Package: kwb.misc (via r-universe)

October 24, 2024

Title Miscellaneous functions, not yet intended for distribution

Version 0.2.2

Description Miscellaneous functions for data analysis with R at KWB,
not yet intended for distribution. So far, functions of the
following lib-files have been imported: hsLibMiaCsoData.R,
hsLibDataAvailability.R, hsLibTimeshift.R, hsLibDataSource.R,
hsLibImpSenData.R, hsLibRainDist.R, hsLibCalib.R.

License MIT + file LICENSE

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

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

Imports kwb.base, kwb.datetime, kwb.db, kwb.event, kwb.plot, kwb.read,
kwb.utils, mvbutils, RODBC

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

RoxygenNote 7.1.2

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

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

RemoteRef HEAD

RemoteSha 7c35541788230c9c027e2b1da135fa2ffa584263

Contents

add_value_labels *Add Value Labels*

Description

Add Value Labels

Usage

```
add_value_labels(x, y, labels = y, cex = 0.9)
```

Arguments

x	x coordinates
y	y coordinates
labels	labels, default: y coordinates
cex	character expansion factor, default: 0.9

`create_and_plot_rain_events` *plot rain events from given rain data*

Description

plot rain events from given rain data. Rain data is filtered for rows where signals from all gauges are available and where the sum of signals is greater than zero.

Usage

```
create_and_plot_rain_events(
  frmRain,
  strTimestamp = "Zeitstempel",
  strPdf = NULL,
  dbg = FALSE
)
```

Arguments

`frmRain`	data frame containing rain data
`strTimestamp`	name of timestamp field, default: "Zeitstempel"
`strPdf`	optional. full path to pdf output file
`dbg`	If `TRUE`, debug messages are shown

`defaultDictionary` *Get Default Dictionary from Meta Database*

Description

Get a default dictionary from the meta database

Usage

```
defaultDictionary(mdb = mmdb(), dbg = FALSE)
```

Arguments

`mdb`	full path to meta database, default: `mmdb()`
`dbg`	If `TRUE`, debug messages are shown

documentPackageFunctionDependencies *plot graphs showing the dependencies between package functions*

Description

plot graphs showing the dependencies between package functions

Usage

```
documentPackageFunctionDependencies(packagenames, to.pdf = FALSE)
```

Arguments

packagenames	vector of character containing the names of the packages of which functional dependencies are to be documented.
to.pdf	if TRUE, graphical output is written to a pdf file

Examples

```
# Show names of all installed KWB-packages
grep("^kwb\\.", library()$results[, "Package"], value = TRUE)

# Document one of the installed packages
## Not run:
documentPackageFunctionDependencies("kwb.plot")

## End(Not run)
```

example_rain_data *example rain data*

Description

example rain data

Usage

```
example_rain_data(version = 1)
```

Arguments

version	Number representing version of data to return. Default: 1

`get_rain_stat` *Get Rain Stats*

Description

Get Rain Stats

Usage

```
get_rain_stat(rain, strTimestamp = names(rain)[1])
```

Arguments

`rain`	data frame containing rain data
`strTimestamp`	Name of column containing timestamps. Default: Name of first column in `rain`

`hsAllCoefAnaCombis` *hsAllCoefAnaCombis*

Description

Generates all possible combinations of events as used for coefficient analysis

Usage

```
hsAllCoefAnaCombis(n, dbg.level = n)
```

Arguments

`n`	number of elements to be combined
`dbg.level`	if not 0, combinations are shown when dbg.level-th element just changed

`hsAllCombis` *hsAllCombis*

Description

Generate all possible combinations of elements in `x` with order mattering

Usage

```
hsAllCombis(x)
```

Arguments

`x`	vector of elements of which to create combinations

hsAllTimeshiftPlots *All Timeshift Plots*

Description

For one water quality parameter, all overflow events given in "evt" are plotted in different scales given by "fracts" (fractions of interval length). The time-series of the water quality parameter at the container, upstream and downstream are plotted over time as well as the flow.

Usage

```
hsAllTimeshiftPlots(
  frmOrig,
  frmUs,
  frmDs,
  evt,
  fieldNames,
  plusOverview = FALSE,
  type1 = "l",
  fracts = c(0, 0.75, 0.5, 0.25, 0.1, 0.05),
  fieldPrefix = ""
)
```

Arguments

`frmOrig`	Original time-series
`frmUs`	upstream shifted time-series
`frmDs`	downstream shifted time-series
`evt`	event list
`fieldNames`	vector containing the relevant table field names as named elements: tsOrig: timestamp in Q time-series), tsUsDs: timestamp in us/ds time-series), fieldQ: name of field containing Q, fieldP: name of field containing wq parameter
`plusOverview`	If true, one plot comprising the whole time-series in `frmOrig` is added
`type1`	Plot type (default: "l" = line)
`fracts`	Vector of scaling factors. For each factor a plot is generated representing the corresponding fraction of the whole time interval of the event. A factor of zero will plot the whole event.
`fieldPrefix`	Default: ""

hsAttribMatrix *Attribute Strings to Matrix*

Description

Converts a vector of attribute strings to a matrix with as many columns as there are different attributes occurring in the vector and each row representing an element of the vector for which the values of assigned attributes will occur in the corresponding attribute column.

