Package: kwb.raindrop (via r-universe)

June 14, 2026

Title R Package for Optimisation Simulations for Rainwater Management Simulations Performed with Calculation Engine Provided by Tandler

Version 0.0.0.9000

Description R Package for Optimisation Simulations for Rainwater Management Simulations Performed With Calculation Engine Provided by Tandler.

License MIT + file LICENSE

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

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

Imports dplyr, forcats, fs, future, future.apply, ggplot2, hdf5r, kwb.event, kwb.utils, lubridate, magrittr, parallel, progressr, purrr, rlang, stringr, tibble, tidyr

Suggests covr, DT, htmlwidgets, knitr, plotly, readr, rmarkdown, writexl

VignetteBuilder knitr

Remotes github::kwb-r/kwb.event, github::kwb-r/kwb.utils

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/pak/sysreqs cmake make libhdf5-dev libicu-dev libuv1-dev

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

Date/Publication 2026-06-03 12:05:31 UTC

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

RemoteRef HEAD

RemoteSha 0644b0ff9ac4e3f845d959272d3b41390197903a

Contents

`add_overflow_events_and_waterbalance`

Add overflow-event metrics and water-balance shares (percent) to simulation results

---

Description

Computes (i) overflow event statistics from the element outflow time series and (ii) water-balance components as percent shares for both `element` and `connected_area` per scenario.

Usage

```
add_overflow_events_and_waterbalance(
  simulation_results,
  event_separation_hours = 4,
  canonical_variables = NULL
)
```

Arguments

`simulation_results`
: Named list of scenario results. Each entry is expected to contain:

  - `element$water_balance` with columns `variable`, `value`
  - `connected_area$water_balance` with columns `variable`, `value`
  - `element$rates` with columns `time`, `variable`, `value`

`event_separation_hours`
: Numeric. Minimum time between two overflow events (in hours). Defaults to `4`.

`canonical_variables`
: Optional `character()` vector of water-balance variable names (without the `element.` / `connectedarea.` prefix and without the trailing `_`), e.g. `c("WB_Regen", "WB_Evapotranspiration", "WB_InfiltrationNetto", "WB_Oberflaechenablauf_Ueberlauf", "WB_Oberflaechenablauf_Ver`. When **every** scenario in `simulation_results` is `NULL` (or otherwise provides no water-balance data), the function would normally return only the four headline columns. Pass `canonical_variables` to attach `element.<var>_` and `connectedarea.<var>_` `NA`-filled stub columns to such rows so the rendered datatable still exposes the expected column structure. Defaults to `NULL` (no canonical fallback).

Details

Water-balance percentages are computed with sign preserved (`value / denom`):

- **element** denominator: `WB_Regen + abs(WB_Oberflaechenablauf_Verschaltungen)`
- **connected_area** denominator: `WB_Regen` (fallback to `abs(WB_Oberflaechenablauf_Verschaltungen)` if `WB_Regen` is `NA` or `0`)

Overflow events are derived from positive `Oberflaechenablauf_Ueberlauf` values using `kwb.event::hsEvents()`. Assuming the overflow rate is in `mm/h`, event sums (in `mm`) are obtained by integrating each sample over its **local** time step (`time[i+1] - time[i]`; the last sample inherits the previous step). A warning is emitted if the time step is non-uniform, and `sum_overflows` is returned as `NA` if it cannot be determined (single sample).

Missing components are tolerated: scenarios whose entry in `simulation_results` is `NULL` (or which lack any of `element`, `element$water_balance`, `connected_area`, `connected_area$water_balance` or `element$rates`) still produce a row of the output tibble. The four "headline" columns (`s_name`, `n_overflows`, `median_duration_overflows_hours`, `sum_overflows`) are always present and filled with `NA` where they cannot be computed.

For the `element.*_` and `connectedarea.*_` water-balance columns: if one side is missing in a given scenario but the other side has data, a stub of `NA`-filled columns is fabricated for the missing side by mirroring the populated side's variable names. This preserves the table's column structure when, for example, every run disables roof ET (so the engine skips writing Dach.h5 and every scenario has `connected_area = NULL`): the `connectedarea.*_` columns are kept and filled with `NA`, instead of being dropped from the output entirely. The mirror is a best-effort hint for the user, not a guarantee that the names match what a populated `connected_area` would have produced.

Value

A tibble with one row per scenario containing:

- s_name
- n_overflows
- median_duration_overflows_hours
- sum_overflows
- water-balance percentage columns for element.*_ and connectedarea.*_

Examples

```
## Not run:
out <- add_overflow_events_and_waterbalance(simulation_results)
out <- add_overflow_events_and_waterbalance(
  simulation_results,
  event_separation_hours = 6
)

## End(Not run)
```

---

compute_costs *Compute construction costs for an infiltration-swale parameter grid*

Description

Given a parameter grid that drives the simulation, attach a per-scenario breakdown of construction costs plus a cost_total column suitable for filtering / sorting in the results datatable.

Usage

```
compute_costs(
  param_grid,
  storage_type = c("infiltration_box", "gravel_trench"),
  cost_rates = default_cost_rates()
)
```

Arguments

param_grid	data.frame / tibble — must contain the four geometry columns listed above. Other columns are passed through unchanged.
storage_type	character(1) — "infiltration_box" (default; ~95% porosity, the Austrian "Sickerbox") or "gravel_trench" (~30% porosity, the Austrian "Schotterrigol"). Only used when param_grid does not have a storage_type column.
cost_rates	list — unit costs as returned by default_cost_rates(). Override individual entries to run cost sensitivity analyses.

