Package: kwb.BerlinWaterModel.public (via r-universe)

May 19, 2026

Title R Package of Berlin Water Model

Version 0.2.0

Description R Package of Berlin Water Model.

License MIT + file LICENSE

URL https://github.com/KWB-R/kwb.BerlinWaterModel.public

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

Depends R (>= 3.5)

Imports dplyr, ggplot2, gridExtra, htmlwidgets, igraph, kwb.utils, lubridate, magrittr, matrixStats, networkD3, purrr, readr, rlang, stringr, tibble, tidyr, tidyselect, units

Suggests covr, DT, knitr, openxlsx, rmarkdown

VignetteBuilder knitr

Remotes github::kwb-r/kwb.utils

Encoding UTF-8

LazyData true

LazyDataCompression bzip2

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/pak/sysreqs cmake libglpk-dev make libicu-dev libuv1-dev libxml2-dev libudunits2-dev libx11-dev

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

Date/Publication 2026-03-20 21:18:11 UTC

RemoteUrl https://github.com/KWB-R/kwb.BerlinWaterModel.public

RemoteRef HEAD

RemoteSha e0368ae31bb833a080502a2ab5461b52183bf350

Contents

`add_rain_direct_and_evaporation` *Add Direct Rain and Evaporation to flows_in_out config for sections with defined area_m2*

Description

Add Direct Rain and Evaporation to flows_in_out config for sections with defined area_m2

Usage

```
add_rain_direct_and_evaporation(config)
```

Arguments

`config`	config object as retrieved by `config_read`

Value

config with added flows_in_out for sections with defined area_m2, in case none area_m2 is defined the original config is returned

add_scenario *Add scenario*

Description

Add scenario

Usage

```
add_scenario(config, use_scenario = FALSE, debug = TRUE)
```

Arguments

config	config as retrieved by config_read
use_scenario	use scenario (default: FALSE)
debug	print debug messages (default: TRUE)

Value

list with input parameters required for function prepare_input

add_substances *Add substances*

Description

Add substances

Usage

```
add_substances(config)
```

Arguments

config	config as retrieved by config_read

Value

adds new columns 'conc_<substance-name.kg.m3' and re-calculates input concentrations from (ng|ug|mg|g)/L to kg/m3

`add_tracers` *Add tracers*

Description

Add tracers

Usage

```
add_tracers(config)
```

Arguments

`config`	config object as retrieved by `config_read`

Value

adds new columns 'conc_tracer.xxx' for four tracers ('CSO', 'inlet', 'rain_runoff' and 'WWTP') to flows_in_out for tracking sources withint the water cycle

`aggregate_flows_monthly` *Aggregate Flows Monthly*

Description

Aggregate Flows Monthly

Usage

```
aggregate_flows_monthly(flows, aggregation_function = median)
```

Arguments

`flows`	flows as retrieved by `calculate_flows_auto`
`aggregation_function`	function used for aggregation (default: median)

Value

tibble with monthly aggregated flows data

`aggregate_qualities_monthly` *Aggregate Qualities Monthly*

Description

Aggregate Qualities Monthly

Usage

```
aggregate_qualities_monthly(
  qualities,
  aggregation_function = median,
  minimum_tracer_sum = 0.999
)
```

Arguments

`qualities`	list as retrieved by `calculate_qualities`
`aggregation_function`	function used for aggregation (default: median)
`minimum_tracer_sum`	minimum tracer sum (default: 0.999 i.e. 99.9 percent) for filtering out results for with tracer has not reached almost 100 percent

Value

list with sublist "conc" with monthly aggregated concentrations data

---

`bfs_convert_equation` *Bank Filtration Share: convert equation*

Description

Bank Filtration Share: convert equation

Usage

```
bfs_convert_equation(
  df,
  col_equation_a = "equation_a",
  col_equation_b = "equation_b"
)
```

Arguments

df	data frame with bank filtration share equations, defined in sublist config$bfstypes_equations as retrieved by config_read
col_equation_a	column name of equation parameter a (default: "equation_a")
col_equation_b	column name of equation parameter b (default: "equation_b")

Value

adds "equation_function" to data frame

---

calc_conc *Calculate Concentration*

Description

Calculate Concentration

Usage

```
calc_conc(c_in, c_0, Q, V, k, t)
```

Arguments

c_in	c_in concentration of inflow
c_0	c_0 concentration in section at t = 0
Q	total inflow rate into section (m3/s)
V	volume of section (m3)
k	degradation parameter
t	time in seconds (s)

Value

concentration of substance at specific time

Examples

```
calc_conc(c_in = 10, c_0 = 0, Q = 40000, V = 300000, k = 0, t = 1)
```

`calculate_flow` *Calculate Flow*

Description

Calculate Flow

Usage

```
calculate_flow(
  df,
  input,
  shares_timeseries_wide = NULL,
  config,
  use_dynamic = FALSE,
  return_inputs = FALSE,
  debug = TRUE
)
```

Arguments

`df`	data frame with model as retrieved by `get_flowpath_table`
`input`	input flows as retrieved by `prepare_input` and sublist "flows"
`shares_timeseries_wide`	shares timeseries in wide formate, as retrieved by `prepare_input` and sublist "shares_timeseries" (default: NULL), only used if parameter "use_dynamic" is set to TRUE
`config`	list with config as imported with `config_read`
`use_dynamic`	for multiple outputs only: should static shares (as defined in column "section_out_share" of "outflows_multiple.csv") be used for separating the flow within a section or a function (as defined in column "section_out_function" of "outflows_multiple.csv")), (default: FALSE)
`return_inputs`	should also input data be returned in result dataset (default: FALSE)
`debug`	print debug messages (default: TRUE)

