Package: kwb.event (via r-universe)

August 26, 2024

Title Generate Events from Time Series and Work with Events

Version 0.4.0

Description Functions to generate events from time series and work with events.

License MIT + file LICENSE

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

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

Imports kwb.datetime, kwb.plot, kwb.utils

Suggests covr, testthat

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

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

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

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

RemoteRef HEAD

RemoteSha 7fe54ea62fa3b88992098b59af4c562104c9c427

Contents

`analyseEventRelations` *Analyse Event Relations*

Description

Analyse Event Relations

Usage

```
analyseEventRelations(eventRelations)
```

Arguments

eventRelations	data frame as returned by getEventRelations

---

eventDuration *Event Duration*

Description

Event Duration

Usage

```
eventDuration(tBeg, tEnd, signalWidth)
```

Arguments

tBeg	timestamps representing the event begins
tEnd	timestamps representing the event ends
signalWidth	see description in hsEvents

---

eventLimits *only tBeg and tEnd of events (possibly extended)*

Description

return only the limits *tBeg* and *tEnd* of the event, possibly extended by a "context"

Usage

```
eventLimits(events, context = c(0, 0), absolute = FALSE)
```

Arguments

events	events as returned by e.g. toEvents
context	Vector of two elements giving the "context" before and after the event to be plotted, in percentage of event duration. e.g. c(0.1, 0.2) means that the time interval to be plotted starts 10 percent of the event duration before the event begin and ends 20 percent of the event duration after the end of the event.
absolute	if TRUE, the context values are interpreted as absolute values (seconds) instead of fractions of the event duration. Default: FALSE

Value

data frame with columns *tBeg* and *tEnd*, taken from *events* and possibly reduced (tBeg) and/or extended (tEnd) by a fraction of the event duration (read from column *dur* in *events*).

`eventPauses` *Pauses between Events in Seconds*

Description

Pauses between Events in Seconds

Usage

```
eventPauses(
  events,
  signalWidth = .getSignalWidth(events),
  timeDifferences = NULL
)
```

Arguments

`events`	event information as returned by `hsEvents`
`signalWidth`	see description in `hsEvents`
`timeDifferences`	if time differences have been calculated beforehand, these may be given here (in seconds)

Value

data frame with columns *pBefore*, *pEnd* with the pauses between the given *events*, in seconds

---

`eventRelation` *Relations of Start/End Times of Events*

Description

Relations of Start/End Times of Events

Usage

```
eventRelation(event1, events2)
```

Arguments

event1	data frame containing exactly one row representing the event to which the event(s) in *events2* is/are to be compared. Columns *tBeg* (begin of event), *tEnd* (end of event) and *event* (event number/ID) are required.
events2	data frame containing in rows the event(s) that are to be compared to the event given in *event1*. Columns *tBeg* (begin of event), *tEnd* (end of event) and *event* (event number/ID) are required.

---

eventsByState *Get Events by Evaluation of a State Variable*

Description

Get Events by Evaluation of a State Variable

Usage

```
eventsByState(
  timestamps,
  states,
  eventSeparationTime,
  signalWidth,
  in.state = 1,
  out.state = 0
)
```

Arguments

timestamps	vector of timestamps (POSIXct)
states	vector of state values in which each element corresponds to one timestamp in *timstamps*. If the state is the value given in *in.state*) the corresponding times are considered to be lying within an event. If the state is the value given in *out.state*) the corresponding times are considered to be lying out of an event. For values that are neither of the values given in *in.state* and *out.state*, respectively, the corresponding timestamps are considered to either belong to an event or not, depending on the previous clear state ("in" or "out") in the sequence of states.
eventSeparationTime	same meaning as in hsEvents
signalWidth	see description in hsEvents
in.state	value in *states* indicating the state "in event". Default: 1
out.state	value in *states* indicating the state "out of event". Default: 0

Value

event characteristics (begin, end, duration, ...) in a data frame, as returned by hsEvents