Details

Required columns in `param_grid`:

- `mulde_area` — element footprint area in m²
- `mulde_height` — surface depression depth in **mm**
- `filter_height` — soil-filter layer thickness in **mm**
- `storage_height` — storage layer thickness in **mm**

Optional column `storage_type` (character: `"infiltration_box"` or `"gravel_trench"`) selects the per-scenario storage rate. If absent, the value of the `storage_type` argument is used for every row.

Cost formulas (Leimgruber, 2026-03-27):

```
cost_excavation = mulde_area * (mulde_h + filter_h + storage_h)/1000 * 70
cost_profiling  = mulde_area * 10
cost_filter     = mulde_area * filter_h/1000 * 200
cost_storage    = mulde_area * storage_h/1000 * <350 | 50>
cost_total      = sum of the four above
```

Value

The input `param_grid` with the columns `storage_type`, `cost_excavation`, `cost_profiling`, `cost_filter`, `cost_storage`, `cost_total` (all in EUR) appended.

Examples

```
grid <- tibble::tibble(
  scenario_name = c("s00001", "s00002"),
  mulde_area = c(50, 50),
  mulde_height = c(300, 300),
  filter_height = c(300, 300),
  storage_height = c(600, 600)
)

# Infiltration box (default)
compute_costs(grid)

# Gravel trench
compute_costs(grid, storage_type = "gravel_trench")

# Per-scenario storage type
compute_costs(dplyr::mutate(grid,
                            storage_type = c("infiltration_box",
                                             "gravel_trench")))
```

default_canonical_wb_variables

Canonical water-balance variable names emitted by the Tandler engine

Description

Returns the list of water-balance scalar names the Tandler "Regenwasserbewirtschaftung" engine writes into the `Wasserbilanz` group of each result HDF5 (without the `element.` / `connectedarea.` prefix and without the trailing _ that `add_overflow_events_and_waterbalance()` appends when building the output columns). Pass this to that function's `canonical_variables` argument to keep the expected water-balance column structure visible in the rendered datatable even when every scenario in a batch is `NULL` (e.g. the engine returns Status 1 for every input and writes no result HDF5).

Usage

```
default_canonical_wb_variables()
```

Value

A `character()` vector of variable names.

Source

Values observed in the May 2026 432-row Wien sweep (Michael Rustler) and referenced explicitly by `plot_wb_tradeoff_overflows()`.

Examples

```
default_canonical_wb_variables()
```

default_cost_rates

Default unit-cost rates for infiltration-swale installations

Description

Returns the default Austrian unit-cost rates used by `compute_costs()`. Rates were provided by Johannes Leimgruber (OeStaP) on 2026-03-27 for the RAINDROP cost-optimisation work.

Usage

```
default_cost_rates()
```

Value

A named `list` of rates in EUR per m² or m³: `excavation_eur_per_m3`, `profiling_eur_per_m2`, `filter_eur_per_m3`, `infiltration_box_eur_per_m3`, `gravel_trench_eur_per_m3`.

Source rates (Leimgruber, 2026-03-27, "Kostenansätze Optimierung")

- Aushub (excavation, incl. loading + transport): 70 EUR/m³
- Profilierung und Begrünung (profiling + greening): 10 EUR/m²
- Bodenfilter (soil filter, incl. installation): 200 EUR/m³
- Sickerbox (infiltration box, incl. installation): 350 EUR/m³
- Schotterrigol (gravel trench, incl. installation): 50 EUR/m³

`download_engine` *Download the Tandler "Regenwasserbewirtschaftung" calculation engine*

Description

Fetches the Windows executable from the corresponding GitHub Release of the companion `KWB-R/kwb.raindrop.binaries` repository and caches it under a per-version sub-directory of the user cache. Subsequent calls are no-ops once the file is present.

Usage

```
download_engine(
  version = default_engine_version(),
  cache_dir = tools::R_user_dir("kwb.raindrop", "cache"),
  force = FALSE
)
```

Arguments

`version`	`character(1)` Engine release version, matching the part after `engine-` in the `KWB-R/kwb.raindrop.binaries` Release tag. Defaults to the package's pinned version.
`cache_dir`	`character(1)` Directory where engine binaries are cached. A sub-directory named after `version` is used so multiple versions can coexist. Defaults to `tools::R_user_dir("kwb.raindrop", "cache")`.
`force`	`logical(1)` If `TRUE`, re-download even if the executable already exists in the cache. Default: `FALSE`.

Details

Releases in KWB-R/kwb.raindrop.binaries follow the tag scheme engine-<version> and contain a single asset named Regenwasserbewirtschaftung.exe (the version is encoded in the tag, not in the filename, so multiple engine versions can coexist side-by-side in the cache).

The executable is a Windows binary; on non-Windows platforms the file is still downloaded but cannot be executed directly. The function does not validate the download with a checksum — pin a specific version to guarantee reproducibility.

Value

character(1) — absolute path to the cached .exe.