Value

tibble with Urban Water Model results

calculate_flow_share *Calculate flow share*

Description

Calculate flow share

Usage

```
calculate_flow_share(
  flow,
  shares_timeseries_id,
  shares_timeseries_wide,
  debug = FALSE
)
```

Arguments

flow	vector with flows (same length as)
shares_timeseries_id	id column in shares_timeseries_wide
shares_timeseries_wide	tibble or data frame with shares time series data in wide format
debug	print debug messages (default: FALSE)

Value

flow vector corrected with values from shares_timeseries data.frame for provided shares_timeseries_id

calculate_flow_stats *Calculate Flow Statistics Per Section*

Description

Computes discharge statistics from a daily or hourly flow dataset for each river section. The function supports both daily data (using a `date` column) and hourly data (using a `datetime` column), depending on the `"temporal_resolution"` attribute of the input.

Usage

```
calculate_flow_stats(flows)
```

Arguments

flows	A data frame containing a date or datetime column (depending on `attr(flows, "temporal_resolution")`) followed by one column per river section with discharge values.

Details

The returned statistics include:

- **MQ** Mean discharge
- **MNQ_years** Mean of yearly minimum values (yearly low flows)
- **MHQ_years** Mean of yearly maximum values (yearly high flows)
- **NNQ** Overall minimum discharge (absolute low flow)
- **HHQ** Overall maximum discharge (absolute high flow)

Additional tables contain statistics per year and per month, including monthly mean, monthly minimum, and monthly maximum discharges.

Value

A list with three elements:

per_section A tibble with discharge statistics per section (MQ, MNQ_years, MHQ_years, NNQ, HHQ).

per_year A tibble with statistics per section and year (MQ, MNQ_months, MHQ_months, NQ, HQ).

per_month A tibble with statistics per section and month (MQ, MNQ_same_months, MHQ_same_months, NQ, HQ).

`calculate_flowpath` *Calculate Flowpath*

Description

Calculate Flowpath

Usage

```
calculate_flowpath(flow_name, links, nodes, backward = FALSE)
```

Arguments

`flow_name`	flow name (as named in nodes) where calculation should start
`links`	links
`nodes`	nodes
`backward`	should flowpath be calculated backwards? (default: FALSE)

Value

data frame with column "order" indicating the flow path order

`calculate_flows` *Calculate Flows*

Description

Calculate Flows

Usage

```
calculate_flows(
  df_order,
  input,
  shares_timeseries_wide = NULL,
  config,
  use_dynamic = FALSE,
  return_inputs = FALSE,
  debug = TRUE
)
```

Arguments

`df_order`	list for each order id (corresponding to same level of distance of section from selected outflow)
`input`	input flows as retrieved by `prepare_input` and sublist "flows"
`shares_timeseries_wide`	shares timeseries in wide format, as retrieved by `prepare_input` and sublist "shares_timeseries" (default: NULL), only used if parameter "use_dynamic" is set to TRUE
`config`	list with config as imported with `config_read`
`use_dynamic`	for multiple outputs only: should static shares (as defined in column "section_out_share" of "outflows_multiple.csv") be used for separating the flow within a section or a function (as defined in column "section_out_function" of "outflows_multiple.csv")), (default: FALSE)
`return_inputs`	should also input data be returned in result dataset (default: FALSE)
`debug`	print debug messages (default: TRUE)

Value

returns modelled flows in tibble format

calculate_flows_auto *Calculate Flows Automatically*

Description

Calculate Flows Automatically

Usage

```
calculate_flows_auto(
  config,
  input_list,
  network,
  use_dynamic = FALSE,
  debug = FALSE
)
```

Arguments

config	list with config as imported with config_read
input_list	input_list as retrieved by prepare_input
network	tibble with water cycle flow network data, as retrieved by {prepare_network}
use_dynamic	for multiple outputs only: should static shares (as defined in column "section_out_share" of "outflows_multiple.csv") be used for separating the flow within a section or a function (as defined in column "section_out_function" of "outflows_multiple.csv")), (default: FALSE)
debug	print debug messages (default: FALSE)

Value

tibble with flows

calculate_outflows *Calculate Outflows*

Description

Calculate Outflows

Usage

```
calculate_outflows(
  list_order,
  input,
  config,
  use_dynamic = TRUE,
  return_inputs
)
```

Arguments

list_order	list for each order id (corresponding to same level of distance of section from selected outflow)
input	input as retrieved by prepare_input
config	list with config as imported with config_read
use_dynamic	for multiple outputs only: should static shares (as defined in column "section_out_share" of "outflows_multiple.csv") be used for separating the flow within a section or a function (as defined in column "section_out_function" of "outflows_multiple.csv")), (default: TRUE)
return_inputs	should also input data be returned in result dataset (default: FALSE)

Value

tibble with outflows added to provided dataset

---

calculate_qualities *Calculate Qualities*

Description

Calculate Qualities

Usage

```
calculate_qualities(
  input_list,
  flows,
  network,
  config,
  reverse_flow = FALSE,
  branchwise = TRUE,
  max_sections = NULL,
  debug = TRUE
)
```

Arguments

input_list	input_list as retrieved by prepare_input
flows	tibble with (modelled) flows for the network (e.g. precomputed upstream by the user)
network	tibble with water cycle flow network data, as retrieved by prepare_network
config	list with config as imported with config_read
reverse_flow	calculate reverse flow (default: FALSE)
branchwise	improved calculation workflow minimising unneeded section calculations (default: TRUE)
max_sections	restrict number of calculated sections in case problems occur. Provide a number <= number of sections to be calculated. If NULL, all sections will be calculated (default: NULL). Only used if reverse_flow = FALSE
debug	print debug messages (default: TRUE)