Examples

```
# Generate random timestamps
starttime <- as.POSIXct("2015-03-12 10:51")

n <- 100
timestamps <- seq(starttime, by = 60, length.out = n)
values <- rnorm(n)

# Give values above 1 the state "in" and values below -1 the state "out"
states <- rep("", times = n)
states[values > 1] <- "in"
states[values < -1] <- "out"

# Generate the events
events <- eventsByState(
  timestamps, states, eventSeparationTime = 5 * 60, in.state = "in",
  out.state = "out", signalWidth = 60
)

# Prepare a vector of colours
col <- rep("black", length(states))
col[states == "in"] <- "green"
col[states == "out"] <- "red"

# Plot the values, the threshold lines and a legend
graphics::plot(timestamps, values, type = "l", ylim = c(-5, 5))
points(timestamps, values, col = col)

graphics::abline(h = c(1, -1), lty = 2)

legend(
  "topright", bty = "n", legend = c("in", "out"), col = c("green", "red"),
   pch = 1, bg = "white", horiz = TRUE
)

# Plot the event borders
ganttPlotEvents(events, add = TRUE, y1 = -5, y2 = 4)
```

---

eventsOnChange *Changes in Value Vector to Events*

Description

Creates "events" from vector *x* of values based on changes in the value of consecutive elements in *x*.

Usage

```
eventsOnChange(x, numberOnly = FALSE, include.value = FALSE)
```

Arguments

x	vector containing elements to be grouped into "events"
numberOnly	if TRUE, only the number of "events" is returned instead of a data frame containing first and last index of each "event".
include.value	if TRUE and *numberOnly* is FALSE, the returned data frame will contain a column *value* containing the value that was found in each index section between *iBeg* and *iEnd*.

Value

Per default (*numberOnly* = FALSE) a data frame is returned with as many rows as "events" were found in vector *x*. As long as the value in *x* does not change from one index to the next, it is assumed to belong to the same event. If the value changes, a new event begins. In the result data frame each event is represented by *iBeg* and *iEnd* which are the indices of the first and last element, respectively, in *x* that build the event. If *numberOnly* is TRUE the number of "events" is returned, that is one plus the number of changes in the value of *x* from its first to its last element.

See Also

hsEvents

Examples

```
eventsOnChange(c(1,2,2,3,4,4,4,5))

# Ouput: list of five events, i.e. there are four changes of
#        the value in the given vector.
#
#   iBeg iEnd
# 1    1    1
# 2    2    3
# 3    4    4
# 4    5    7
# 5    8    8

eventsOnChange(c(1, 2, 2, 3, 4, 4, 4, 5), numberOnly = TRUE) ## 5 (events)
```

---

eventToXLim *tBeg and tEnd of event to two-element vector*

Description

puts tBeg and tEnd of event into a vector of two POSIXct elements

Usage

```
eventToXLim(event)
```

Arguments

event	data frame with columns *tBeg*, *tEnd*

Value

vector of two elements: *tBeg* and *tEnd* from *event*, both converted to UTC timezone

---

exampleEvents *Example Events*

Description

Example events for testing purposes

Usage

```
exampleEvents(
  signalWidth = 60,
  eventSeparationTime = 60 * signalWidth,
  from = "2015-06-11",
  to = "2015-06-12",
  signalDensity = 0.01,
  ...
)
```

Arguments

signalWidth	see description in hsEvents
eventSeparationTime	see description of evtSepTime in hsEvents
from	first day as character string in format yyyy-mm-dd
to	last day as character string in format yyyy-mm-dd
signalDensity	fraction of all timestamps in a full sequence of timestamps that are to be selected randomly from the sequence and that are treated as the "signals" contributing to an event. Default: 0.01, i.e. one percent of a full sequence of timestamps are randomly selected, ordered and passed on to hsEvents that groups these "signal" timestamps into events
...	further arguments passed to hsEvents

Examples

```
events <- exampleEvents()

# Calculate event durations manually
dur <- as.integer(events$tEnd) - as.integer(events$tBeg) + hsSigWidth(events)

# All durations should be equal to the durations given in column "dur"
all(dur == events$dur)

# All pauses after event i should be equal to the pauses before event i+1
all(events$pBefore[-1] == events$pAfter[-nrow(events)])
```

`filterEventsWithStatistics` *Filter Events with Statistics*

Description

Filter Events with Statistics

Usage

```
filterEventsWithStatistics(
  eventData,
  durationThreshold = 0,
  durationComparisonOperator = "gt",
  sumThreshold = 0,
  sumComparisonOperator = "gt"
)
```

Arguments

`eventData`	eventData of one time series, as returned in one list element by `getEventsWithStatistics`.
`durationThreshold`	duration in seconds that needs to be exceeded (durationComparisonOperator == "gt") or reached (durationComparisonOperator == "ge") by the duration of the rain events. Default: 0
`durationComparisonOperator`	Operator to be applied when comparing the duration of the events with *durationThreshold*. Must be one of "gt" (greater than) or "ge" greater than or equal. Default: "gt"
`sumThreshold`	value that needs to be exceeded (sumComparisonOperator == "gt") or reached (sumComparisonOperator == "ge") by the 'sum' of values within the events. Default: 0