Examples

```
## Not run:
exe <- download_engine()
exe <- download_engine("2026-02-24")

## End(Not run)
```

---

find_single_param_variations

Find scenarios that differ from a reference in exactly one parameter

---

Description

Identifies rows that differ from a reference in *exactly one* parameter. You may pass param_cols explicitly, or let the function infer them by excluding id_col, exclude_cols, and any columns matching exclude_cols_regex (default excludes result columns like h_*).

Usage

```
find_single_param_variations(
  data,
  ref_scenario = "s00001",
  id_col = "scenario_name",
  param_cols = NULL,
  exclude_cols = NULL,
  exclude_cols_regex = "^h_",
  tol = 1e-09,
  quiet = FALSE,
  include_reference = c("per_param", "none")
)
```

Arguments

data	A data frame with scenarios.
ref_scenario	Reference scenario ID (default "s00001").
id_col	Name of the scenario ID column (default "scenario_name").
param_cols	Optional character vector of parameter columns. If NULL, parameters are inferred.
exclude_cols	Character vector to exclude from parameter detection.
exclude_cols_regex	Regex to auto-exclude non-parameter columns (default "^h_").
tol	Numeric tolerance for numeric comparisons (default 1e-9).
quiet	Logical; print diagnostics if FALSE (default).
include_reference	"per_param" (default) to include one reference row per parameter with hits, or "none".

Value

A data frame with columns: id_col, param_name, param_value, is_reference. Attributes: "param_cols", "ref_row", "diff_mat", "n_diff".

---

get_simulation_results_all

Read Raindrop optimisation simulation results (all) from HDF5

Description

Reads per-run result files (HDF5) for both the *measure element* and the *connected area* results and returns a named list (one entry per simulation).

Usage

```
get_simulation_results_all(paths, path_list, simulation_names, debug = TRUE)
```

Arguments

paths	A list of path definitions. Used for messaging and expected to contain file_results_hdf5_element and file_results_hdf5_flaeche (file names). Note: within the loop, paths is overwritten by the resolved paths for the current simulation run.
path_list	A list passed to kwb.utils::resolve() to generate run-specific paths (must yield path_results_hdf5_element, path_results_hdf5_flaeche, and dir_target_output).
simulation_names	Character vector of simulation run identifiers (e.g. c("s00001", "s00002")).
debug	print debug messages (default: TRUE)

Details

For each simulation name (e.g. `"s00001"`), the function resolves the run directory via `kwb.utils::resolve(path_list, dir_target = s_name)` and then loads standard result groups from two HDF5 files:

- **element**: `Metainfo`, `Raten`, `Wasserbilanz`, `Zustandsvariablen`
- **connected_area**: `Metainfo`, `Raten`, `Wasserbilanz`, `Zustandsvariablen`

If either of the expected HDF5 files is missing for a run, the corresponding list entry will be `NULL`.

The function uses `hdf5r::H5File$new(..., mode = "r")` to open the files. The HDF5 handles are not explicitly closed; depending on your workflow, you may want to close them (see `hdf5r::H5File$close_all()` / `$close()`).

Value

A named list with one entry per `simulation_names`. Each entry is either `NULL` (missing files) or a nested list:

element **meta** Data.frame/list of scalar metadata (from `Metainfo`).

rates Time series table (from `Raten`).

water_balance Scalars table (from `Wasserbilanz`).

states Time series table (from `Zustandsvariablen`).

connected_area Same structure as `element`, read from the area HDF5.

See Also

`resolve`, `read_hdf5_scalars`, `read_hdf5_timeseries`

`get_simulation_results_optim`

Read Raindrop optimisation simulation results from HDF5

Description

Reads per-run result files (HDF5) for both the *measure element* and the *connected area* results and returns a named list (one entry per simulation).

Usage

```
get_simulation_results_optim(paths, path_list, simulation_names, debug = TRUE)
```

Arguments

paths	A list of path definitions. Used for messaging and expected to contain file_results_hdf5_element and file_results_hdf5_flaeche (file names). Note: within the loop, paths is overwritten by the resolved paths for the current simulation run.
path_list	A list passed to kwb.utils::resolve() to generate run-specific paths (must yield path_results_hdf5_element, path_results_hdf5_flaeche, and dir_target_output).
simulation_names	Character vector of simulation run identifiers (e.g. c("s00001", "s00002")).
debug	print debug messages (default: TRUE)

Details

For each simulation name (e.g. "s00001"), the function resolves the run directory via kwb.utils::resolve(path_list, dir_target = s_name) and then loads standard result groups from two HDF5 files:

- **element**: Metainfo, Raten, Wasserbilanz, Zustandsvariablen
- **connected_area**: Metainfo, Raten, Wasserbilanz, Zustandsvariablen

If the **element** HDF5 is missing the entry is NULL; if only the **connected-area** HDF5 is missing, the entry is a list with the element side populated and connected_area = NULL. HDF5 handles are closed on exit via on.exit(...$close_all()).

The function uses hdf5r::H5File$new(..., mode = "r") to open the files and registers an on.exit(...$close_all()) so handles are released even when the iteration body errors out.

Value

A named list with one entry per simulation_names. Each entry is either NULL (element HDF5 missing) or a nested list:

element **meta** Data.frame/list of scalar metadata (from Metainfo).

rates Time series table (from Raten).

water_balance Scalars table (from Wasserbilanz).

states Time series table (from Zustandsvariablen).

connected_area Same structure as element, read from the connected-area HDF5, or NULL if that file is missing for the run.