Value

returns modelled qualities in tibble format

---

calculate_qualities_backward

Calculate Qualities Backward

Description

Calculate Qualities Backward

Usage

```
calculate_qualities_backward(input_list, flows, network, config, debug = TRUE)
```

Arguments

input_list	input_list as retrieved by prepare_input
flows	flows as retrieved by xxxx
network	tibble with water cycle flow network data, as retrieved by {prepare_network}
config	list with config as imported with config_read
debug	print debug messages (default: TRUE)

Value

returns modelled flows in tibble format

`calculate_qualities_backward_branchwise`
Calculate Qualities Backward Branchwise

Description

Calculate Qualities Backward Branchwise

Usage

```
calculate_qualities_backward_branchwise(
  input_list,
  flows,
  network,
  config,
  debug = TRUE
)
```

Arguments

`input_list`	input_list as retrieved by `prepare_input`
`flows`	flows as retrieved by xxxx
`network`	tibble with water cycle flow network data, as retrieved by {`prepare_network`}
`config`	list with config as imported with `config_read`
`debug`	print debug messages (default: TRUE)

Value

returns modelled flows in tibble format

`calculate_qualities_forward`
Calculate Qualities Forward

Description

Calculate Qualities Forward

Usage

```
calculate_qualities_forward(
  input_list,
  flows,
  network,
  config,
  max_sections = NULL,
  debug = TRUE
)
```

Arguments

input_list	input_list as retrieved by prepare_input
flows	tibble with (modelled) flows for the network (e.g. precomputed upstream by the user)
network	tibble with water cycle flow network data, as retrieved by prepare_network
config	list with config as imported with config_read
max_sections	restrict number of calculated sections in case problems occur. Provide a number <= number of sections to be calculated. If NULL, all sections will be calculated (default: NULL). Only used if reverse_flow = FALSE
debug	print debug messages (default: TRUE)

Value

returns modelled flows in tibble format

---

calculate_quality *Calculate Quality in Section*

Description

Calculate Quality in Section

Usage

```
calculate_quality(
  df,
  input,
  shares_timeseries_wide = NULL,
  flows,
  quality = NULL,
  config,
  reverse_flow = FALSE,
  result_type = "list",
  debug = FALSE
)
```

Arguments

`df`	data frame with model as retrieved by `get_flowpath_table`
`input`	model input data flows as retrieved by `prepare_input` and sublist "flows"
`shares_timeseries_wide`	shares timeseries in wide format, as retrieved by `prepare_input` and sublist "shares_timeseries" (default: NULL), only used if parameter "use_dynamic" is set to TRUE
`flows`	flows as retrieved by `calculate_flows`
`quality`	list with sublist "conc" and "load" for all sections (default: NULL)
`config`	list with config as imported with `config_read`
`reverse_flow`	calculate reverse flow (default: FALSE)
`result_type`	define how the results should be returned. either a tibble with loads and concentrations in long format.(if result_type == "tibble"), a list with concentrations in wide format (result_type == "list") or a list with loads in wide format (result_type == "load")
`debug`	print debug messages (default: FALSE)

Value

tibble with Urban Water Model results with loads and concentrations (if result_type == "raw"), a list with concentrations (result_type == "conc" or "load") or a list with loads, default: "list"

`check_backflows_multiple` *Check backflows multiple*

Description

Check backflows multiple

Usage

```
check_backflows_multiple(config)
```

Arguments

`config`	list with config as imported with `config_read`

Value

nothing besides stops the code in case checks do not pass or message in case no config file configs/backflows_multiple.csv is provided

---

`check_if_function_and_calculate_flow`
Check if function and calculate flow

Description

Check if function and calculate flow

Usage

```
check_if_function_and_calculate_flow(df, q)
```

Arguments

`df`	data frame with one row containing column 'section_out_function_parsed' with function for calculating flow
`q`	flow vector

Value

if function: calculated flow vector, if not: NA_real

`check_network_errors` *Check Network Errors*

Description

Check Network Errors

Usage

```
check_network_errors(network)
```

Arguments

`network`	tibble with water cycle flow network network, as retrieved by {prepare_network}

Value

list with information on network errors

`check_outflow_multiple_dynamic_functions` *Check outflow multiple dynamic functions*

Description

Check outflow multiple dynamic functions

Usage

```
check_outflow_multiple_dynamic_functions(
  config,
  q = 1,
  allowed_relative_offset_percent = 0.001
)
```

Arguments

`config`	list with config as imported with `config_read`
`q`	total section flow used for testing, needs to be a scalar! (default: 1)
`allowed_relative_offset_percent`	maximum allowed percental offset for sum of all section outflows compared to section inflow (i.e. parameter q), default: 0.0001)

Value

nothingt if check passes for all outflow_id s with defined functions, otherwise error

---

`check_shares_timeseries` *Check Shares of Multiple Outflow Time Series*

Description

Check Shares of Multiple Outflow Time Series

Usage

```
check_shares_timeseries(shares_timeseries, config, debug = TRUE)
```

Arguments

`shares_timeseries`	shares timeseries dataset (default: kwb.BerlinWaterModel.public::shares_timeseries)
`config`	model network configuration (as retrieved by `config_read`)
`debug`	print debug messages (default: TRUE)

Value

prints debug messages or stop in case of errors

Examples

```
config <- kwb.BerlinWaterModel.public::config_read()
check_shares_timeseries(
shares_timeseries = kwb.BerlinWaterModel.public::shares_timeseries,
config = config
)
```

---

combine_and_clean_dfs *Combine and clean dataframe*

Description

Combine and clean dataframe

Usage

```
combine_and_clean_dfs(list)
```

Arguments

list	list miwth Urban Water Model results

Value

tibble with Urban Water Model results

---

compute_gallery_kpis *Compute abstraction metrics per gallery*

Description

Computes yearly, monthly and aggregated abstraction metrics for a single gallery (Brunnengalerie).

