# Package 'dataplane'

January 12, 2026

**Title** Data Architecture and Processing Helpers

**Version** 0.1.0

**Description** Tools for structuring, validating, and transforming datasets with a focus on repeatable pipelines and clear metadata.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**URL** https://github.com/jclark50/dataplane

**BugReports** https://github.com/jclark50/dataplane/issues

**Imports** arrow,
data.table,
digest,
dplyr,
httr2,
jsonlite

## Contents

---

`cbindlist` *Column-bind a list of objects*

---

## Description

Thin wrapper around ‘do.call(cbind, ...)‘.

## Usage

```
cbindlist(alist)
```

## Arguments

| | |
|---|---|
| `alist` | List of objects with compatible row counts. |

## Value

The column-bound result.

## Examples

```
cbindlist(list(a = 1:3, b = 4:6))
```

---

`closeEnvironment` *Gracefully Stop the Watchdog and Cluster*

---

## Description

The 'closeEnvironment()' function stops the watchdog by creating the stop file, then uses 'jstopCluster()' to kill the associated Rscript.exe processes.

## Usage

```
closeEnvironment(env_info, remove_watchdog_stopfile = TRUE)
```

## Arguments

| | |
|---|---|
| `env_info` | A list as returned by 'startEnvironment()', containing 'cluster', 'stop_file', and 'worker_pids'. |
| `remove_watchdog_stopfile` | Boolean. If true, removes stopfile corresponding to this watchdog/environment. Default is TRUE. |

## Details

This function is useful for a clean shutdown after your computations are finished. It signals the watchdog script to stop by creating the specified stop file, and then forcefully kills the worker processes to ensure a clean slate.

After calling 'closeEnvironment()', if you also wish to stop the parallel cluster, call: `parallel::stopCluster(env_info`

## Value

NULL (invisibly).

## Examples

```
## Not run:
env_info <- startEnvironment(num_cores = 2)

# ... run computations ...

closeEnvironment(env_info)
# Now the watchdog should stop and the processes are terminated.

parallel::stopCluster(env_info$cluster)

## End(Not run)
```

## collapse *Collapse strings into a quoted, delimited single string*

### Description

Convenience helper for building strings like: ‘’a’, ’b’, ’c’‘

### Usage

```
collapse(someStrings, collapsewith = "', '", leadendwith = "'")
```

### Arguments

| | |
|---|---|
| someStrings | Vector/list of strings. |
| collapsewith | Character. Delimiter placed between values. |
| leadendwith | Character. String placed at the start and end (often a quote). |

### Value

A single character string.

### Examples

```
collapse(c("a","b","c"))
collapse(c("x","y"), collapsewith = ",", leadendwith = "\"")
```

## deleteNULL *Remove NULL elements from a list*

### Description

Remove NULL elements from a list

### Usage

```
deleteNULL(alist)
```

### Arguments

| | |
|---|---|
| alist | A list. |

### Value

A list with all ‘NULL‘ entries removed.

### Examples

```
deleteNULL(list(a = 1, b = NULL, c = 3))
```

## filterCols *Filter column names by storage type*

### Description

Returns the column names whose *primary class* matches a target set, or (if ‘exclude = TRUE‘) those that do not.

### Usage

```
filterCols(dat, colType = "numeric", exlcludedatetime = TRUE, exclude = FALSE)
```

### Arguments

| | |
|---|---|
| dat | A data.frame / data.table. |
| colType | Character. Either ‘"numeric"‘ (default) or a character vector of class names (e.g., ‘c("character")‘). |
| exlcludedatetime | Logical. If ‘TRUE‘, columns with primary class in ‘c("Date","POSIXct","POSIXt")‘ are excluded from the match set. |
| exclude | Logical. If ‘TRUE‘, returns columns *not* matching ‘colType‘. |

### Value

Character vector of column names.