See Also

resolve, read_hdf5_scalars, read_hdf5_timeseries

`get_simulation_results_optim_parallel` *Read Raindrop optimisation simulation results from HDF5 (parallel via future.apply + progress)*

Description

Parallel variant of `get_simulation_results_optim()` using `future.apply::future_lapply()` including optional progress reporting via **progressr**.

Usage

```
get_simulation_results_optim_parallel(
  paths,
  path_list,
  simulation_names,
  debug = TRUE,
  workers = NULL,
  show_progress = TRUE,
  future_seed = TRUE,
  progress_handler = progressr::handler_txtprogressbar
)
```

Arguments

`paths`	A list of path definitions. Used for messaging and expected to contain `file_results_hdf5_element` and `file_results_hdf5_flaeche` (file names). Note: within the loop, `paths` is overwritten by the resolved paths for the current simulation run.
`path_list`	A list passed to `kwb.utils::resolve()` to generate run-specific paths (must yield `path_results_hdf5_element`, `path_results_hdf5_flaeche`, and `dir_target_output`).
`simulation_names`	Character vector of simulation run identifiers (e.g. `c("s00001", "s00002")`).
`debug`	print debug messages (default: TRUE)
`workers`	Optional number of parallel workers. If not NULL, a temporary `future::multisession` plan is set.
`show_progress`	Logical (default TRUE).
`future_seed`	Passed to `future.apply::future_lapply()`.
`progress_handler`	A progressr handler function (default: `progressr::handler_txtprogressbar`). If NULL, no handler is set.

Value

Named list (see `get_simulation_results_optim`). As with the non-parallel sibling, the entry for a run is `NULL` only when the element HDF5 is missing; a missing connected-area HDF5 yields a partial result with `connected_area = NULL`.

h5_ensure_dataset *Ensure that an HDF5 dataset exists (create if missing)*

Description

Creates missing groups along the path and then creates the dataset. Designed for RAINDROP input files where missing scalar parameters can crash the model definition reader.

Usage

```
h5_ensure_dataset(h5, path, value, dtype = NULL, dims = NULL)
```

Arguments

h5	Open hdf5r::H5File (mode "a" or "r+").
path	Absolute dataset path (e.g. "/Berechnungsparameter/Zeitschritt_Verschaltungen").
value	Initial value to write after creation (scalar, vector, matrix, or 2-col TS).
dtype	Optional. Either an H5T object or a string ("double","integer","logical","character"). If NULL, inferred from value.
dims	Optional integer vector. If NULL, inferred from value.

Value

Invisibly TRUE

h5_ensure_datasets_from_values *Ensure that all datasets referenced by a values list exist*

Description

Creates missing groups and datasets for all names(values). Datasets are created with dtype/dims inferred from the corresponding value. No values are written here.

Usage

```
h5_ensure_datasets_from_values(
  h5,
  values,
  ts_cols = c("time", "value"),
  ts_layout = c("2xN", "Nx2"),
  ts_dtype = "double",
  strict = TRUE
)
```

Arguments

h5	Open hdf5r::H5File (mode "a" or "r+").
values	Named list; names are dataset paths (leading // allowed).
ts_cols	Character(2). Column names for TS (default time/value).
ts_layout	One of "2xN" or "Nx2". For RAINDROP typically "2xN" in HDF5.
ts_dtype	Either an H5T object or a string ("double","integer","logical","float").
strict	Stop if creation fails.

Value

Invisibly, the character vector of created dataset paths.

---

h5_read_values *Read values of all (or selected) datasets*

Description

Read values of all (or selected) datasets

Usage

```
h5_read_values(
  h5,
  paths = NULL,
  simplify_scalars = TRUE,
  timeseries_as_tibble = TRUE,
  ts_names = c("time", "value")
)
```

Arguments

h5	An open hdf5r::H5File.
paths	character vector; if NULL read all.
simplify_scalars	logical: simplify scalar datasets to length-1 atoms.
timeseries_as_tibble	logical: convert 2xN / Nx2 arrays to tibble(time,value).
ts_names	character(2): names for time/value columns.

Value

Named list.

h5_validate_write *Validate what would be written where (pre-flight check)*

Description

Summarizes for each path:

- current dataset dims/maxdims (HDF5 order)
- intended value shape (R order)
- intended write shape (HDF5 order; for TS always 2xN)
- decision: SKIP / WRITE / RESIZE / NEED_RECREATE / ERROR

Usage

```
h5_validate_write(h5, values, ts_cols = c("time", "value"))
```

Arguments

h5	An open hdf5r::H5File.
values	Named list of values to write (names are HDF5 paths).
ts_cols	Character(2). Column names for TS (default time/value).

Details

Notes:

- 2-column data.frames/tibbles are treated as Nx2 in R and mapped to 2xN in HDF5.
- Many RAINDROP files store time series as HDF5 dims 2xN (appearing as Nx2 in R).