`sumComparisonOperator`
: Operator to be applied when comparing the 'sum' of values within the events with *sumThreshold*. Must be one of "gt" (greater than) or "ge" greater than or equal. Default: "gt"

`ganttPlotEventLists` *Gantt Plot of Event Lists*

Description

Plot event lists, one above the other

Usage

```
ganttPlotEventLists(
  eventLists,
  margin.top = 0.8,
  time.format = NULL,
  n.xticks = 10,
  showLabels = TRUE,
  ...
)
```

Arguments

`eventLists`	list of data frames containing events (containing columns *tBeg*, *tBeg*, as returned by `hsEvents`)
`margin.top`	top margin as a fraction of the total plot height
`time.format`	passed to `addTimeAxis`
`n.xticks`	passed to `addTimeAxis`
`showLabels`	passed to `ganttPlotEvents`
`...`	further arguments passed to ganttPlotEvents

`ganttPlotEvents` *Gantt-like Diagram to plot Event's Time Extension*

Description

Gantt-like Diagram to plot Event's Time Extension

Usage

```
ganttPlotEvents(
  events,
  add = FALSE,
  y1 = 1,
  y2 = y1 + 1,
  xlim = NULL,
  ylim = c(min(y1), max(y2)),
  col = "black",
  density = 5,
  showLabels = TRUE,
  eventLabels = rownames(events),
  yLabel = (y1 + y2)/2,
  type = "rectangle",
  title = "",
  leftMargin = 0.2,
  xlab = "Time",
  cex = 0.8,
  indicate = NULL,
  indicationColour = "red",
  bandheight = 0.1,
  alternating = FALSE,
  adj = 0.5,
  ...
)
```

Arguments

events	event list as retrieved by hsEvents. Required columns: *tBeg* (begin of event) and *tEnd* (end of event), both of class POSIXt
add	if TRUE, the event boxes are added to the current plot, otherwise a new plot is generated
y1	lower coordinates of the event boxes
y2	upper coordinates of the event boxes
xlim	x limits. If NULL (default) the limits will be chosen so that all events fit into the plot
ylim	y limits
col	colour of shading lines
density	density of shading lines
showLabels	if TRUE, the event boxes are labelled with the row names of the events
eventLabels	labels to be given to the events. Default: rownames(*events*)
yLabel	y-position of labels, if labels are to be shown
type	one of c("rectange", "vertical")
title	title to be plotted left of event rectangles

leftMargin	left margin (where title is printed) as fraction of the range of the total time interval spanned by the events
xlab	x axis label
cex	character expansion factor
indicate	indices of events to be indicated in a different color (indicationColuor)
indicationColour	colour to be used for indication, default: "red" extension factor for labels (event numbers)
bandheight	passed to addLabels
alternating	passed to addLabels
adj	passed to text plotting the event labels
...	further arguments passed to rect or segments

`getAndFilterEventsWithStatistics` *Get and filter Events with Statistics*

Description

Get and filter Events with Statistics

Usage

```
getAndFilterEventsWithStatistics(
  rainData,
  seriesName,
  eventSeparationTime,
  signalThreshold = 0,
  durationThreshold = 1,
  sumThreshold = 0,
  signalComparisonOperator = "gt",
  durationComparisonOperator = "gt",
  sumComparisonOperator = "gt",
  signalWidth = NA
)
```

Arguments

rainData	passed to getEventsWithStatistics
seriesName	passed to getEventsWithStatistics
eventSeparationTime	passed to getEventsWithStatistics
signalThreshold	passed to getEventsWithStatistics

durationThreshold
: passed to filterEventsWithStatistics

sumThreshold
: passed to filterEventsWithStatistics

signalComparisonOperator
: passed to getEventsWithStatistics

durationComparisonOperator
: filterEventsWithStatistics

sumComparisonOperator
: filterEventsWithStatistics

signalWidth
: passed to getEventsWithStatistics

`getEventRelations` *Begin/End Relations between Events*

Description

relations between begin and end of events

Usage

```
getEventRelations(
  events,
  referenceName,
  partnerName,
  parallelEventInfo = NULL,
  dbg = TRUE
)
```

Arguments

events	list of event lists (in a data frame), as e.g. returned by getEventsWithStatisticsForMultipleSeries
referenceName	name of column in *parallelEventInfo* containing the numbers of the events to which the events in column *partnerName* are to be compared.
partnerName	name of column in *parallelEventInfo* containing the numbers of the events which are compared to the events in column *partnerName*.
parallelEventInfo	data frame as returned by getParallelEventsInfo. If NULL, *events* must be specified.
dbg	if TRUE, debug messages are shown.