Usage

```
hsAttribMatrix(attribs)
```

Arguments

attribs	Vector containing strings of the form "<key1>=<val1>,<key2>=<val2>,..."

hsAvailCalibs *Available Calibrations*

Description

Return names of available calibrations according to calibration database

Usage

```
hsAvailCalibs(
  moniPoint = NULL,
  parAcronym = NULL,
  skipCur = FALSE,
  mdbCal = NULL,
  dbg = FALSE
)
```

Arguments

moniPoint	acronym of monitoring point
parAcronym	acronym of parameter
skipCur	if TRUE, the name of the current specification (<parAcronym>_<moniPoint>) is excluded from the list of available calibrations
mdbCal	Path to database containing queries that getting calibrated data according to the currently active calibration setting
dbg	If `TRUE`, debug messages are shown

Value

Return character vector of (filtered) calibration names

`hsBrowseCoefAnaRes` *hsBrowseCoefAnaRes*

Description

Browse through result `tree` of regression coefficient analysis and "rbind" data frames *linreg*

Usage

```
hsBrowseCoefAnaRes(tree, combilen = -1)
```

Arguments

`tree`	list representing a `tree` structure as returned by `hsCoefAna`
`combilen`	length of combinations. Default: -1

`hsBrowseCoefAnaResList` *hsBrowseCoefAnaResList*

Description

Browse through result tree of regression coefficient analysis and "rbind" data frames *linreg*

Usage

```
hsBrowseCoefAnaResList(reslist, dbg.level = 10)
```

Arguments

`reslist`	result list as returned by hsCoefAna(..., recursive = FALSE, aslist = TRUE))
`dbg.level`	debug level

hsBrowseCombis *Browse Combinations*

Description

browses through result tree and collects all combinations

Usage

```
hsBrowseCombis(tree, combis = list())
```

Arguments

tree list representing a tree structure as returned by hsCoefAna

combis List of combinations. Default: list()

Value

list with first element containing matrix of combinations of length 1, second element containing matrix of combinations of length 2, and so on.

hsCalibAna *Analyse Calibration Uncertainty*

Description

Try all different combinations of events for calibration

Usage

```
hsCalibAna(data, dbg = TRUE, doplot = TRUE, plot.main = "", pdf)
```

Arguments

data data frame with column evtid (event ID) and further columns required by hsTestCombi to which the data frame is passed.

dbg If TRUE, debug messages are shown

doplot logical telling whether to plot within hsTestCombi. Default: TRUE

plot.main plot title. Default: ""

pdf path to PDF file

hsCoefAna *hsCoefAna*

Description

regression coefficient analysis

Usage

```
hsCoefAna(
  data,
  recursive = TRUE,
  evtNums = unique(data$evtID),
  aslist = recursive,
  uselm = FALSE,
  prep = FALSE,
  ...,
  dbg.level = max(2, length(evtNums) - 8)
)
```

Arguments

data	data frame containing columns *tstamp* (time stamp), *pval* (probe value), *lval* (lab value), *evtID* (event ID)
recursive	if TRUE, the recursive version hsCoefAnaRes of the regression coefficient analysis is used, otherwise the non-recursive version. default: TRUE
evtNums	event numbers to be considered for the analysis. Only considered when *recursive* == TRUE. default: all distinct values provided in column *evtID* of *data*)
aslist	default: boolean value given in *recursive*
uselm	if TRUE, the lm function is used to calculate the linear regression, otherwise (uselm == FALSE) the regression is calculated "manually" which is much faster. Default: FALSE
prep	Default: FALSE
...	further arguments passed to hsCombiLinReg, e.g. *clever*
dbg.level	default: max(2, length(evtNums) - 8)

hsCoefAnaNonRec *hsCoefAnaNonRec*

Description

non-recursive version of regression coefficient analysis

Usage

```
hsCoefAnaNonRec(data, uselm = FALSE, aslist = FALSE, ..., dbg.level = 1)
```

Arguments

data	data frame containing columns *tstamp* (time stamp), *pval* (probe value), *lval* (lab value), *evtID* (event ID)
uselm	if TRUE, the lm function is used to calculate the linear regression, otherwise (uselm == FALSE) the regression is calculated "manually" which is much faster. default: FALSE
aslist	if TRUE the result is retunred in forms of a list with each list element representing one combination. otherwise in forms of a database with columns *np* (number of points), *offset*, *slope*, *combi*. Default: FALSE.
...	further arguments passed to hsCombiLinReg, e.g. *clever*
dbg.level	debug level

hsCoefAnaRec *hsCoefAnaRec*

Description

recursive version of regression coefficient analysis

Usage

```
hsCoefAnaRec(
  data,
  evtNums = unique(data$evtID),
  combi = NULL,
  tree = TRUE,
  resframe = NULL,
  uselm = FALSE,
  ...,
  dbg.level = 1
)
```

Arguments

`data` `data` frame containing columns *tstamp* (time stamp), *pval* (probe value), *lval* (lab value), *evtID* (event ID)

`evtNums` event numbers to be considered for the analysis (default: all distinct values provided in column *evtID* of *data*)

`combi` current combination to be evaluated and to be the base for the next combinations to be determined

`tree` if TRUE, result is given in a `tree` structure, otherwise as a `data` frame