Usage

```
compute_gallery_kpis(
  df,
  gallery_col = "gallery",
  date_col = "date",
  q_col = "cbm_per_day",
  temporal_resolution = c("auto", "daily", "monthly"),
  date_min = NULL,
  date_max = NULL
)
```

Arguments

`df`	A `data.frame` containing at least gallery ID, date and abstraction rates (e.g. in m^3/d) for a single gallery.
`gallery_col`	Name of the gallery column (default: `"gallery"`).
`date_col`	Name of the date column (default: `"date"`). The column must be coercible to `Date`.
`q_col`	Name of the abstraction column (e.g. `"cbm_per_day"`, interpreted as m^3/d).
`temporal_resolution`	Character string specifying the temporal resolution, one of `"auto"`, `"daily"` or `"monthly"`. If set to `"auto"`, the resolution is inferred from the median time step between consecutive dates.
`date_min`	Optional start date of the observation period. If `NULL` (default), the minimum of the date column is used.
`date_max`	Optional end date of the observation period. If `NULL` (default), the maximum of the date column is used.

Details

The function automatically detects whether daily values were synthetically generated from monthly data, e.g. by completing a daily grid and applying `tidyr::fill(direction = "up")`. In that case, all volume-based yearly and monthly metrics are computed, but daily extreme metrics (`Q1Ist`, `MQ1Ist`, `Q30`, `MQ30`) are suppressed and returned as `NA`, because they would not represent real daily extremes.

Months with `NA` or zero abstraction are excluded from the calculation of monthly-based KPIs (e.g. NQMonat, HQMonat, MQMonat). Additional size indicators are provided: `n_months_with_data`, `n_months_in_operation` and `n_months_observation_period` per gallery, as well as two percentage indicators describing the share of operational months within the observed data and within the full observation period.

Value

A named `list` with the following elements:

per_year	A tibble of yearly metrics per gallery and year, including yearly volumes (e.g. QA, NQMonat, HQMonat, Q365, Q1Ist, Q30) as well as the number of months with data (n_months_with_data), the number of months in operation (n_months_in_operation), the length of the observation period in months (n_months_observation_period) and the corresponding percentages (n_months_operation_with_data_percent, n_months_operation_within_observati
per_gallery	A tibble of aggregated metrics per gallery over the full observation period (e.g. MQA, NQA, HQA, MNQA, NNQA, MHQA, HHQA, MQ365, HQ365, MQ1Ist, MQ30), together with n_months_with_data, n_months_in_operation, n_months_observation_period, n_months_operation_with_data_percent and n_months_operation_within_observationper
per_month	A tibble of monthly metrics per gallery and calendar month aggregated over all years (e.g. MQMonat, NQMonat, HQMonat), plus the number of months with data (n_months_with_data), the number of months in operation (n_months_in_operation), the observation-period length in months (n_months_observation_period) and the corresponding percentages (n_months_operation_wit n_months_operation_within_observationperiod_percent).
meta	A named list with meta information such as temporal_resolution, synthetic_daily_from_mon has_daily_extremes, the vector requires_real_daily_for = c("Q1Ist", "MQ1Ist", "Q30", "MQ30"), and the effective date_min, date_max and n_months_observation_period.

`config_read` *Configuration: Read*

Description

Configuration: Read

Usage

```
config_read(
  config_dir = system.file("extdata/config/network_complete_mean-start-conc", package =
    "kwb.BerlinWaterModel.public"),
  file_encoding = "UTF-8"
)
```

Arguments

config_dir	directory with configuration files (default: system.file("extdata/config/network_complete_mean-start-conc", package = "kwb.BerlinWaterModel")). It is mandatory that there are three files within this folder: "flows_in_out.csv", "outflows_multiple.csv" and "sections.csv"
file_encoding	encoding for reading the files (default: "UTF-8")

Value

list with three sublists "flows_in_out", "outflows_multiple" and "sections

Examples

```
## Not run: config <- kwb.BerlinWaterModel.public::config_read()
config
## End(Not run)
```

`convert_concentration_units`
Convert concentration units to kg/m3

Description

Convert concentration units to kg/m3

Usage

```
convert_concentration_units(df, return_inputs = FALSE)
```

Arguments

`df`	data frame with "conc_.<g|mg|mg|ng>."
`return_inputs`	should input data frame also be returned (default: FALSE) or only newly calculated columns?

Value

tibble with "conc_.kg.m3"

Examples

```
df <- data.frame(
conc_As.ng.L = c(10, 50, 100),  # Nanogramm pro Liter
conc_Zn.ug.L = c(500, 1000, 2000),  # Mikrogramm pro Liter
conc_Fe.mg.L = c(0.1, 0.2, 0.3),    # Milligramm pro Liter
conc_Cu.g.L = c(0.005, 0.01, 0.02)  # Gramm pro Liter
)

convert_concentration_units(df, return_inputs = TRUE)
```

`cso` *Dataset: Combined Sewer Overflows (cso) for one rainfall event simulated with Infoworks in 2019*

Description

Dataset with combined sewer overflow events for one rainfall rainfall event simulated with Infoworks in 2019

Usage

```
cso
```

Format

A tibble with 12,096 rows and five columns

file_name filename of data origin

datetime datetime

seconds seconds

CSO_id CSO_id

cbm_per_second flow (cbm per second)

`evapo_p` *Dataset: Daily Potential Evaporation from DWD*

Description

Daily potential evaporation based on DWD 1x1 km raster data.