Examples

```
# Load example data set containing a list of rain events at different gauges
data(rainEvents)

cat(sprintf(
  "Event lists available for: %s\n",
   paste(names(rainEvents), collapse = ", ")
))

# How are rain events in BlnX related to rain events in Wil?
eventRelations <- getEventRelations(
  events = rainEvents,
  referenceName = "BlnX",
  partnerName = "Wil"
)

# Let's have a look at the output
eventRelations

# Example 1: partner events that are fully containing the reference events
isContaining <- eventRelations$beginRelation == "beginsBefore" &
eventRelations$endRelation == "endsAfter"

# The following table relates numbers of "partner" events (event2) to numbers
# of "reference" events (event1) for each case in which a reference event is
# fully contained in a partner event.
containing <- eventRelations[isContaining, ]

# Let's check this graphically:

# Define plot matrix of two rows and one column
old.par <- graphics::par(mfrow = c(2, 1))

ganttPlotEvents(
  events = rainEvents$BlnX[],
  indicate = containing$event1,
  ylim = c(1, 2.8),
  title = "BlnX",
  indicationColour = "blue"
)

ganttPlotEvents(
  rainEvents$Wil,
  indicate = containing$event2,
  add = TRUE,
  y1 = 1.8,
  title = "Wil"
)

graphics::title("Events at Wil (red), fully containing events at BlnX (blue)")

# Example 2: partner events that are starting before the reference event starts
```

```
# and ending before the reference event ends
isOverlappingLeft <- eventRelations$beginRelation == "beginsBefore" &
eventRelations$endRelation == "endsBefore"

overlappingLeft <- eventRelations[isOverlappingLeft, ]

# Again, check this graphically:
ganttPlotEvents(
  events = rainEvents$BlnX[],
  indicate = overlappingLeft$event1,
  ylim = c(1, 2.8),
  title = "BlnX",
  indicationColour = "blue"
)

ganttPlotEvents(
  rainEvents$Wil,
  indicate = overlappingLeft$event2,
  add = TRUE,
  y1 = 1.8,
  title = "Wil"
)

graphics::title(paste(
  "Events at Wil (red), starting before the start and ending before",
  "the end\nof the events at BlnX (blue)"
))

# Reset graphical parameters
graphics::par(old.par)
```

`getEvents` *Get Events*

Description

Get Events

Usage

```
getEvents(
  rainData,
  seriesName,
  signalThreshold = 0,
  signalComparisonOperator = "gt",
  eventSeparationTime = 6 * 3600,
  eventSeparationOperator = "gt",
  signalWidth = NA,
  column.time = names(rainData)[1]
)
```

Arguments

rainData	data frame with time stamps in the first column and rain heights (or intensities) in the remaining columns
seriesName	Column name in rainData representing the time series to be analysed.
signalThreshold	Value that needs to be exceeded (signalComparisonOperator == "gt") or reached (signalComparisonOperator == "ge") by the rain heights (or intensities) in order to be counted as a "signal". Default: 0
signalComparisonOperator	Operator to be applied when comparing rain values with signalThreshold. Must be one of "gt" (greater than) or "ge" greater than or equal. Default: "gt".
eventSeparationTime	Time difference in seconds that needs to be exceeded (eventSeparationOperator == "gt") or reached (eventSeparationOperator == "ge") by two consecutive signals in order to let the signals belong to two distinct events. Otherwise the signals are assumed to belong to one and the same event. Default: 6*3600 = six hours.
eventSeparationOperator	Operator to be applied when comparing the time differences between consecutive signals with the eventSeparationTime. Must be one of "gt" (greater than) or "ge" greater than or equal. Default: "gt".
signalWidth	signal width (= length of the time interval represented by one row in rainData) in seconds
column.time	name of the column containing the time. Default: Name of the first column

---

getEventStatistics *Get Event Statistics*

Description

Get Event Statistics

Usage

```
getEventStatistics(
  dataFrame,
  seriesName,
  events,
  functions = c("sum", "mean", "min", "max", "number.na"),
  eventNumbers = 1:nrow(events)
)
```

Arguments

dataFrame	data frame containing event data
seriesName	name of column in dataFrame
events	data frame containing event information as provided by hsGetEvent
functions	define statistical functions
eventNumbers	vector of same length as *events* has rows, giving the numbers that identify the events. Default: 1:nrow(*events*)

Value

data frame with event number in first column *event* and statistical values in further columns.