### Examples

```
d <- data.frame(x = 1:3, y = c("a","b","c"), t = as.POSIXct("2026-01-01") + 1:3)
filterCols(d, "numeric")
filterCols(d, "character")
filterCols(d, "numeric", exclude = TRUE)
```

## flatten_list_columns *Flatten list-columns into character columns*

### Description

For any list-column, converts each element to a single string:

- ‘NULL‘ -> ‘NA‘
- length-1 -> that value
- length>1 -> comma-separated string

### Usage

```
flatten_list_columns(dt)
```

### Arguments

| | |
|---|---|
| dt | A data.frame / tibble / data.table-like object. |

### Value

An object of the same class as ‘dt‘, with list-columns flattened.

### Examples

```
x <- data.frame(a = 1:3, b = I(list("x", c("y","z"), NULL)))
flatten_list_columns(x)
```

---

## jstopCluster *Forcefully Stop Worker Processes*

---

### Description

‘jstopCluster()‘ attempts to kill the processes by PID. On Windows, uses ‘taskkill /F /PID‘. On Linux, uses ‘kill -9‘.

### Usage

```
jstopCluster(workerpids = NULL)
```

### Arguments

| | |
|---|---|
| workerpids | Integer vector of worker PIDs. If ‘NULL‘, kills all ‘Rscript.exe‘ (Windows) or ‘R‘ (Linux) processes. Use with caution. |

### Value

NULL invisibly.

### Examples

```
## Not run:
jstopCluster(env_info$worker_pids)

## End(Not run)
```

## klimo_attach_units *Attach declared units to columns (sets attr(x[[col]], "unit"))*

### Description

This does NOT convert values. It is simply a "tagging" step based on metadata.

### Usage

```
klimo_attach_units(
  dt,
  audit_dt = NULL,
  spec_dt = NULL,
  prefer = c("declared_units", "spec"),
  system = c("metric", "imperial"),
  warn_unmatched = FALSE
)
```

### Arguments

| | |
|---|---|
| dt | A data.table. |
| audit_dt | Parsed audit table from metadata (preferred), or spec_dt. |
| prefer | Character: "declared_units" (audit) or "metric_units"/"imperial_units" (spec). |
| system | If using spec_dt units columns, which system to use. |
| warn_unmatched | Warn if declared units exist for columns not present. |

### Value

data.table (modified copy)

## klimo_decode_encoded *Decode spec-declared encoded columns (encoding='scaled_int')*

### Description

For columns declared as already encoded, creates a decoded numeric column.

### Usage

```
klimo_decode_encoded(dt, spec_dt, suffix = "_decoded", drop_encoded = FALSE)
```

### Arguments

| | |
|---|---|
| dt | data.table |
| spec_dt | parsed spec table (e.g., meta$spec_dt) |
| suffix | suffix for decoded column name (default "_decoded") |
| drop_encoded | whether to drop the encoded/original column |

## klimo_decode_scaled *Decode scaled integer columns back to floats (using audit metadata)*

### Description

If your writer created columns like ta_i with scale=100, this recreates ta = ta_i / 100. Optionally drops the scaled integer columns afterward.

### Usage

```
klimo_decode_scaled(
  dt,
  audit_dt,
  prefer_scaled_status = TRUE,
  drop_scaled_cols = FALSE,
  overwrite = FALSE
)
```

### Arguments

| | |
|---|---|
| dt | A data.table. |
| audit_dt | Parsed audit table from metadata. |
| prefer_scaled_status | If TRUE, only decode rows where audit says did_scale==TRUE or scale_status=="scaled". If FALSE, decode whenever scaled_column + suggested_scale exist. |
| drop_scaled_cols | Drop the *_i columns after decoding. |
| overwrite | If TRUE, overwrite existing base columns if present. |

### Value

data.table (modified copy)

## klimo_meta_summary *Quick human summary of Klimo metadata for a parquet file*

### Description

Quick human summary of Klimo metadata for a parquet file

### Usage

```
klimo_meta_summary(path, max_show = 12L)
```

`klimo_open_dataset_with_meta` *Open an Arrow Dataset plus a reference Klimo metadata bundle*