Usage

```
evapo_p
```

Format

A tibble with 365 rows and 10 columns:

file Name of the file from the DWD https/ftp server

date Date

year Year

month Month

day Day

mean Mean (mm/d) for n raster cells (defined in n_values)

sd Standard deviation (mm/d) for n raster cells (defined in n_values)

min Minimum (mm/d) for n raster cells (defined in n_values)

max Maximum (mm/d) for n raster cells (defined in n_values)

n_values Number of selected raster cells

Source

https://opendata.dwd.de/climate_environment/CDC/grids_germany/daily/evapo_p/DESCRIPTION_gridsgermany_daily_evapo_p_en.pdf https://opendata.dwd.de/climate_environment/CDC/grids_germany/daily/evapo_p/

`fill_month_to_start` *Expand monthly end-of-month values to all days or hours of the same month*

Description

For each row in `df`, this function assumes that the time column contains a monthly value at (or near) the end of a month (e.g. "2002-01-31" or "2002-01-31 23:00:00") and expands it backwards to all days or all hours from the 1st of that month up to the given timestamp.

Usage

```
fill_month_to_start(
  df,
  date_col = "date",
  datetime_col = "datetime",
  temporal_res = c("auto", "days", "hours"),
  date_min = NULL,
  date_max = NULL
)
```

Arguments

`df`	A data frame or tibble.
`date_col`	Name of the daily date column (default: "date").
`datetime_col`	Name of the hourly datetime column (default: "datetime").
`temporal_res`	One of "auto", "days", "hours". If "auto", the presence of `date_col` / `datetime_col` decides.
`date_min`	Optional lower bound of the output time window. For daily mode this is interpreted as `Date` (or converted via `as.Date()`); for hourly mode it can be a `POSIXct` or `Date`. If `NULL`, no lower cropping is applied.
`date_max`	Optional upper bound of the output time window. Same interpretation as `date_min`. If `NULL`, no upper cropping is applied.

Details

If `temporal_res = "auto"` (default), the function chooses daily expansion when a column `date` is present and hourly expansion when a column `datetime` is present.

The resulting series can optionally be cropped to a user-defined time window via `date_min` and `date_max`.

Value

A tibble where each original row is expanded so that all days (for "days") or all hours (for "hours") from the 1st of the respective month up to the original timestamp are present. All other columns are copied to the created rows. The result is then cropped to `[date_min, date_max]` if those are supplied. The name of the time column is preserved ("date" or "datetime").

`fill_timeseries` *Fill Timeseries based on User Defined Intervall (days, hours, minutes or seconds)*

Description

Fill Timeseries based on User Defined Intervall (days, hours, minutes or seconds)

Usage

```
fill_timeseries(
  df,
  col_datetime = "date",
  temporal_resolution = "days",
  direction = "up"
)
```

Arguments

`df`	data frame with data
`col_datetime`	column of date or datetime
`temporal_resolution`	select one of ("days", "hours", "minutes", "seconds"), (default: "days")
`direction`	in which direction should the parameter be filled ("up" or "down"), default: "up"

Value

tibble with filled dates. Note that if min(daste) == "xxxx-xx-end-of-month-day" the min(date) is set to "xxxx-xx-01", as monthly measured (originally summed) values were set to the last day of the month, but are valid for the whole month

find_node_orders *Helper function: find node order*

Description

Helper function: find node order

Usage

```
find_node_orders(links, nodes, start_node, reverse = FALSE, mode = "out")
```

Arguments

links	links
nodes	nodes
start_node	start node
reverse	should network direction be reversed? (default: FALSE)
mode	Character constant, gives whether the shortest paths to or from the given vertices should be calculated for directed graphs. If out then the shortest paths from the vertex, if in then to it will be considered. If all, the default, then the graph is treated as undirected, i.e. edge directions are not taken into account. This argument is ignored for undirected graphs (default: "out")

Value

data frame with additional column "order"

get_bfshares *Get Bank Filtration shares Model input*

Description

Get Bank Filtration shares Model input

Usage

```
get_bfshares(
  config,
  ww = kwb.BerlinWaterModel.public::ww,
  temporal_resolution = "days",
  bfshare_dynamic = TRUE
)
```

Arguments

config	model network configuration (as retrieved by config_read)
ww	waterworks dataset (default: kwb.BerlinWaterModel.public::ww)
temporal_resolution	specify temporal resolution of model input dataset. (default: "days"). Valid options are: "days" or "hours"
bfshare_dynamic	should dynamic bankfiltration shares be used or the static ones contained in column "bank_filtration_share" of config$flows_in_out

Value

data framer with flow ids, date/datetime and well gallery metadata, column "bank_filtration_share" is set depending on bfshare_dynamic == TRUE (based on column "bank_filtration_share_dynamic") or FALSE (bank_filtration_share_static)

---

get_flowpath_table *Get Flowpath Table*

Description

Get Flowpath Table

Usage

```
get_flowpath_table(outflow_id, network, config)
```

Arguments

outflow_id	id of outflow section
network	tibble with water cycle flow network data, as retrieved by {prepare_network}
config	config

Value

list for each order id containing tibble with all sections of same order id

get_gallery_kpis_meta

Extract meta information from a galleries lookup table

Description

Helper to flatten the `meta` entries from the `kpis` list-column in `ww_galleries_lookup` (or a similar lookup table). Each gallery gets one row with all meta fields as columns.

Usage

```
get_gallery_kpis_meta(lookup)
```

Arguments

`lookup`	A tibble with columns `waterworks`, `gallery` and a list-column `kpis` as returned by `compute_gallery_kpis()`.