---

getEventsWithStatistics
Get Events with Statistics

Description

Get Events with Statistics

Usage

```
getEventsWithStatistics(
  rainData,
  seriesName,
  eventSeparationTime,
  signalThreshold = 0,
  signalComparisonOperator = "gt",
  eventSeparationOperator = "gt",
  functions = c("sum", "mean", "min", "max", "number.na", "length"),
  signalWidth = NA
)
```

Arguments

rainData	data frame with time stamps in the first column and rain heights (or intensities) in the remaining columns
seriesName	Column name in rainData representing the time series to be analysed.
eventSeparationTime	passed to getEvents
signalThreshold	value that needs to be exceeded (signalComparisonOperator == "gt") or reached (signalComparisonOperator == "ge") by the rain heights (or intensities) in order to be counted as a "signal". Default: 0

signalComparisonOperator
: Operator to be applied when comparing rain values with signalThreshold. Must be one of "gt" (greater than) or "ge" greater than or equal. Default: "gt"

eventSeparationOperator
: passed to getEvents

functions
: passed to getEventStatistics

signalWidth
: passed to getEvents

---

getEventsWithStatisticsForMultipleSeries

Get Events with Statistics for multiple Series

Description

Get Events with Statistics for multiple Series

Usage

```
getEventsWithStatisticsForMultipleSeries(
  rainData,
  eventSeparationTime,
  signalWidth = kwb.datetime::getTimestepInSeconds(timestamps = rainData[, 1]),
  signalThreshold = 0,
  signalComparisonOperator = "gt"
)
```

Arguments

rainData
: data frame with time stamps in the first column and rain heights (or intensities) in the remaining columns

eventSeparationTime
: "event separation time" in seconds. Maximal allowed time difference between two consecutive timestamps within the same event.

signalWidth
: "signal width" in seconds. Length of time interval that one timestamp is representing, e.g. $5 * 60 = 300$ if each timestamp respresents a time interval of five minutes (as e.g. a time series is recorded on a five minute time scale). This parameter is needed to calculate event durations.

signalThreshold
: value that needs to be exceeded (signalComparisonOperator == "gt") or reached (signalComparisonOperator == "ge") by the rain heights (or intensities) in order to be counted as a "signal". Default: 0

signalComparisonOperator
: Operator to be applied when comparing rain values with signalThreshold. Must be one of "gt" (greater than) or "ge" greater than or equal. Default: "gt"

getParallelEventNotEndingAfter *"Parallel" Events*

Description

Calculates event borders (event begin, event end) considering "parallel" events2. The returned results not ending after events1. For each event *E* in *events1* (defined by event number, event begin and event end time), this function first identifies the "partner" events E2,i from *events2* that lie within *E* or have an intersecton with *E*. There may be no, one ore more than one "partner" events.

Usage

```
getParallelEventNotEndingAfter(
  events1,
  events2,
  eventRelations,
  extended = FALSE
)
```

Arguments

events1	data frame containing the reference events, e.g. discharge events
events2	parallel events, e.g. rain events
eventRelations	event relations as returned by getEventRelations
extended	if TRUE, the output contains more columns as the minimum columns that are containd else: event1, tBeg.merged, tEnd.merged, event2first, event2last

getParallelEventsInfo *Information on Events in parallel*

Description

Information on Events in parallel

Usage

```
getParallelEventsInfo(eventLists)
```

Arguments

eventLists	list of data frames, each of which represents a list of events as e.g. generated by hsEvents

Value

data frame with timestamps in the first column indicating any begin or end of any event within *eventLists* and columns for each element of *eventLists*, containing event numbers. If you go along one row you can find the events that occur in parallel.