## Description

IMPORTANT: Arrow datasets do not reliably carry per-file schema metadata in a way that supports your Klimo spec/audit workflow. The practical pattern is: - choose ONE reference file (e.g., first in directory) - read its metadata as the dataset-level metadata - optionally validate other files share the same fingerprint (if you store one)

## Usage

```
klimo_open_dataset_with_meta(path, ref_file = NULL, ...)
```

## Arguments

| | |
|---|---|
| `path` | Directory (partitioned dataset) OR vector of parquet files. |
| `ref_file` | Optional explicit reference file; if NULL, uses the first file found. |
| `...` | Passed to arrow::open_dataset |

## Value

list(ds=<Arrow Dataset>, meta=<metadata bundle>, ref_file=<path>)

`klimo_read_parquet` *Read a Parquet file into a data.table, optionally using Klimo metadata*

## Description

Read a Parquet file into a data.table, optionally using Klimo metadata

## Usage

```
klimo_read_parquet(
  path,
  metadata = c("apply", "return", "ignore"),
  decode_scaled = TRUE,
  attach_units = TRUE,
  unit_source = c("audit", "spec"),
  system = c("metric", "imperial"),
  ...
)
```

### Arguments

| | |
|---|---|
| path | Parquet path. |
| metadata | One of: - "ignore": do not parse metadata - "return": parse and return metadata but do not change data - "apply": parse and apply selected behaviors (decode scaled, attach units) |
| decode_scaled | If TRUE and metadata is available, attempt to decode scaled columns. |
| attach_units | If TRUE and metadata is available, attach declared unit attributes. |
| unit_source | "audit" uses declared_units (most direct); "spec" uses metric/imperial fields. |
| system | When unit_source="spec", choose which units column to use. |
| ... | Passed to arrow::read_parquet (e.g., col_select) |

### Value

If metadata=="ignore": data.table Else: list(dt=..., meta=..., spec_dt=..., audit_dt=...)

---

## klimo_read_parquet_meta

*Read Klimo Parquet schema metadata (spec + audit)*

---

### Description

Read Klimo Parquet metadata (spec + audit) without loading full data

### Usage

```
klimo_read_parquet_meta(path, parse = TRUE)

klimo_read_parquet_meta(path, parse = TRUE)
```

### Arguments

| | |
|---|---|
| path | Parquet file path. |
| parse | Logical. If TRUE, parse klimo JSON blobs into data.tables. |

### Value

A list with kv (named list), kv_dt (data.table), spec_dt (data.table or NULL), audit_dt (data.table or NULL)

## klimo_spec_default *Build a default Klimo spec tailored to a dataset*

### Description

‘klimo_spec_default()‘ takes a dataset (data.frame/data.table) and returns a ‘klimo_spec‘ whose rows match the dataset columns. Columns not recognized by the catalog are kept with ‘NA‘ concept/units/scale (so you can still attach partial metadata).

### Usage

```
klimo_spec_default(
  x,
  system = c("metric", "imperial"),
  include_unmatched = TRUE
)
```

### Arguments

| | |
|---|---|
| x | A data.frame/data.table. Only column names are used. |
| system | ‘"metric"‘ or ‘"imperial"‘. This is a *label* used when selecting declared units (metric_units vs imperial_units) by downstream functions. |
| include_unmatched | If TRUE, include all columns (unmatched ones get NA fields). If FALSE, return only the matched subset. |

### Value

A ‘klimo_spec‘ object.