`resframe` if *tree* is FALSE, this argument contains the results that have been found so far in a `data` frame

`uselm` if TRUE, the lm function is used to calculate the linear regression, otherwise (`uselm` == FALSE) the regression is calculated "manually" which is much faster. default: FALSE

`...` further arguments passed to `hsCombiLinReg`, e.g. *clever*

`dbg.level` debug level

Value

Recursive list representing a `tree` structure. At the top level the list contains elements *e<i>* where <i> are the event IDs to be considered (elements in *evtNums*). The sub lists below the top level (but not the "leafs" of the `tree`) also contain elements *e<j>* where <j> are the "remaining" event IDs, i.e. the IDs that do not yet occur in the "path" of event IDs leading to the respective sub `tree`. These sub lists also have elements *combi* (vector of event IDs representing the respective event combination) and *linreg* containing the results from linear regression. In fact, *linreg* is a `data` frame with each line representing the *slope* and *offset* of the linear regression through *np* number of points, taken from the events in *combi*.

`hsCombiLinReg` *Linear Regression for Event Combination*

Description

Calculation of linear regressions for given combination of events

Usage

```
hsCombiLinReg(
  data,
  combi,
  uselm = FALSE,
  clever = FALSE,
  prep = FALSE,
  calc.rmse = TRUE,
  dbg = FALSE
)
```

Arguments

`data` — `data` frame containing columns *tstamp* (time stamp), *pval* (probe value), *lval* (lab value), *evtID* (event ID)

`combi` — combination of events for which linear regressions are to be calculated in the following way: the first event numbers in `combi`, at positions `seq_len(length(combi) - 1)`, are considered to be "base" events, i.e. events of which all `data` points are considered for the linear regression. The `data` points belonging to the event given at the last position of `combi` are added "point by point" to these "base points" and each time a separate regression is calculated

`uselm` — if TRUE, the lm function is used to calculate the linear regression, otherwise (`uselm` == FALSE) the regression is calculated "manually" which is much faster. default: FALSE

`clever` — if TRUE, sums and means are updated by knowledge of previous values with the current `data` point, otherwise they are always recalculated for all datapoints to be considered

`prep` — if TRUE, `data` is expected to contain columns *x2* (squares of x), *xy* (product of x and y values). Unfortunately, this does not give a better performance...

`calc.rmse` — if TRUE, the root mean square error (RMSE) is calculated

`dbg` — If `TRUE`, debug messages are shown

`hsCurCal` *Current calibration*

Description

Get calibrated data according to current calibration

Usage

```
hsCurCal(moniPoint, parAcronym, globOnly = FALSE, mdbCal = NULL)
```

Arguments

`moniPoint` — acronym of monitoring point

`parAcronym` — acronym of parameter

`globOnly` — logical. Globals only? Default: FALSE

`mdbCal` — Path to database containing queries that getting calibrated data according to the currently active calibration setting

hsDataAvailability *Data Availability in time-series data*

Description

data availability in time-series data

Usage

```
hsDataAvailability(
  data,
  tstep = minTimeStep(data[[1]], dbg = dbg),
  interval = 60 * 60 * 24,
  includeCount = TRUE,
  dbg = FALSE
)
```

Arguments

data	data frame with timestamp in first column
tstep	expected timestep between consecutive timestamps in seconds. Default: minimum time difference occurring in timestamps of *data*.
interval	length of time intervals to which data is grouped, in seconds. Default: 60*60*24 = one day intervals; data availability is calculated separately for each time interval.
includeCount	if TRUE, not only the data availability in percent but also the number of records per interval from which the percentage has been calculated are included as separate columns in the result data frame.
dbg	If TRUE, debug messages are shown

hsDataAvailability.old *Data availability of time series data*

Description

Data availability of time series data

Usage

```
hsDataAvailability.old(
  info,
  dateFirst = NULL,
  dateLast = NULL,
  tstep = NULL,
  dbg = FALSE
)
```

Arguments

`info` list with the named elements (*mdb*: full path to Access database, *tbl*: table name, *tsField*: name of timestamp field, *parField*: name of parameter field)

`dateFirst` Date object representing first date to be considered

`dateLast` Date object representing last date to be considered

`tstep` expected time step between time stamps in seconds. Default: minimum time difference found between consecutive timestamps in given interval

`dbg` If TRUE, debug messages will be shown

Value

data.frame with each row representing a day within the specified time interval and columns $intervalBeg$ (day), $n < Par >$ (number of non-NA-values in column <Par> within the interval) and $p < Par >$ (data availability of parameter <Par> in percent = number of available non-NA-values divided by maximum possible number of non-NA-values per day (= 86400 / *tstep*).

`hsdbg` *Call browser() in Debug Mode*

Description

Call browser() in Debug Mode

Usage

```
hsdbg()
```

`hsDirStructure` *Read Directory Structure from Metadata Database*

Description

Gets recursively defined directory structure from RMeta.mdb

Usage

```
hsDirStructure(asMatrix = FALSE, dbg = FALSE)
```

Arguments

`asMatrix` logical indicating whether to return a matrix or not

`dbg` If `TRUE`, debug messages are shown

`hsDropExistingTable` *Drop an existing table (user interaction)*

Description

Drop an existing table (user interaction)

Usage

```
hsDropExistingTable(channel, strTable, boolAsk = TRUE)
```

Arguments

`channel`	database connection
`strTable`	table name
`boolAsk`	logical indication whether the user should be asked before dropping the table

Value

table name of created table.