---

getXLimFromEventLists *overall first begin and last end of events*

Description

minimum tBeg and maximum tEnd found in event lists

Usage

```
getXLimFromEventLists(eventLists)
```

Arguments

eventLists	list of data frames containing events (containing columns *tBeg*, *tBeg*, as returned by hsEvents)

Value

vector of two elements: the first begin (minimum of tBeg) and the last end (maximum of tEnd), found in any of the event data frames given in *eventLists*

---

hsEventNumber *number timestamps according to event information*

Description

numbering timestamps according to event information

Usage

```
hsEventNumber(
  tstamps,
  events,
  eventNumbers = seq_len(nrow(events)),
  commaSeparated = FALSE
)
```

Arguments

tstamps	vector of timestamps
events	event information as returned by hsEvents
eventNumbers	optional vector of event numbers with as many elements as there are rows in *tstamps*. Default: seq_len(nrow(events))
commaSeparated	if there are timestamps taht belong to more than one event, the default behaviour (commaSeparated = FALSE) of this function is to return a list with each list element being a vector of integer numbers representing the numbers of events to which the corresponding timestamps belong. With commaSeparated = TRUE, the list of event numbers is converted into a vector of character where each element is a text string in which more than one event number are separated by a comma. E.g. c("1", "1,2", "2") would be returned if the first timestamp belongs to event 1, the second to both event 1 and 2, and the third to event 2.

---

hsEvents *Timestamp differences to events*

Description

Creates events from vector *tseries* of timestamps based on time differences between consecutive timestamps in *tseries*.

Usage

```
hsEvents(
  tseries,
  evtSepTime,
  signalWidth,
  tUnit = "s",
  pause = TRUE,
  evtSepOp = "gt",
  dbg = FALSE,
  check.sorting = FALSE
)
```

Arguments

tseries	vector containing a sorted list of timestamps.
evtSepTime	"event separation time" in seconds. Maximal allowed time difference between two consecutive timestamps within the same event.
signalWidth	"signal width" in seconds. Length of time interval that one timestamp is representing, e.g. $5 * 60 = 300$ if each timestamp respresents a time interval of five minutes (as e.g. a time series is recorded on a five minute time scale). This parameter is needed to calculate event durations.
tUnit	time unit of event duration and event pauses

pause	if TRUE, pauses before and after the events are calculated
evtSepOp	event separation operator, either "gt" or "ge". If *evtSepOp* = "gt" (default) events are separated on time differences between two consecutive timestamps that are *g*reater *t*han *evtSepTime*. If *evtSepOp* = "ge" events are separated on time differences between two consecutive timestamps that are *g*reater than or *e*qual to *evtSepTime*.
dbg	if TRUE, debug messages are shown.
check.sorting	if TRUE, it is checked whether the timestamps given in tseries are sorted and the program stops if this is not the case.

Value

data frame with columns *iBeg* and *iEnd* indicating first and last index of the event in the *tseries* vector, *tBeg* and *tEnd* indicating first and last timestamp of the event and *dur* indicating the event duration in seconds.

See Also

eventsOnChange

---

hsEventsOnChange *Deprecated. Use eventsOnChange() instead.*

Description

Deprecated. Use eventsOnChange() instead.

Usage

```
hsEventsOnChange(...)
```

Arguments

...	passed to eventsOnChange

hsEventsToUnit *Time unit conversion for events*

Description

Converts event durations and pauses before and after events to the requested time unit. The time unit will be stored in the attribute "tUnit" of the returned data frame.

Usage

```
hsEventsToUnit(evts, tUnit)
```

Arguments

evts	data frame representing events as provided by hsEvents
tUnit	time unit to which durations and pauses shall be converted.

Value

data frame containing events with durations (and pauses) given in the new time unit.

hsGetEvent *Get Sub-Timeseries belonging to Event(s)*

Description

Get Sub-Timeseries belonging to Event(s)

Usage

```
hsGetEvent(tSeries, events, evtnums, useIndex = FALSE)
```

Arguments

tSeries	data frame representing time series with first column holding the timestamp
events	event information as returned by hsEvents
evtnums	vector of event numbers to be selected
useIndex	if TRUE, *tSeries* is filtered by comparing the real row number in *tSeries* with the begin and end indices given in columns *iBeg* and *iEnd* of *events*. If FALSE, tSeries is filtered by comparing the timestamps in tSeries with the begin and end timestamps given in columns *tBeg* and *tEnd* of *events*. Defaults to TRUE if *events* contains columns *iBeg* and *iEnd*

Value

rows of *tSeries* belonging to the event numbers listed in *evtnums*

hsJoinEvents *Join events in event list*

Description

Join consecutive events in event list *evts*. The result of joining two events A and B is a event with begin time of A and end time of B.