## klimo_spec_preview *Preview how a spec applies to a dataset (coverage + units + scaling/encoding)*

### Description

Preview how a spec applies to a dataset (coverage + units + scaling/encoding)

### Usage

```
klimo_spec_preview(
  x,
  field_spec,
  declared_system = c("metric", "imperial"),
  scaled = FALSE,
  scaled_suffix = "_i"
)
```

### Arguments

| | |
|---|---|
| x | data.frame/data.table |
| field_spec | klimo_spec or spec dt/df or "metric"/"imperial" |
| declared_system | Only used if field_spec is a dt/df (not a klimo_spec). |
| scaled | Whether writer scaling is requested (controls scale_status output). |
| scaled_suffix | Suffix for writer-scaled integer columns. |

### Value

data.table preview

---

## klimo_spec_set_scale *Update scaling in a Klimo spec (post-hoc)*

---

### Description

Convenience helper to modify ‘suggested_scale‘ for one or more columns after building a spec. This edits the spec only; it does not scale any data.

### Usage

```
klimo_spec_set_scale(spec, columns, suggested_scale)
```

### Arguments

| | |
|---|---|
| spec | A ‘klimo_spec‘ object. |
| columns | Character vector of column names to edit. Matching is case-insensitive (via normalized column names). |
| suggested_scale | Integer scalar scale factor (e.g., ‘100L‘) or ‘NA‘ to clear. |

### Value

The modified ‘klimo_spec‘ object.

## klimo_spec_set_units *Update units in a Klimo spec (post-hoc)*

### Description

Convenience helper to modify the units for one or more columns after building a spec. This does *not* convert any data; it just edits the spec.

### Usage

```
klimo_spec_set_units(spec, columns, metric_units = NULL, imperial_units = NULL)
```

### Arguments

| | |
|---|---|
| spec | A 'klimo_spec' object. |
| columns | Character vector of column names to edit. Matching is case-insensitive. |
| metric_units, imperial_units | Replacement units (character scalar). |

### Value

The modified 'klimo_spec'.

## klimo_write_parquet *Write a Parquet file with Klimo spec metadata (units + scaling + audit)*

### Description

'klimo_write_parquet()' writes a dataset to a Parquet file (via 'arrow') and stores a robust specification + audit trail in Parquet key_value_metadata.

### Usage

```
klimo_write_parquet(
  x,
  path,
  field_spec = NULL,
  declared_system = c("metric", "imperial"),
  unit_mode = c("validate", "none", "convert"),
  unit_strict = FALSE,
  src_units = NULL,
  scaled = FALSE,
  scaled_suffix = "_i",
  drop_original = TRUE,
  compression = "zstd",
  tag_missing_units = FALSE,
  tag_missing_units_warn = TRUE,
  tag_missing_units_max_show = 12L,
  ...
)
```

## Arguments

| | |
|---|---|
| x | A data.frame/data.table. Will be converted to data.table internally. |
| path | Output parquet path. |
| field_spec | NULL, ‘"metric"‘, ‘"imperial"‘, a ‘klimo_spec‘, or a spec ‘data.table‘. If NULL: writes without a spec and emits a message showing how to attach defaults. |
| unit_mode | ‘"validate"‘ (default), ‘"none"‘, or ‘"convert"‘. |
| unit_strict | If TRUE, validation errors stop instead of warning. For convert mode, this also controls whether missing sources stop. |
| src_units | Named character vector of source units for untagged columns (convert mode only), e.g., ‘c(ta="degF", td="degF")‘. |
| scaled | Logical. If TRUE, apply scaling as described above. |
| scaled_suffix | Suffix for scaled columns, default ‘"_i"‘. |
| drop_original | If TRUE, drop original numeric columns that were scaled. |
| compression | Parquet compression codec; passed to ‘arrow::write_parquet‘. |
| ... | Additional arguments passed to ‘arrow::write_parquet‘. |
| system | If ‘field_spec‘ is custom, which units column to treat as declared units (‘"metric"‘ or ‘"imperial"‘). Ignored if ‘field_spec‘ is ‘"metric"‘/‘"imperial"‘ or a ‘klimo_spec‘ with ‘system‘ already set. |

## Details

It implements a hybrid unit strategy:

**Unit modes** - ‘unit_mode = "none"‘: write without validating or converting; metadata still includes declared units (if spec present) and observed units (if attributes exist). - ‘unit_mode = "validate"‘ (recommended default): does not convert values; it checks ‘attr(x[[col]], "unit")‘ against declared units and warns (or errors if strict). - ‘unit_mode = "convert"‘: converts numeric columns to declared units using ‘unit<-‘ (from this package) and then writes. If the column has no ‘attr(unit)‘, you must provide ‘src_units‘ (named vector), otherwise conversion is refused.