`hsEqualUntilPos` *hsEqualUntilPos*

Description

returns the first index at which elements in `combi1` and `combi2` differ

Usage

```
hsEqualUntilPos(combi1, combi2)
```

Arguments

`combi1`	vector
`combi2`	vector of same mode as `combi1`

hsExampleCoefData *hsExampleCoefData*

Description

provide example dataset for coefficient analysis with hsCoefAna

Usage

```
hsExampleCoefData(
  nevts = 5,
  step = 30,
  ex.avail = FALSE,
  dev = 0.01,
  dbg = FALSE
)
```

Arguments

nevts	number of gaps to be produced in continuous examle data in order to split events. default: 5
step	timestep in seconds. default: 30
ex.avail	example data available? default: FALSE
dev	deviation. default: 0.01
dbg	If TRUE, debug messages are shown

hsFileCands *File Candidates*

Description

find file candidates according to paths defined in dictionary

Usage

```
hsFileCands(mdb, dict.lst, dbg = FALSE)
```

Arguments

mdb	path to MS Access database
dict.lst	path dictionary (a list)
dbg	If TRUE, debug messages are shown

hsGetOrCreateParID *hsGetOrCreateParID*

Description

Lookup a water quality parameter defined by (SEN-ID, name, unit)

Usage

```
hsGetOrCreateParID(parInfo, parDefs)
```

Arguments

`parInfo`	parameter definition
`parDefs`	list of parameter definitions

Value

List with two elements *myParID* and *parDefs*. If the parameter defined in *parInfo* was found in the list of all parameters *parDefs* that have been found so far, the unique parameter ID used in `parDefs` is returned in *myParID*. If the parameter was not found it is added to `parDefs` and

hsGetParID *hsGetParID*

Description

Lookup name of water quality parameter and return its ID or NA if the parameter is not yet contained in the data frame `wqpNames`

Usage

```
hsGetParID(parName, wqpNames)
```

Arguments

`parName`	parameter name
`wqpNames`	data frame describing (water quality) parameters

hsGetSqls *hsGetSqls*

Description

Returns vector of SQL strings each of which selects the values of one water quality parameter (in one column) from table “tbl” giving general column names

Usage

```
hsGetSqls(mdb, tbl, mpID, belowAbove = FALSE, bis2007 = FALSE)
```

Arguments

`mdb`	path to MS Access database
`tbl`	table name
`mpID`	monitoring point ID
`belowAbove`	Default: FALSE
`bis2007`	Default: FALSE

hsGsDataListToMdb *hsGsDataListToMdb*

Description

write table containing all grab sample data in "all-in-one-table"-format to `mdb` database

Usage

```
hsGsDataListToMdb(gsDataList, mdb, ...)
```

Arguments

`gsDataList`	List of grab sample data sets
`mdb`	path to MS Access database
`...`	arguments passed to hsPutTable

hsImpGsData *import SENATE's grab sample data from* csv

Description

import SENATE's grab sample data from csv

Usage

```
hsImpGsData(
  csv,
  mdb,
  sep = ",",
  dateFormat = underscoreToPercent("_d/_m/_Y"),
  blockBeginPtrn = "Messstelle",
  tblNamePtrn = "tblSenGrabSmp_@M",
  boolAsk = TRUE,
  dbg = FALSE
)
```

Arguments

csv	full path to csv file
mdb	full path to MS Access database (*.mdb)
sep	separator in csv file
dateFormat	date format in csv file (default: %d/%m/%Y)
blockBeginPtrn	pattern indicating the begin of a data block in the file (default: Messstelle)
tblNamePtrn	table name pattern. Default: "tblSenGrabSmp_@M" with the placeholder @M being replaced with the acronym of the monitoring point
boolAsk	logical passed to hsWriteBlockToTable
dbg	If TRUE, debug messages are shown

hsIntSumGeThreshold *Interval Sum >= Threshold*

Description

For each index <iStart> of vector *values*, this function tries to find a corresponding index <iStop> in such a way that the sum of the vector elements at indices between <iStart> and <iStop> reaches the given threshold. For each possible start index i, the algorithm starts either looking forward at indices i+1, i+2, ... or backwards at indices i-1, i-2, ..., accumulating the values at these indices. Once the accumulated sum reached the given threshold or if the difference between the indices exceeds the maximum allowed index difference *maxDist* the algorithm stops and continues with the next start index.