Value

tibble

h5_write_values *Write (updated) values back into existing HDF5 datasets (robust for your hdf5r build)*

Description

- Scalars: write with required args (args=list() or args=1L fallback).
- 2-col TS (data.frame/tibble): expects Nx2 in R; writes as 2xN in HDF5 (RAINDROP style), using explicit hyperslab args=list(1:2, 1:N) to avoid empty selections.
- If TS length changes and dataset maxdims blocks resize, the dataset is deleted via parent$link_delete(name) and recreated with dims=2xN (fixed), then written. (No maxdims argument used; compatible with your create_dataset signature.)

Usage

```
h5_write_values(
  h5,
  values,
  resize = TRUE,
  strict = TRUE,
  ts_cols = c("time", "value"),
  scalar_strategy = c("error", "first", "collapse"),
  collapse_sep = ";",
  ts_dtype = "double",
  verbose = FALSE
)
```

Arguments

h5	Open hdf5r::H5File.
values	Named list; names are dataset paths (leading // allowed).
resize	Logical. If TRUE tries set_extent() where possible.
strict	Logical. If TRUE stop on first error else warn and continue.
ts_cols	Character(2). Column names for TS (default time/value).
scalar_strategy	"error"|"first"|"collapse".
collapse_sep	Separator for collapse.
ts_dtype	Either an H5T object or one of "double","float","integer","logical".
verbose	Logical.

Value

Invisibly, written paths.

---

list_h5_datasets *List all datasets (recursive)*

Description

List all datasets (recursive)

Usage

```
list_h5_datasets(h5)
```

Arguments

h5	An open hdf5r::H5File.

Value

A tibble with columns: path, obj_type, dims, maxdims.

---

plot_hpond_vs_ref *Plot the influence of single-parameter variations on a response*

Description

Uses find_single_param_variations() (by default excluding ^h_ columns from parameter detection) and plots absolute or percentage deviations of the chosen response. The title shows the reference value in mm.

Usage

```
plot_hpond_vs_ref(
  data,
  response,
  ref_scenario = "s00001",
  diff = c("abs", "pct"),
  param_cols = NULL,
  exclude_cols = NULL,
  exclude_cols_regex = "^h_",
  tol = 1e-09,
  quiet = FALSE
)
```

Arguments

data	Data frame with at least scenario_name, parameter columns, and response.
response	Character; the response column to plot (e.g. "h_pond_max" or "h_pond_mean").
ref_scenario	Reference scenario ID (default "s00001").
diff	"abs" for absolute or "pct" for percentage deviation.
param_cols	Optional explicit parameter columns (recommended if you prefilter).
exclude_cols	Extra columns to exclude from parameter detection.
exclude_cols_regex	Regex to auto-exclude non-parameter columns (default "^h_").
tol	Numeric tolerance passed through to find_single_param_variations().
quiet	Logical; diagnostics if FALSE.

Value

A ggplot object.

`plot_main_effects` *Plot main effects of multiple parameters on an outcome (violin/box/jitter)*

Description

Creates a facetted overview plot showing the distribution of an outcome (for example `n_overflows`) across the tested levels of multiple varied parameters. Parameters are sorted by a simple effect-size proxy: the range of median outcome values across parameter levels.

Usage

```
plot_main_effects(
  df,
  y = "n_overflows",
  params,
  max_levels = 25,
  ylim_lower = 0,
  lang = c("de", "en")
)
```

Arguments

`df`	A data.frame or tibble containing the outcome column `y` and the parameter columns listed in `params`.
`y`	Character scalar. Name of the outcome column to plot on the y-axis. Defaults to `"n_overflows"`.
`params`	Character vector of parameter column names in `df` to consider.
`max_levels`	Integer. Parameters with more than `max_levels` distinct values are dropped to keep the plot readable. Defaults to 25.
`ylim_lower`	Numeric scalar or `NULL`. Optional lower display limit for the y-axis. Uses `ggplot2::coord_cartesian()` so data are not removed before computing violin and boxplots. Defaults to `0`.
`lang`	Character. Plot language: `"de"` or `"en"`.

Details

The function is intended for optimisation or sensitivity grids with many parameters, where a single 2D scatter plot is not informative.

The plot language can be switched via `lang = "de"` or `lang = "en"`. This affects the title, y-axis label, and selected parameter labels.

Value

A ggplot object with facetted violin, boxplot, and jitter layers.

plot_valid_design_space *Valid solutions in design space (x × y) with overflow-threshold discrete color scale*

Description

Visualises model results in a 2D design space (by default `mulde_area` vs. `mulde_height`) and highlights valid solutions (by default `n_overflows <= valid_max`). The plot is prepared for direct use with `plotly::ggplotly(..., tooltip = "text")` by mapping an HTML tooltip via `aes(text = ...)` that lists all other varied parameters from `param_grid`, excluding `x` and `y`.