Value

A tibble with one row per gallery and all meta information expanded into regular columns (e.g. `temporal_resolution`, `synthetic_daily_from_month`, `has_daily_extremes`, ...), including `waterworks` and `gallery`.

get_gallery_kpis_per_gallery

Extract aggregated KPIs per gallery from a lookup table

Description

Helper to flatten the nested `kpis$per_gallery` entries from `ww_galleries_lookup` (or a similar lookup table) into a single tibble.

Usage

```
get_gallery_kpis_per_gallery(lookup)
```

Arguments

`lookup`	A tibble with columns `waterworks`, `gallery` and a list-column `kpis` as returned by `compute_gallery_kpis()`.

Details

This returns one row per gallery with all aggregated KPIs that were computed over the full observation period (e.g. `MQA`, `NQA`, `HQA`, `MNQA`, `NNQA`, `MHQA`, `HHQA`, `MQ365`, `HQ365`, `MQ1Ist`, `MQ30`, ...).

Any `waterworks` or `gallery` columns inside `per_gallery` are dropped to avoid name clashes; the outer ID columns are kept.

Value

A tibble with one row per gallery, containing all aggregated KPIs from `kpis$per_gallery` and the identifying columns `waterworks` and `gallery`.

`get_gallery_kpis_per_month`

Extract monthly KPIs from a galleries lookup table

Description

Helper to flatten the nested `kpis$per_month` entries from `ww_galleries_lookup` (or a similar lookup table) into a single tibble.

Usage

```
get_gallery_kpis_per_month(lookup)
```

Arguments

`lookup`	A tibble with columns `waterworks`, `gallery` and a list-column `kpis` as returned by `compute_gallery_kpis()`.

Value

A tibble with monthly KPIs (gallery × calendar month), including the columns `waterworks` and `gallery`.

`get_gallery_kpis_per_year`

Extract yearly KPIs from a galleries lookup table

Description

Helper to flatten the nested `kpis$per_year` entries from `ww_galleries_lookup` (or a similar lookup table) into a single tibble.

Usage

```
get_gallery_kpis_per_year(lookup)
```

Arguments

`lookup`	A tibble with columns `waterworks`, `gallery` and a list-column `kpis` as returned by `compute_gallery_kpis()`.

Value

A tibble with yearly KPIs, one row per gallery and year, including the columns `waterworks` and `gallery`.

`get_ids_from_names` *Helper function: get ids from names*

Description

Helper function: get ids from names

Usage

```
get_ids_from_names(vector, config)
```

Arguments

`vector`	vector with names
`config`	config as retrieved by `config_read`

Value

tibble with section names and ids contained in configuration

`get_link_names` *Helper function: get link names*

Description

Helper function: get link names

Usage

```
get_link_names(network)
```

Arguments

`network`	tibble with water cycle flow network data, as retrieved by {`prepare_network`}

Value

tibble with link names and columns source and target

`get_names_from_ids` *Helper function: get names from ids*

Description

Helper function: get names from ids

Usage

```
get_names_from_ids(vector, config)
```

Arguments

`vector`	vector with ids
`config`	config as retrieved by `config_read`

Value

tibble with section names and ids contained in configuration

`get_nodes` *Helper function: get nodes*

Description

Helper function: get nodes

Usage

```
get_nodes(network, config)
```

Arguments

`network`	tibble with water cycle flow network data, as retrieved by {`prepare_network`}
`config`	config as retrieved by `config_read`

Value

tibble with columns name (from_name) and group (source_group)

get_reverse_flows_per_section *Get Reverse Flows Per Section*

Description

Get Reverse Flows Per Section

Usage

```
get_reverse_flows_per_section(flows)
```

Arguments

flows	flows

Value

tibble with sections ordered decreasing by number of data points below zero and date_min (first neg. flow) and date_max (last neg. flow) within dataset

get_section_idnames *Helper function: get section idnames*

Description

Helper function: get section idnames

Usage

```
get_section_idnames(config)
```

Arguments

config	config as retrieved by config_read

Value

tibble with section names and ids contained in configuration

`inflows` *Dataset: Inflows*

Description

Dataset with inflows

Usage

```
inflows
```

Format

A tibble with 21,170 rows and three columns

date date

id id of inflow station

cbm_per_second daily average flow in cubicmeter per second

`label_signif_clean` *Format numeric values with significant digits and minimal trailing zeros*

Description

This function formats numeric values to a specified number of significant digits, avoiding scientific notation and unnecessary trailing zeros. Useful for plotting labels where readability is important across varying numeric magnitudes.

Usage

```
label_signif_clean(x, digits = 2)
```

Arguments

`x`	Numeric vector of values to be formatted.
`digits`	Integer. Number of significant digits to retain (default is 2).

Value

A character vector of formatted numbers.

Examples

```
label_signif_clean(c(0.0001234, 0.0456, 1.23, 12.3, 123.4))
```

log10_minor_breaks *Generate minor tick marks for a log10 axis*

Description

Returns a function that computes minor tick positions for a log10-scaled axis. This is useful in conjunction with `scale_y_log10()` or `scale_x_log10()` in `ggplot2`, where minor breaks are not added by default.

Usage

```
log10_minor_breaks(minor_base = 1:9)
```

Arguments

`minor_base`	A numeric vector of factors (default is 1:9) to be multiplied by each power of 10. This defines which intermediate ticks (e.g., 2, 3, ..., 9) are included between each decade (e.g., between 0.1 and 1).

Value

A function that takes a numeric range and returns a numeric vector of minor tick positions.