Usage

```
hsIntSumGeThreshold(
  tSeries,
  threshold,
  forward,
  maxTDiff,
  tsField = names(tSeries)[1],
  valField = names(tSeries)[2],
  valFactor = 1,
  includeIndices = TRUE,
  dbg = FALSE
)
```

Arguments

`tSeries`	data.frame with timestamps in first column and values in second column.
`threshold`	Threshold that shall be reached/exceeded by the sum of successive elements of *values* of which the maximum time difference is below or equal *maxTDiff*.
`forward`	If TRUE, the algorithm looks `forward`, else backwards, i.e. when looking `forward` (backwards), the start indices <iStart> are always less or equal (greater or equal) the assigned indices <iStop>.
`maxTDiff`	Maximum allowed time difference in seconds between two related timestamps.
`tsField`	Name of time stamp field; default: name of first column
`valField`	Name of value field containing the values of which the sum shall reach the `threshold`; default: name second column
`valFactor`	Factor to be applied to column *valField* before calculating value sums.
`includeIndices`	if TRUE, two columns *iStart* and *iStop* are included in the output data frame indicating the indices in *tSeries* corresponding to the timestamps *tStart* and *tStop*.
`dbg`	If `TRUE`, debug messages are shown

Value

data frame with columns *iStart* and *iStop* being the indices of *tSeries* that represent the beginning and the end of the time interval in which the value field sums up to at least *threshold*, *tStart*, *tStop* and *tDiff* representing the first timestamp, last timestamp and duration of the corresponding time intervals and the column *sumWithin* being the sum of values that was actially reached within the interval.

`hsMergeAvailCalibs` *Merge Data in Available Calibrations*

Description

Merge Data in Available Calibrations

Usage

```
hsMergeAvailCalibs(moniPoint, parAcronym, mdbCal, ...)
```

Arguments

`moniPoint`	acronym of monitoring point
`parAcronym`	acronym of parameter
`mdbCal`	path to MS Access database containing calibration data
`...`	arguments given to `hsPlotEventOverview`

Value

data frame with the first two columns representing the timestamp and the global calibration, respectively and the following columns representing available calibrations, beginning with the "current" calibration that is stored in the calibration database under the name <parAcronym>_<moniPoint>, e.g. "AFS_STA"

`hsMiaDir` *hsMiaDir*

Description

Return directory paths containing mdb databases matching the criteria defined by the `...` parameter list

Usage

```
hsMiaDir(dbg = FALSE, ...)
```

Arguments

`dbg`	If `TRUE`, debug messages are shown
`...`	assignments of the form "name = value"

`hsNextCoefAnaCombi` *hsNextCoefAnaCombi*

Description

given a current combination the next combination of events as used for coefficient analysis is provided

Usage

```
hsNextCoefAnaCombi(combi, n)
```

Arguments

`combi`	current combination of which the "successor" combination is to be provided
`n`	number of elements in the combination (= length of combination)

`hsPlotCalOverview` *Plot Results of Calibration Analysis*

Description

Plot Results of Calibration Analysis

Usage

```
hsPlotCalOverview(moniPoint, parAcronym, mdbCal, pdf, ...)
```

Arguments

`moniPoint`	acronym of monitoring point
`parAcronym`	acronym of parameter
`mdbCal`	path to MS Access database containing calibration data
`pdf`	path to PDF file to be created
`...`	arguments given to `hsPlotEventOverview`

`hsPlotCoefAnaRes` *hsPlotCoefAnaRes*

Description

Plot function to visualise the regression lines calculated by `hsCoefAna`.

Usage

```
hsPlotCoefAnaRes(data, res, recursive = FALSE)
```

Arguments

`data`	`data` frame containing columns *tstamp* (time stamp), *pval* (probe value), *lval* (lab value), *evtID* (event ID)
`res`	result tree as returned by `hsCoefAna`.
`recursive`	Default: FALSE

`hsPlotCoefAnaRes2` *hsPlotCoefAnaRes2*

Description

Plot function to visualise the distribution of slopes and offsets of regression lines through possible combinations of events.

Usage

```
hsPlotCoefAnaRes2(data, res, olim = NULL, slim = NULL)
```

Arguments

`data`	`data` frame containing columns *tstamp* (time stamp), *pval* (probe value), *lval* (lab value), *evtID* (event ID)
`res`	result tree as returned by `hsCoefAna`.
`olim`	limits of offset values to be used for plotting the offsets
`slim`	limits of slope values to be used for plotting the offsets

hsPlotDataAvailability *Plot Data Availability*

Description

barplot showing data availability in (e.g. daily) time intervals.

Usage

```
hsPlotDataAvailability(
  avail,
  colNames = NULL,
  firstIntBeg = NULL,
  lastIntBeg = NULL,
  main = "hsPlotDataAvailability",
  barCols = NULL,
  labelStep = 2,
  firstPlot = TRUE,
  dbg = FALSE,
  ...
)
```

Arguments

`avail`	data frame containing the availibility information as returned by `hsDataAvailability`
`colNames`	name of column containing availabilities
`firstIntBeg`	timestamp indicating the begin of the first interval to be plotted
`lastIntBeg`	timestamp indicating the begin of the last interval to be plotted
`main`	`main` title of barplot
`barCols`	bar colour(s). If `avail` is a list of data frames, each data frame is shown in its own colour as given here in `barCols`
`labelStep`	if set to <n>, only every n-th date label will be shown in the plot
`firstPlot`	if TRUE, barplot is replotted, else plot is added to existing plot
`dbg`	If `TRUE`, debug messages are shown
`...`	further arguments to be passed to R’s barplot() function.

hsPlotEventOverview *Plot Event Overview*

Description

Plot Event Overview

Usage

```
hsPlotEventOverview(
  dat,
  evts = NULL,
  evtSepTime = 3600,
  myTitle = "Event Overview",
  plotTypes = rep("l", ncol(dat) - 1),
  dbg = TRUE,
  ...
)
```