Usage

```
plot_valid_design_space(
  param_grid,
  sim_results,
  id_col = "scenario_name",
  overflow_col = "n_overflows",
  valid_max = 1,
  x = "mulde_area",
  y = "mulde_height",
  max_levels = 50,
  alpha_invalid = 0.12,
  size = 2.4,
  digits = 4,
  drop_overflow_gt_valid_max = FALSE,
  jitter = FALSE,
  jitter_width = NULL,
  jitter_height = NULL,
  jitter_factor = 0.005,
  alpha_mode = c("none", "duplicates"),
  alpha_min = 0.2,
  alpha_max = 1,
  keep_param_grid_limits = TRUE,
  lang = c("de", "en"),
  title = NULL,
  subtitle = NULL,
  legend_position = "top"
)
```

Arguments

`param_grid`	A data.frame/tibble containing the scenario id (`id_col`) and parameters.
`sim_results`	A data.frame/tibble containing the scenario id (`id_col`) and `overflow_col`.
`id_col`	Character. Join key (scenario id), default `"scenario_name"`.

overflow_col	Character. Overflow outcome column, default "n_overflows".
valid_max	Numeric. Validity threshold: overflow_col <= valid_max. Also used as the orange breakpoint in the color scale.
x	Character. Name of the x-axis column, default "mulde_area".
y	Character. Name of the y-axis column, default "mulde_height".
max_levels	Integer. Parameter columns with more than max_levels distinct values are ignored in the tooltip list. Default 50.
alpha_invalid	Numeric. Extra alpha multiplier for invalid solutions (background layer).
size	Numeric. Base point size.
digits	Integer. Significant digits for numeric tooltip values.
drop_overflow_gt_valid_max	Logical. If TRUE, scenarios with overflow_col > valid_max are not plotted at all. Default FALSE.
jitter	Logical. If TRUE, apply jitter, useful when many points share identical x/y.
jitter_width	Numeric or NULL. Jitter width in data units. If NULL, defaults from x-range.
jitter_height	Numeric or NULL. Jitter height in data units. If NULL, defaults from y-range.
jitter_factor	Numeric. Factor used to derive default jitter width/height from x/y ranges. Default 0.005.
alpha_mode	Character. Either "none" for fixed alpha or "duplicates" to vary alpha by the number of scenarios sharing the same x/y. Default "none".
alpha_min	Numeric. Minimum alpha used when alpha_mode = "duplicates". Default 0.20.
alpha_max	Numeric. Maximum alpha used when alpha_mode = "duplicates". Default 1.00.
keep_param_grid_limits	Logical. If TRUE and drop_overflow_gt_valid_max = TRUE, the x/y axis limits are fixed to the full range or full set of levels found in param_grid, so the design-space axes do not shrink after filtering. Default TRUE.
lang	Character. Plot language: "de" or "en".
title	Character or NULL. Plot title. If NULL, a language-specific default title is used.
subtitle	Character or NULL. Plot subtitle. If NULL, a language-specific default subtitle is used.
legend_position	Character. Legend position, e.g. "top", "bottom", "left", or "right". Default "top".

Details

Varied parameters are detected automatically as columns in param_grid with more than one distinct value. Optionally, scenarios with overflow_col > valid_max can be removed entirely (drop_overflow_gt_valid_max = TRUE). To help identify multiple scenarios that share identical x/y coordinates but differ in other parameters, optional jitter and/or variable transparency can be applied.

Discrete color mapping (threshold-style):

- **dark green** at `n_overflows = 0`
- a discrete palette from dark green to yellow-green for integer levels `1..(valid_max-1)`
- **orange** at `n_overflows = valid_max`
- **red** for `n_overflows > valid_max`, shown as level `">valid_max"`

The plot language can be switched via `lang = "de"` or `lang = "en"`. This affects title, subtitle, legend title, tooltip labels, and selected axis labels.

Value

A ggplot object. Tooltip text is mapped via `aes(text = ...)`.

`plot_wb_tradeoff_overflows` *Trade-off plot: Infiltration vs. Evapotranspiration (discrete colors by overflow threshold)*

Description

Creates a scatter plot from optimisation results with element infiltration on the x-axis and element evapotranspiration on the y-axis.

Usage

```
plot_wb_tradeoff_overflows(
  simulation_results_optimisation,
  param_grid,
  x = 1,
  filter_n_gtx = FALSE,
  use_jitter = TRUE,
  jitter_width = 0.15,
  jitter_height = 0.15,
  jitter_seed = 1L,
  digits = 2L,
  digits_params = 4L,
  lang = c("de", "en"),
  title = NULL,
  lab_x = NULL,
  lab_y = NULL,
  legend_position = "top"
)
```

Arguments

simulation_results_optimisation	Data frame with simulation results. Required columns are scenario_name, n_overflows, sum_overflows, element.WB_InfiltrationNetto_, element.WB_Evapotranspiratio and element.WB_Oberflaechenablauf_Ueberlauf_.
param_grid	Data frame with parameter grid. Must contain scenario_name.
x	Numeric, typically integer. Threshold for overflow coloring. Values greater than or equal to x are mapped to the red category ">=x".
filter_n_gtx	Logical. If TRUE, scenarios with n_overflows >= x are removed before plotting.
use_jitter	Logical. If TRUE, slight jitter is applied to reduce overplotting.
jitter_width, jitter_height	Numeric. Jitter strength in x- and y- direction, only used if use_jitter = TRUE.
jitter_seed	Integer. Seed for reproducible jitter.
digits	Integer. Number of digits used for rounding water balance values in the tooltip.
digits_params	Integer. Number of digits used for rounding numeric parameter values in the tooltip.
lang	Character. Plot language: "de" or "en".
title	Character or NULL. Plot title. If NULL, a language-specific default title is used.
lab_x	Character or NULL. X-axis label. If NULL, a language-specific default label is used.
lab_y	Character or NULL. Y-axis label. If NULL, a language-specific default label is used.
legend_position	Character. Legend position, e.g. "top", "bottom", "left", or "right". Default "top".