Usage

```
hsJoinEvents(evts, ..., renumber = TRUE, dbg = FALSE)
```

Arguments

evts	data frame containing events as provided by e.g. hsEvents
...	numeric vectors containing the event numbers to be joined, e.g. 5:10, 15:20 will join events 5 to 10 and 15 to 20 to one event in each case
renumber	if TRUE, rows in result data frame are renumbered from one to number of rows.
dbg	if TRUE, debug messages are shown.

Value

A data frame with fields *tBeg*, *tEnd*, *dur* containing the times of event begin and event end and the event duration in seconds, respectively. The event duration is the difference between end and begin of the event plus the time period that one timestamp represents (signal width).

hsMergeEvents *Merge two Event Lists*

Description

Events in data frames *events1* and *events2* are merged in such a way that overlapping events are combined to one event and events that are fully contained in other events are discarded.

Usage

```
hsMergeEvents(events1, events2, renumber = TRUE, dbg = FALSE)
```

Arguments

events1	data frame containing events as provided by e.g. hsEvents
events2	data frame containing events as provided by e.g. hsEvents
renumber	if TRUE, rows in result data frame are renumbered from one to number of rows.
dbg	if TRUE, debug messages are shown.

Value

data frame with fields *tBeg*, *tEnd*, *dur* containing the times of event begin and event end and the event duration in seconds, respectively. The event duration is the difference between end and begin of the event plus the time period that one timestamp represents (signal width).

---

hsSigWidth *Find signal width in event list*

Description

Calculates signal width that was applied in event list *evts*

Usage

```
hsSigWidth(evts, dbg = FALSE)
```

Arguments

evts	data frame containing events (as e.g. provided by hsEvents)
dbg	if TRUE, debug messages are shown.

Value

signal width in seconds

---

indicesOfEventsContainedInEvent *indicesOfEventsContainedInEvent*

Description

indicesOfEventsContainedInEvent

Usage

```
indicesOfEventsContainedInEvent(events, event)
```

Arguments

events	data frame with columns *tBeg*, *tEnd*
event	data frame of one row with columns *tBeg*, *tEnd*

Value

vector of indices representing the positions of the events in *events* that are fully contained in *event*

indicesOfEventsContainingEvent *indicesOfEventsContainingEvent*

Description

indicesOfEventsContainingEvent

Usage

```
indicesOfEventsContainingEvent(events, event)
```

Arguments

events	data frame with columns *tBeg*, *tEnd*
event	data frame of one row with columns *tBeg*, *tEnd*

Value

vector of indices representing the positions of the events in *events* in which *event* is fully contained

invertedEvents *Events to Gaps between Events*

Description

"inverted" events: gaps between ends of events and begins of next events

Usage

```
invertedEvents(events)
```

Arguments

events	data frame with columns *tBeg* (begin of event) and *tEnd* (end of event), representing events

mergeAllEvents *Merge all Events*

Description

'merge' all events in a list of event lists

Usage

```
mergeAllEvents(eventList, dbg = TRUE)
```

Arguments

eventList	list of data frames, each of which represents a list of events as e.g. generated by hsEvents
dbg	if TRUE, debug messages are shown

overlapping *Are there overlapping Events?*

Description

Check event list for overlaps (begin of one event before the end of a previous event)

Usage

```
overlapping(events)
```

Arguments

events	event list as returned by hsEvents

Value

TRUE if there are overlapping events, otherwise FALSE

Examples

```
events <- kwb.event::exampleEvents()

# The example events do not overlap
overlapping(events)

# The order of the events (here reverse order) does not matter
overlapping(events[nrow(events):1, ])
```

```
# Put the begin of the second event before the end of the last event
events$tBeg[2] <- mean(c(events$tBeg[1], events$tEnd[1]))

# Now there are overlapping events!
overlapping(events)
```

`plotEventInfo` *Plot Event Info*

Description

Plot Event Info

Usage

```
plotEventInfo(eventInfo)
```

Arguments

`eventInfo`	as returned by *getParallelEventsInfo*, with first columns (timestamps) removed

---

`plotEventProperty1VersusEventProperty2`
Plot Event Property 1 versus Event Property 2

Description

Plot Event Property 1 versus Event Property 2

Usage

```
plotEventProperty1VersusEventProperty2(
  events,
  propertyName1,
  propertyName2,
  eventNumbers = events$eventNumber,
  xlab = propertyName1,
  ylab = propertyName2,
  cex = 0.7,
  ...
)
```