Arguments

dat	data frame with at least two columns and the timestamps being in the first column
evts	data frame with columns *tBeg* and *tEnd* containing first and last timestamp, respectively, of the events. If NULL (default) the events are generated by calling hsEvents (using all timestamps in dat, this is maybe not what we want!!!)
evtSepTime	event separation time in seconds, default: 3600 (= 1h)
myTitle	plot title
plotTypes	vector containing the plot type (cp. type argument of plot function) for each data column to be plotted
dbg	If TRUE, debug messages are shown
...	e.g. inset = ...

hsPrintToPlot *hsPrintToPlot*

Description

prints content of an object to a plot

Usage

```
hsPrintToPlot(data, main = "Printed by hsPrintToPlot", addLines = NULL, ...)
```

Arguments

`data`	object to print
`main`	plot title
`addLines`	additional lines
`...`	additional arguments passed to legend, e.g. cex

`hsSel` *Filter Timeseries Interactively*

Description

Filter timeseries interactively for `interval` or parallel intervals

Usage

```
hsSel(tseries, interval = FALSE, tsfield = names(tseries)[1], dbg = FALSE)
```

Arguments

`tseries`	data frame containing time series data
`interval`	logical. Default: FALSE
`tsfield`	Name of column containing the time stamps. Default: name of first column
`dbg`	If `TRUE`, debug messages are shown

`hsSelectParallelTimeIntervals` *Interactively Select Parallel Time Intervals*

Description

Interactively select parallel time intervals from data.frame containing timeseries

Usage

```
hsSelectParallelTimeIntervals(tseries, tsfield = names(tseries)[1])
```

Arguments

`tseries`	data.frame containing timeseries (at least one timestamp column and one additional numeric column)
`tsfield`	name of timestamp column; default: name of first column in `tseries`

hsSelectTimeInterval *Interactively Select Time Interval*

Description

Interactively select time interval from data.frame containing timeseries

Usage

```
hsSelectTimeInterval(tseries, tsfield = names(tseries)[1])
```

Arguments

`tseries`	data.frame containing timeseries (at least one timestamp column and one additional numeric column)
`tsfield`	name of timestamp column; default: name of first column in `tseries`

hsSourceList *hsSourceList*

Description

Return data frame containing ids, properties and paths of mdb databases matching criteria given in `...` argument list

Usage

```
hsSourceList(keyptrn = "^DB_", dbg = FALSE, ...)
```

Arguments

`keyptrn`	Pattern matching the keys representing databases. Default: "^DB_"
`dbg`	If `TRUE`, debug messages are shown
`...`	definition of filter criteria given as "key = value" pairs

hsSpecCal *Special calibration*

Description

Get calibrated data according to a special calibration given by its name. Instead of running the prepared query in “KWB_CAL.mdb” the specific SQL query respecting the given calibratio name is built and run here.

Usage

```
hsSpecCal(moniPoint, parAcronym, calName = NULL, mdbCal = NULL)
```

Arguments

moniPoint	acronym of monitoring point
parAcronym	acronym of parameter
calName	name of calibration
mdbCal	Path to database containing queries that getting calibrated data according to the currently active calibration setting

hsSqlExCal *SQL to Get Calibrated Data*

Description

Generate SQL expression needed to get calibrated data

Usage

```
hsSqlExCal(moniPoint, parAcronym, calName = NULL)
```

Arguments

moniPoint	acronym of monitoring point
parAcronym	acronym of parameter
calName	name of calibration

hsTestCombi *Test Combination of Events*

Description

Evaluate a specific combination of events used for calibration

Usage

```
hsTestCombi(
  data,
  combi,
  plot = TRUE,
  plot.main = "Lab values vs probe values",
  COLS = rainbow(length(unique(data$evtid)))
)
```

Arguments

data	data frame
combi	combination of events used for calibration
plot	logical indicating whether to plot
plot.main	plot title
COLS	vector of colours

hsTimeshift *Timeshift*

Description

`upstream` or downstream "timeshift" of water quality data given time-series of hydraulic and water quality data in one data frame

Usage

```
hsTimeshift(
  hq,
  threshold,
  upstream = TRUE,
  tsField = names(hq)[1],
  valField = names(hq)[2],
  quaFields = names(hq)[-c(1, 2)],
  maxTDiff = 3600,
  valFactor = 1,
  dbg = FALSE
)
```

Arguments

`hq` hydraulic and water quality data in one data frame

`threshold` Threshold that shall be reached/exceeded by the sum of successive values in column *valField* of which the maximum time difference is below or equal *maxTDiff*.

`upstream` if TRUE, the algorithm “looks” `upstream`, else downstream

`tsField` name of timestamp field in *hq*

`valField` name of column in *hq* containing the values to be summed up until the `threshold` is reached

`quaFields` vector containing column names of water quality parameters, e.g. c("AFS", "CSB", "CSBf")

`maxTDiff` Maximum allowed time difference in seconds between two related timestamps.

`valFactor` factor to be applied to column *valField* before calculating value sums.

`dbg` If `TRUE`, debug messages are shown

`hsTimeshiftPlot` *Timeshift Plot*

Description

A plot is generated, containing flow measurements as well as measurements (original, upstream/downstream time-shifted) of one water quality parameter and for one overflow events `i` contained in the event list "evt".