Examples

```
## Not run:
ggplot2::scale_y_log10(
  limits = c(0.01, 100),
  breaks = c(0.01, 0.1, 1, 10, 100),
  minor_breaks = kwb.BerlinWaterModel.public::log10_minor_breaks()
)

## End(Not run)
```

merge_two_level_lists *Merge nested two-level lists while preserving structure*

Description

This function merges two nested lists of the form `list(top_id = list(sub_id = tibble))`, as used in the BerlinWaterModel package. The structure (top-level and sub-level keys) is preserved, and missing elements are added. If the same element exists in both lists, the preferred side can be chosen.

Usage

```
merge_two_level_lists(x, y, prefer = c("left", "right"))
```

Arguments

x	A nested list with structure `list(top_id = list(sub_id = tibble))`.
y	A nested list with the same structure as `x`.
prefer	Character string, either `"left"` (default) or `"right"`. If `"left"`, values from `x` are kept in case of duplicates. If `"right"`, values from `y` overwrite those in `x`.

Value

A nested list with the same two-level structure containing the union of all elements from `x` and `y`.

Examples

```
library(tibble)

a <- list("S03" = list("S21" = tibble(a = 1, b = 2)))
b <- list("S03" = list("S07" = tibble(a = 3, b = 4)))
c <- list("S01" = list("S02" = tibble(a = 5, b = 6)))

# Default: prefer = "left"
merge_two_level_lists(a, b)

# Right side overwrites in case of duplicates
merge_two_level_lists(a, b, prefer = "right")

# Merge more than two lists using purrr::reduce
purrr::reduce(list(a, b, c), merge_two_level_lists)
```

---

parse_a_and_b *Helper function: parse a and*

Description

Helper function: parse a and

Usage

```
parse_a_and_b(a, b)
```

Arguments

a	parameter a
b	parameter b

Value

R function of parsed equation

Examples

```
a <- 1
b <- 2
parse_a_and_b(a, b)
```

`parse_equation` *Helper function: parse equation*

Description

Helper function: parse equation

Usage

```
parse_equation(equation)
```

Arguments

`equation`	equation in text form (either "constant" -> y = c or "ln" -> y = a * ln(x) + b)

Value

R function of parsed equation

`plot_flows_and_bfshares_per_section`
Plot Flows and BF & MAR shares per section

Description

Plot Flows and BF & MAR shares per section

Usage

```
plot_flows_and_bfshares_per_section(
  config,
  flows,
  network,
  ww,
  scale_factor = 1,
  debug = TRUE,
  add_caption_simulation = FALSE,
  add_caption_weblink = FALSE,
  add_caption_outflowsmultiple = FALSE
)
```

Arguments

config	list with config as imported with config_read
flows	flows as retrieved by xxxx
network	tibble with water cycle flow network data, as retrieved by {prepare_network}
ww	waterworks dataset (default: kwb.BerlinWaterModel.public::ww)
scale_factor	scale factor for increasing size of labels
debug	print debug messages (default: TRUE)
add_caption_simulation	add caption with simulation information (default: FALSE)
add_caption_weblink	Add one line in caption to reference webpage of vignette (default: FALSE)
add_caption_outflowsmultiple	Add information of model configuration for multiple outflows (default: FALSE)

Value

plots flows and BF & MAR shares for each section

---

plot_network_complex *Plot Network: complex*

Description

Plot Network: complex

Usage

```
plot_network_complex(network, config, show_labels = FALSE)
```

Arguments

network	tibble with water cycle flow network data, as retrieved by {prepare_network}
config	list with config as imported with config_read
show_labels	show labels (default: FALSE)

Value

complex network graph

plot_network_simple *Plot Network: simple*

Description

Plot Network: simple

Usage

```
plot_network_simple(network)
```

Arguments

network	tibble with water cycle flow network data, as retrieved by {prepare_network}

Value

simple network graph

prepare_input *Prepare Model input*

Description

Prepare Model input

Usage

```
prepare_input(
  temporal_resolution = "days",
  config = config_read(),
  cso = kwb.BerlinWaterModel.public::cso,
  inflows = kwb.BerlinWaterModel.public::inflows,
  rain = kwb.BerlinWaterModel.public::rain,
  evapo_p = kwb.BerlinWaterModel.public::evapo_p,
  shares_timeseries = kwb.BerlinWaterModel.public::shares_timeseries,
  ww = kwb.BerlinWaterModel.public::ww,
  wwtp = kwb.BerlinWaterModel.public::wwtp,
  bfshare_dynamic = FALSE,
  share_wwtp_sch_to_nordgraben_timeseries = TRUE,
  share_wwtp_sch_panke_1 = 0.1,
  date_separation_panke_1_2 = "2015-04-15",
  share_wwtp_sch_panke_2 = 0.9,
  col_wwtp_sch = "Q_KW_SCH",
  col_wwtp_sch_nordgraben = "Q_KW_SCH_Nordgraben",
  col_wwtp_sch_panke = "Q_KW_SCH_Panke",
```

```
  col_panke_baseflow_no_Q_wwtp = "Panke_baseflow_no_Q_KW_SCH_Panke",
  date_min = "2002-01-01",
  date_max = "2022-12-31",
  debug = TRUE
)
```