Details

Coloring is *discrete* based on n_overflows and a threshold x:

- **dark green** for n_overflows = 0
- a discrete palette from dark green to yellow-green for integer levels 1..(x-1)
- **red** for n_overflows >= x, shown as category ">=x"

The plot language can be switched via lang = "de" or lang = "en". This affects title, axis labels, legend title, and tooltip labels unless custom labels are supplied explicitly.

Tooltip text additionally includes all parameters from param_grid that vary across scenarios, excluding scenario_name.

Value

A ggplot object.

read_hdf5_connections *Read surface-water connections from an HDF5 results file*

Description

Reads an HDF5 file that contains surface-water *connections* (formerly "Verschaltungen"), e.g. a file like `s78853_Verschaltungen.h5`. The file is expected to have:

- a root-level scalar dataset `"Anzahl Verschaltungen"`, and
- one group per connection (`"0"`, `"1"`, ...) that contains
  - scalar metadata (e.g. `von_Layer`, `von_Massnahmenelement`, `nach_Layer`, `nach_Massnahmenelement`) and
  - time-series datasets stored as 2×N matrices (e.g. `Qtatsaechlich`, `delta_h_pond_*`, `delta_theta_*`).

Scalar metadata are returned

- als breite Tabelle (`meta`),
- als lange Tabelle (`scalars`) und
- pro Verbindung in einer Unterliste (`connections$group_0`, ...).

Time series werden in Long-Format zurückgegeben und mit den Metadaten verknüpft.

Usage

```
read_hdf5_connections(file)
```

Arguments

`file`	An `hdf5r::H5File` object pointing to a `*_Verschaltungen.h5` file, already opened in read mode.

Value

A named list with components:

n_connections `integer(1)`. Value of the root dataset `"Anzahl Verschaltungen"` (or `NA_integer_` if not present).

meta A tibble with one row per connection: `connection_id`, `from_layer`, `from_element`, `to_layer`, `to_element`.

timeseries A tibble with long-format time series for all connections: `connection_id`, `variable`, `time`, `value`, `from_layer`, `from_element`, `to_layer`, `to_element`.

scalars A tibble with all scalar datasets per connection in long format: `connection_id`, `variable`, `value` (list), `type`.

connections A named list of per-connection sublists: `connections$group_0`, `connections$group_1`, ...; each containing `connection_id`, `meta`, `timeseries`, `scalars` für genau diese Verbindung.

Examples

```
## Not run:
  library(hdf5r)

  f <- H5File$new(paths$path_verschaltungen_hdf5, mode = "r")
  conn <- read_hdf5_connections(f)

  conn$n_connections
  conn$meta
  conn$timeseries

  # Unterliste für die erste Verbindung:
  conn$connections$group_0$meta
  conn$connections$group_0$timeseries

## End(Not run)
```

`read_hdf5_scalars` *Read scalar datasets from an HDF5 group*

Description

Reads all scalar datasets contained in an HDF5 group and returns them as a tibble. By default, only numeric / integer scalars are returned. If non-numeric scalars (e.g. strings) are present, they can either be dropped with a warning or kept in a list-column.

Usage

```
read_hdf5_scalars(group, numeric_only = TRUE)
```

Arguments

`group`	An `hdf5r::H5Group` object. Direct children of this group are expected to be scalar datasets (i.e. `dataset.dims == 0`).
`numeric_only`	Logical (default: `TRUE`). If `TRUE`, only numeric / integer scalars are returned (others are dropped with a warning). If `FALSE`, all scalars are returned in a list-column `value` together with a `type` column.

Value

If `numeric_only = TRUE`: A tibble with columns

variable `character`. Dataset name within the group.

value `numeric`. Scalar value read from the dataset.

If `numeric_only = FALSE`: A tibble with columns

variable `character`. Dataset name within the group.

value `list`. Scalar values (numeric, integer, character, ...).

type `character`. First class of each value (e.g. `"numeric"`, `"character"`).

Examples

```
## Not run:
  # Wasserbilanz: nur numerische Skalare
  wb_tbl <- read_hdf5_scalars(res_hdf5_element[["Wasserbilanz"]])

  # Metainfo: gemischte Typen (Integer + String)
  meta_tbl <- read_hdf5_scalars(res_hdf5_element[["Metainfo"]],
                                numeric_only = FALSE)

## End(Not run)
```

`read_hdf5_timeseries` *Read HDF5 time series datasets from a group (supports deeperLayers)*

Description

Reads all datasets in an HDF5 group and returns a long tibble with columns: variable, time, value.

Usage

```
read_hdf5_timeseries(
  ts_groupvariable,
  deeper_layers_pattern = "deeperLayers|deeper_layers"
)
```

Arguments

`ts_groupvariable`
: `hdf5r::H5Group`

`deeper_layers_pattern`
: regex to detect deeper-layers datasets

Details

Supported dataset layouts:

- k x N (rows): `[1, ]` = time, `[2..k, ]` = values (series)
- N x k (cols): `[, 1]` = time, `[, 2..k]` = values (series)

Special handling for names containing "deeperLayers"/"deeper_layers":

- The value series represent layers below layer 1.
- Output variable names get suffixed with the layer-id: _2, _3, ...
- If the dataset contains only time (no value series), it returns 0 rows.