Usage

```
hsTimeshiftPlot(
  frmOrig,
  frmUs,
  frmDs,
  evt,
  fieldNames,
  i,
  evtDurFract,
  type1,
  boolOverview = FALSE,
  fieldPrefix = ""
)
```

Arguments

`frmOrig`	data frame with original measurements
`frmUs`	data frame with measurements time-shifted upstream
`frmDs`	data frame with measurements time-shifted downstream
`evt`	event list
`fieldNames`	field names
`i`	index
`evtDurFract`	event duration fraction
`type1`	plot type, e.g. "p" for "points", passed to `plot`
`boolOverview`	should an overview be plotted?
`fieldPrefix`	field prefix

`hsUnionSqls` *hsUnionSqls*

Description

Creates UNION-queries of given SQL queries, respecting the maximum number of subqueries to be used in one and the same UNION query.

Usage

```
hsUnionSqls(sqls, maxUnions)
```

Arguments

`sqls`	vector of SQL statements
`maxUnions`	number of maximum UNIONs allowed

Value

vector of UNION queries

hsWriteBlockToTable *write data block to database table*

Description

write data block to database table

Usage

```
hsWriteBlockToTable(
  channel,
  block,
  blockName,
  tblNamePtrn = "tblSenGrabSmp_@M",
  boolAsk,
  dbg = FALSE
)
```

Arguments

channel	database connection
block	block
blockName	block name
tblNamePtrn	name of table to be written in database. @M is replaced with the name of the monitoring point
boolAsk	logical. Ask before tropping existing tables?
dbg	If TRUE, debug messages are shown

mmdb *Path to Metadata Database*

Description

Path to Metadata Database

Usage

```
mmdb(id = 0)
```

Arguments

id	id of record. Currently, only 0 allowed (the default)

`na_checksum` *number representing combination of gauges with NA-failure*

Description

calculate checksum for each row of a data frame or matrix *x*. Each combination of NA-occurrence in the row gets a different checksum. Therefore, each column of *x* is represented as a power of two and for the columns in which the row contains NA the corresponding powers of two are added.

Usage

```
na_checksum(x)
```

Arguments

`x`	data frame or matrix

`pageAndPlot` *output object to plots of same row number*

Description

capture the output of printing an object, split this output into blocks of equal size (row per page) and print these blocks as plots using `hsPrintToPlot`

Usage

```
pageAndPlot(data, rpp = 60, to.pdf = TRUE, ...)
```

Arguments

`data`	`data` frame to plot to pdf
`rpp`	rows per page
`to.pdf`	if TRUE (default) the output goes into a temporary PDF file, otherwise to the standard plot device
`...`	arguments to be passed to `hsPrintToPlot`, e.g. main, cex

plotDataAvailability *Plot Data Availability*

Description

Plot Data Availability

Usage

```
plotDataAvailability(
  rainStat,
  n = nrow(rainStat),
  main = "Data Availability",
  cex = 0.9,
  barColours = c("lightgreen", "indianred")
)
```

Arguments

rainStat	rain statistics with one row per rain event
n	number of rain event
main	plot title
cex	character expansion factor. Default: 0.9
barColours	vector of bar colours

plotRainOverview *Plot Rain Overview*

Description

minDate and maxDate must be given as "yyyy-mm-dd" Timestamps of the day given in minDate are included, whereas timestamps of the day given in maxDay are excluded! This way it is easy to select whole months or whole years by selecting the first day of the next month/year as maxDate. It is assumed that all but the first columns contain rain heights!

Usage

```
plotRainOverview(
  rain,
  minDate = "",
  maxDate = "",
  strTimestamp = "Zeitstempel",
  boolMaxDateIncluded = FALSE,
  Nmax_mm = NA,
  cex = 0.9
)
```

Arguments

`rain`	data frame containing rain data
`minDate`	minimum date
`maxDate`	maximum date
`strTimestamp`	name of timestamp column. Default: "Zeitstempel"
`boolMaxDateIncluded`	logical. Should the maximum date be included?
`Nmax_mm`	Default: NA
`cex`	character expansion factor. Default: 0.9

`plotTotalPrecipitation` *Plot Total Precipitation*

Description

Plot Total Precipitation

Usage

```
plotTotalPrecipitation(
  rainStat,
  n = nrow(rainStat),
  main = "Total Precipitation",
  Nmax_mm = NA,
  cex = 0.9
)
```

Arguments

`rainStat`	data frame with column *N_mm* and row names indicating rain gauge names
`n`	number of rain series
`main`	plot title
`Nmax_mm`	maximum total precipitation in mm
`cex`	character expansion factor. Default: 0.9

plot_given_rain_events

plot rain events from rain data and event info

Description

plot all rain events from given rain data and event information. This function calls plot_one_rain_event_2 in a loop over all events with indices between *imin* and *imax*.