Arguments

temporal_resolution	specify temporal resolution of model input dataset. (default: "days"). Valid options are: "days" or "hours"
config	model network configuration (as retrieved by config_read)
cso	cso dataset (default: kwb.BerlinWaterModel.public::cso)
inflows	inflows dataset (default: kwb.BerlinWaterModel.public::inflows)
rain	rain dataset (default: kwb.BerlinWaterModel.public::rain)
evapo_p	evaporation dataset (default: kwb.BerlinWaterModel.public::evapo_p)
shares_timeseries	shares timeseries dataset (default: kwb.BerlinWaterModel.public::shares_timeseries)
ww	waterworks dataset (default: kwb.BerlinWaterModel.public::ww)
wwtp	wastewater treatment plant dataset (default: kwb.BerlinWaterModel.public::wwtp)
bfshare_dynamic	should dynamic bankfiltration shares be used or the static ones contained in column "bank_filtration_share" of config$flows_in_out
share_wwtp_sch_to_nordgraben_timeseries	if TRUE time series to "Nordgraben" is used (column name defined in parameter "col_wwtp_sch_nordgraben") and remaining water is transfered to "Panke" (column name defined in parameter "col_wwtp_sch_panke")
share_wwtp_sch_panke_1	share of WWTP Schoenerlinde to Panke before (default: 0.1) date_separation_panke_1_2; only used if share_wwtp_sch_to_nordgraben_timeseries == FALSE
date_separation_panke_1_2	date to separate shares before (i.e. share_wwtp_sch_panke_1) and after (share_wwtp_sch_panke_2) given date (default: "2025-04-15") (default: 1 - share_wwtp_sch_panke); only used if share_wwtp_sch_to_nordgraben_timeseries == FALSE
share_wwtp_sch_panke_2	share of WWTP Schoenerlinde to Panke before (default: 0.9) date_separation_panke_1_2; only used if share_wwtp_sch_to_nordgraben_timeseries == FALSE
col_wwtp_sch	column name of WWTP Schoenerlinde (default: "Q_KW_SCH")
col_wwtp_sch_nordgraben	column name of WWTP Schoenerlinde outflow to Nordgraben (default: "Q_KW_SCH_Nordgraben")
col_wwtp_sch_panke	column name of WWTP Schoenerlinde outflow to Panke (default: "Q_KW_SCH_Panke")

col_panke_baseflow_no_Q_wwtp
: (default: "Panke_baseflow_no_Q_KW_SCH_Panke")

date_min
: minimum date, i.e. start of simulation (default: "2002-01-01)

date_max
: maximum date, i.e. end of simulation (default: "2022-12-31)

debug
: print debug messages (default: TRUE)

Value

input dataset, will be filled in case temporal resolution in increased

---

prepare_network *Prepare Network Data*

Description

Prepare Network Data

Usage

```
prepare_network(config)
```

Arguments

config
: config as retrieved by config_read

Value

data structure for plot_network functions and for further calculations

---

prepare_qsimVis_input *Preparew QsimVis Input*

Description

Preparew QsimVis Input

Usage

```
prepare_qsimVis_input(config, flows, qualities, rounding_digits = 3)
```

Arguments

config
: config list as retrieved by config_read

flows
: flows input data frame in "wide" format

qualities
: qualities list input data structure for each section in wide format

rounding_digits
: digits used for result data rounding (default: 3)

Value

data frame with flow data in long format joined with config$qsimVis data frame

---

`rain` *Dataset: Rain (rain) with DWD rainfall data*

Description

Dataset with rainfall data (based on DWD station 0433)

Usage

```
rain
```

Format

A tibble with 8,721 rows and two columns

datetime datetime

DWD_0433 hourly rainfall in mm for DWD rainstation 0433

`set_tracer_starting_conc`
Set Tracer Starting Concentrations To Final Modelling Results

---

Description

Set Tracer Starting Concentrations To Final Modelling Results

Usage

```
set_tracer_starting_conc(config, qualities)
```

Arguments

`config`	config list structure as retrieved by `config_read`
`qualities`	qualities list result structure

Value

config list with starting concentrations (defined in config$sections) based on qualities. The last time step is used as starting concentration

`set_tracer_starting_conc_stats` *Set Tracer Starting Concentrations Statistics*

Description

Set Tracer Starting Concentrations Statistics

Usage

```
set_tracer_starting_conc_stats(
  config,
  qualities,
  aggregation_function = median,
  minimum_tracer_sum = 0.999
)
```

Arguments

`config`	config list structure as retrieved by `config_read`
`qualities`	qualities list result structure
`aggregation_function`	function used for data aggregation (default: median)
`minimum_tracer_sum`	minimum tracer sum (default: 0.999 i.e. 99.9 percent) for filtering out results for with tracer has not reached almost 100 percent

Value

config list with starting concentrations (defined in config$sections) based on qualities

---

`shares_timeseries` *Dataset: Shares (0 - 1) of discharge to a specific downstream river section*

Description

Dataset: Shares (0 - 1) of discharge to a specific downstream river section

Usage

```
shares_timeseries
```

Format

A tibble with 15,340 rows and three columns

date date

id id of shares timeseries

share share (0-1) to a specific downstream river section

---

shorten_ww_flow_id *Helper: shorten WW flow id*

Description

Helper: shorten WW flow id

Usage

```
shorten_ww_flow_id(df, col_flow_id = "flow_id")
```

Arguments

df	df with WW flow data andf flow ids as columns
col_flow_id	column name with flow id (default: "flow_id")

Value

vector with shortened WW flow ids

---

ww *Dataset: Waterworks (WW) Abstractions per Gallery*

Description

Dataset with waterworks abstractions per gallery

Usage

```
ww
```

Format

A tibble with 828 rows and three columns

date date

id id of WW gallery

cbm_per_second measured flow in cubicmeter per second, backcalculated from monthly abstraction sums by dividing through number of days in month

`wwtp` *Dataset: WWTP (Wastewater Treatment Plant) Flows*

Description

Dataset with WWTP inflows

Usage

```
wwtp
```

Format

A tibble with 4015 rows and three columns

date date

id id of WWTP

cbm_per_second measured flow in cubicmeter per second, backcalculated from monthly effluent sums by dividing through number of days in month

Index