic data",
  language = "de"
)
```

Arguments

timestamps	vector of timestamps
sequences	data frame as returned by getOverlappingTimeSequences with attribute "sequenceNumber"
main	plot title
language	"de" (German) or something else (English)

selectTimeInterval *Select Time Interval*

Description

Select Time Interval

Usage

```
selectTimeInterval(
  x,
  t1 = NULL,
  t2 = NULL,
  width = "-7d",
  posixColumn = kwb.utils::posixColumnAtPosition(x),
  dbg = TRUE
)
```

Arguments

x	data frame with at least one time column
t1	first timestamp as text, in yyyy-mm-dd format
t2	optional. last timestamp as text, in yyyy-mm-dd format
width	interval width as text containing number and unit (one of "d" - day, "h" - hour, "m" - minute). E.g. "7d" = 7 days, "20m" = 20 minutes. The number may be negative indicating "last <number> units".
posixColumn	name or number of column in x containing the relevant timestamps
dbg	if TRUE, debug messages are shown

---

selectTimeIntervalDays

Select Time Interval Days

Description

Select Time Interval Days

Usage

```
selectTimeIntervalDays(
  dat,
  days = 7,
  firstDay = as.character(as.Date(lastDay) - days),
  lastDay = substr(utils::tail(dat$DateTime, 1), 1, 10),
  dbg = TRUE
)
```

Arguments

dat	data frame with column "DateTime"
days	number of "last" days to select
firstDay	first day as text, in "yyyy-mm-dd" format
lastDay	last day as text, in "yyyy-mm-dd" format
dbg	if TRUE, debug messages are shown

Index