Usage

```
plot_given_rain_events(
  rain,
  evt,
  imin = 1,
  imax = -1,
  sigWidth_s,
  myCex = 0.7,
  myMinN
)
```

Arguments

rain	data frame containing rain data
evt	event information as returned by kwb.event::hsEvents
imin	row index in *evt* of first event to be considered. Default: 1
imax	row index in *evt* of last event to be considered. Set to -1 (default) to consider all events.
sigWidth_s	rain signal width in seconds
myCex	default: 0.7
myMinN	minimum total precipitation in mm. Rain events with less precipitation are not plotted

plot_one_rain_event *Plot One Rain Event*

Description

Plot One Rain Event

Usage

```
plot_one_rain_event(frmR, evt, i)
```

Arguments

`frmR`	data frame containing rain data
`evt`	data frame containing event data
`i`	index into event data

`plot_one_rain_event_2` *Plot One Rain Event*

Description

plot one `rain` event by calling `plot_one_rain_gauge_event`

Usage

```
plot_one_rain_event_2(
  rain,
  rainEventIndex = 1,
  sigWidth_s = NULL,
  boolBarplot = TRUE,
  myCex = 1,
  myMinN = 0
)
```

Arguments

`rain`	data frame containing rain data
`rainEventIndex`	rain event index
`sigWidth_s`	`rain` signal width in seconds
`boolBarplot`	logical. Default: `TRUE`
`myCex`	character expansion factor. Default: 1
`myMinN`	minimum total precipitation. Default: 0

`plot_one_rain_event_3` *Plot One Rain Event 3*

Description

Plot One Rain Event 3

Usage

```
plot_one_rain_event_3(frmR, evt, i, myCols)
```

Arguments

`frmR`	data frame containing rain data
`evt`	data frame containing rain event data
`i`	index
`myCols`	vector of colours

`plot_one_rain_gauge_event` *Plot Rain Event*

Description

Plot Rain Event

Usage

```
plot_one_rain_gauge_event(
  rain,
  sigWidth_s = NULL,
  i = 2,
  Nmax = 1,
  boolLabels = FALSE,
  boolDebug = FALSE,
  myBlue = rainbow(8)[6],
  nMaxLabels = 60,
  dbg = FALSE
)
```

Arguments

`rain`	data frame containing rain data
`sigWidth_s`	signal width in seconds
`i`	Default: 2
`Nmax`	Default: 1
`boolLabels`	plot labels?
`boolDebug`	logical
`myBlue`	nice blue colour
`nMaxLabels`	# max number of labels per page
`dbg`	If `TRUE`, debug messages are shown

plot_rain_events *plot rain events from given rain data*

Description

plot rain events from given rain data. Either one plot per day or one plot per rain event (created within this function). This function calls plot_given_rain_events

Usage

```
plot_rain_events(
  rain,
  strTimestamp = "Zeitstempel",
  strPdf = NULL,
  irng = c(1, -1),
  myCex = 0.7,
  myMinN,
  evtSepTime_s = 60 * 60 * 6,
  sigWidth_s = 300,
  events = TRUE
)
```

Arguments

`rain` data frame containing rain data

`strTimestamp` name of timestamp field, default: "Zeitstempel"

`strPdf` optional. path to output pdf file

`irng` vector of two integer values determining the first and last index, respectively, in the data frame of events (created within this function) to be plotted. Second element can be -1 to indicate the index of the last event. Default: c(1, -1)

`myCex` default: 0.7

`myMinN` minimum precipitation in mm. Events with a total depth less than this value are not plotted

`evtSepTime_s` event separation time in seconds. Default: 6*3600 = 6 hours

`sigWidth_s` rain signal width in seconds. Default: 300 = 5 minutes

`events` one plot per event (events = TRUE) or one plot per day (events = FALSE)? default: TRUE

`plot_rain_events_to_pdf`

Plot Rain Events to PDF

Description

Plot Rain Events to PDF

Usage

```
plot_rain_events_to_pdf(frmR, evt, strPdf, myCols, myVersion = 1)
```

Arguments

`frmR`	data frame containing rain data
`evt`	data frame containing rain event data
`strPdf`	path to PDF file
`myCols`	vector of colours
`myVersion`	Default: 1

`plot_rain_subevents_by_checksum`

Plot Rain Subevents by Checksum

Description

Plot Rain Subevents by Checksum

Usage

```
plot_rain_subevents_by_checksum(rain, evt, n = nrow(evt))
```

Arguments

`rain`	data frame containing rain data
`evt`	data frame containin rain event data
`n`	Default: number of rows in `evt`

plot_rain_with_subevent_lines *Plot Rain with Subevent Lines*

Description

Plot Rain with Subevent Lines

Usage

```
plot_rain_with_subevent_lines(rain, subevt, rainEventIndex)
```

Arguments

rain	data frame containing rain data
subevt	subevent data
rainEventIndex	rain event index

prepare_rain_data *Prepare Rain Data*

Description

Prepare Rain Data

Usage

```
prepare_rain_data(
  rain,
  myVersion,
  event_sep_time = 60 * 60 * 6,
  signal_width = 300
)
```

Arguments

rain	rain data
myVersion	1 or 2, depending on required output
event_sep_time	event separation time in seconds
signal_width	signal width, i.e. time period that a rain data point represents, in seconds

`read_rain_from_mdb` *Read Rain Data from Database*

Description

Read Rain Data from Database

Usage

```
read_rain_from_mdb(myVersion, strDb)
```

Arguments

`myVersion` 1 (to read from tbl_Regen_alleEZG_05min) or 2 (to read from tbl_Regen_alleEZG_05min)

`strDb` path to MS Access database

Index