Arguments

events	data frame with at least two columns named as given in *propertyName1* and *propertyName2*
propertyName1	name of property to appear on the x-axis
propertyName2	name of property to appear on the y-axis
eventNumbers	vector of event numbers used for labelling. Default: rownames of *events*
xlab	default: propertyName1
ylab	default: propertyName2
cex	character expansion factor passed to plot
...	further arguments passed to plot

---

plotMergedEventInfoForValidation *Plot merged Event Info for Validation*

Description

Plot merged Event Info for Validation

Usage

```
plotMergedEventInfoForValidation(mergedEvents)
```

Arguments

mergedEvents	data frame containing information about merged events, i.e. containing columns tBeg.event1, tEnd.event1, tBeg.event2first, tEnd.event2last, tBeg.merged, tEnd.merged

---

rainEvents *Example rain events*

Description

Example lists of rain events observed at different rain gauges.

Usage

```
data(rainEvents)
```

Format

List of eleven data frames each of which represents a list of rain events. Each data frame has the columns *tBeg* (first timestamp), *tEnd* (last timestamp), *dur* (duration in minutes), *pBefore* (duration of dry period before the event in minutes), *pAfter* (duration of dry period after the event in minutes), *event* (event number), *max* (maximum rain intensity in mm/5min), *mean* (mean rain intensitiy in mm/5min), *min* (minimum rain intensity in mm/5min), *sum* (rain height in mm). To each data frame attributes are assigned that contain information about the parameters that were applied during the event creation process.

---

readEventFilesFromDirectory *Read Event Files from Directory*

Description

Read event definitions from files "events_*.txt" in event.dir

Usage

```
readEventFilesFromDirectory(
  event.dir,
  prefix = "events",
  timezone = "UTC",
  ...
)
```

Arguments

event.dir	full path to directory containing event definition files
prefix	prefix of file names to be searched for
timezone	timezone to which the timestamps are to be converted. Default: "UTC"
...	arguments passed to readEventsFromFile

---

readEventsFromFile *Read Event Limits from File*

Description

Read Event Limits from File

Usage

```
readEventsFromFile(file, timezone = "UTC", header = FALSE)
```

Arguments

file	full path to file containing the event definitions
timezone	timezone to which the timestamps are to be converted. Default: "UTC"
header	TRUE if the file contains a header line (first non-comment-line). Default: FALSE. If the file contains a header line, it must contain the column captions "tBeg" and "tEnd" (begin and end timestamps of the event).

Value

data frame with columns *tBeg* and *tEnd* (POSIXct)

---

renumberEvents *renumberEvents*

Description

add event number (= real row number) in column *event*

Usage

```
renumberEvents(events)
```

Arguments

events	event information as returned by hsEvents

Value

data frame with (additional) column *event*

---

timeDifferencesToPauses *Time Differences to Columns "pBefore" and "pAfter"*

Description

Time Differences to Columns "pBefore" and "pAfter"

Usage

```
timeDifferencesToPauses(timeDifferences, signalWidth = 0)
```

Arguments

timeDifferences	numeric vector representing time differences
signalWidth	difference between two consecutive timesteps in the original time series

Value

data frame with columns *pBefore*, *pAfter*, containing the given *timeDifferences*, shifted against each other by one row, i.e. the first element in column *pBefore* and the last element in column *pAfter* will be NA.

---

toEvents *Convert to Data Frame of Events*

Description

Convert to Data Frame of Events

Usage

```
toEvents(
  events,
  signalWidth = .getSignalWidth(events, default = NA),
  timeUnit = "s",
  pause = TRUE,
  timeDifferences = NULL
)
```

Arguments

events	data frame with columns *tBeg* (event begins) and *tEnd* (event ends)
signalWidth	see description in hsEvents
timeUnit	time unit of event duration and event pauses
pause	if TRUE, pauses before and after the events are calculated
timeDifferences	if time differences have been calculated beforehand, these may be given here (in seconds)

`validateEventFunctionArguments` *Validate Event Function Arguments*

Description

Validate Event Function Arguments

Usage

```
validateEventFunctionArguments(...)
```

Arguments

`...`	arguments, given as `key = value` pairs, to be checked for validity

`whichAboveThreshold` *Which Values are above a Threshold?*

Description

Which Values are above a Threshold?

Usage

```
whichAboveThreshold(values, threshold, comparisonOperator)
```

Arguments

`values`	numeric vector of values to be compared with the `threshold`
`threshold`	numeric value against which to check the `values`
`comparisonOperator`	if "gt" it is checked whether the `values` are *greater than* the `threshold`, otherwise it is checked whether the `values` are `greater than or equal to` the `threshold`

Index