Value

tibble::tibble(variable, time, value)

`read_raindrop_errors` *Read RAINDROP error logs into a nested tibble*

Description

Reads the RAINDROP error protocol HDF5 (typically `Fehlerprotokoll.h5`) for a set of simulation runs and returns a tibble with one row per run and a nested tibble column containing all errors found in the file.

Usage

```
read_raindrop_errors(simulation_names, path_list, debug = FALSE)
```

Arguments

`simulation_names`	Character vector of simulation run names (e.g. `"s00001"`).
`path_list`	A path list object passed to `kwb.utils::resolve()`. Must resolve to an element `path_errors_hdf5`.
`debug`	Logical. If `TRUE`, pass through to `kwb.utils::catAndRun()`.

Details

The error HDF5 usually contains:

- `/AnzahlFehler` (scalar integer)
- groups `/0`, `/1`, ... each containing datasets such as `Fehlerbeschreibung`, `Layer1`, `Layer2`, `Massnahmenelement1`, `Massnahmenelement2`

Note: In **hdf5r**, HDF5 group names must be addressed as character strings. So group `0` must be accessed as `h5[["0"]]` (not `h5[[0]]`).

Value

A tibble with columns:

- `id` Integer simulation id (parsed from `simulation_names`).
- `scenario_name` The simulation name (e.g. `"s00001"`).
- `path` File path to the error HDF5.
- `file_exists` Logical.
- `number_of_errors` Scalar integer or `NA`.
- `errors` Nested tibble (list-column) with per-error rows: `error_index`, `Fehlerbeschreibung`, `Layer1`, `Layer2`, `Massnahmenelement1`, `Massnahmenelement2`

Examples

```
## Not run:
errors_df <- read_raindrop_errors(simulation_names, path_list)

# flatten all errors:
all_errors <- tidyr::unnest(errors_df, errors)

## End(Not run)
```

`run_model` *Run an rainwater management model executable with an input file*

Description

Builds and runs a system command of the form "<path_exe> <path_input>", after normalising both paths to absolute paths. Progress and (optionally) command output are wrapped with `kwb.utils::catAndRun()` for neat logging.

Usage

```
run_model(path_exe, path_input, print_output = FALSE, debug = TRUE, ...)
```

Arguments

`path_exe`	`character(1)` Path to the model executable (e.g., a `.exe` on Windows). The file must exist.
`path_input`	`character(1)` Path to the model input file passed as the single argument to the executable. The file must exist.
`print_output`	`logical(1)` If `TRUE`, stream the process output to the console and return the exit status (integer). If `FALSE` (default), capture and return the command output as a character vector.
`debug`	`logical(1)` Forwarded to `kwb.utils::catAndRun(dbg = ...)` to enable/disable the progress message. Default: `TRUE`.
`...`	Additional arguments passed to `base::shell()`, e.g. `timeout` on Windows. See `?base::shell` for details.

Details

Both `path_exe` and `path_input` are converted to absolute, normalised paths via `fs::path_abs()` and `base::normalizePath()`. The command is executed with `base::shell()`, which on Windows invokes the system shell. On non-Windows platforms, prefer `base::system()` if you need full POSIX semantics.

Value

If `print_output = FALSE`, a character vector containing the captured standard output of the command. If `print_output = TRUE`, the (invisible) integer exit status returned by `shell()` (0 indicates success).

Side effects

Executes an external program that may read/write files depending on the model. Ensure you trust the executable and paths provided.

See Also

`base::shell()`, `fs::path_abs()`, `kwb.utils::catAndRun()`

Examples

```
## Not run:
# Example: run a hypothetical model with an input file
exe   <- "C:/path/to/model.exe"
input <- "C:/path/to/input.h5"

# Capture output as character vector
out <- run_model(exe, input, print_output = FALSE)

# Stream output and get exit status
status <- run_model(exe, input, print_output = TRUE)

## End(Not run)
```

`run_scenarios` *Run scenarios (parallel or sequential) with a user-supplied worker function*

Description

Executes scenarios by applying `run_one_scenario` to each element of `indices`. Supports parallel execution via **future.apply** and sequential execution for debugging. Optionally shows progress via **progressr**.

Usage

```
run_scenarios(
  indices,
  run_one_scenario,
  timestep_hours,
  debug = FALSE,
  ...,
```

```
  parallel = TRUE,
  workers = parallel::detectCores(),
  show_progress = TRUE,
  progress_handler = "txtprogressbar"
)
```

Arguments

indices	Vector. Scenario identifiers to iterate over (often integer row indices).
run_one_scenario	Function. Worker function with signature function(x, timestep_hours, debug, ...). Must accept the arguments timestep_hours and debug.
timestep_hours	Numeric. Time step (hours) forwarded to run_one_scenario.
debug	Logical. Debug flag forwarded to run_one_scenario.
...	Additional arguments forwarded to run_one_scenario.
parallel	Logical. If TRUE, use future.apply::future_lapply. If FALSE, use base lapply.
workers	Integer. Number of workers when parallel = TRUE. Defaults to parallel::detectCores().
show_progress	Logical. If TRUE, show progress.
progress_handler	Character. Progress handler key. One of "txtprogressbar", "rstudio", "cli".

Value

A list with one element per indices entry containing the return values of run_one_scenario.

Index