**Scaling** - If ‘scaled = TRUE‘ and a column has ‘suggested_scale‘, the writer will create a scaled integer column named ‘paste0(column, scaled_suffix)‘ (default: ‘"_i"‘). Optionally drops the original.

**Metadata keys written** - ‘klimo:spec_version‘, ‘klimo:spec_system‘ - ‘klimo:field_spec_json‘ (JSON; spec table) - ‘klimo:audit_json‘ (JSON; declared vs observed units + status + scaling status)

#’ Key simplification: - There is no separate ‘system‘ argument anymore. - The declared system comes from the spec (klimo_spec) or from ‘declared_system‘ when ‘field_spec‘ is provided as a plain data.table/data.frame.

## Value

Invisibly returns a list with: - ‘path‘ - ‘spec‘ (klimo_spec or NULL) - ‘preview‘ (data.table) - ‘audit‘ (data.table)

## Examples

```
# Minimal example (document-only, no conversion, no scaling):
library(data.table)
dat <- data.table(ta = c(20, 21), rh = c(50, 55), lat = c(35.9, 36.0), lon = c(-78.9, -78.8))
klimo_write_parquet(dat, tempfile(fileext = ".parquet"), field_spec = "metric", unit_mode = "none", scaled = FA
```

```
# Convert-on-write (requires unit() tagging or src_units):
# unit(dat$ta) <- "degF"  # if already known
```

---

`listS3files` *Robust S3 object listing (no ‘Owner‘ assumption) with retries*

---

## Description

Lists objects from an S3 bucket/prefix using **aws.s3**, avoiding the ‘Owner$ID‘ assumption that can trigger ‘subscript out of bounds‘ when ‘Owner‘ is omitted. Adds retries and (optionally) sorts by ‘LastModified‘ and returns only the newest rows.

## Usage

```
listS3files(
  bucket,
  prefix = NULL,
  delimiter = NULL,
  max = NULL,
  marker = NULL,
  ...,
  max_tries = 3,
  wait = 1,
  backoff = 1.6,
  verbose = TRUE,
  order_by_last_modified = FALSE,
  return_tail = FALSE,
  tail_n = 6
)
```

## Arguments

| | |
|---|---|
| `bucket` | Character. S3 bucket name. |
| `prefix` | Character, optional. Limit results to keys beginning with this prefix. |
| `delimiter` | Character, optional. E.g., ‘"/"‘ to group by common prefixes. |
| `max` | Integer, optional. Maximum number of keys to return (per request). |
| `marker` | Character, optional. Key to start with when listing. |
| `...` | Passed through to [aws.s3::get_bucket()] (e.g., credentials, region). |
| `max_tries` | Integer. Maximum attempts before failing. Default ‘3‘. |
| `wait` | Numeric (seconds). Initial wait between attempts. Default ‘1‘. |
| `backoff` | Numeric multiplier for exponential backoff. Default ‘1.6‘. |
| `verbose` | Logical. If ‘TRUE‘, prints retry messages. Default ‘TRUE‘. |
| `order_by_last_modified` | Logical. If ‘TRUE‘, sort ascending by ‘LastModified‘. Default ‘FALSE‘. |
| `return_tail` | Logical. If ‘TRUE‘, return only the last ‘tail_n‘ rows *after* optional sorting. Default ‘FALSE‘. |
| `tail_n` | Integer. Number of rows to return when ‘return_tail = TRUE‘. Default ‘6‘. |

## Details

Calls [aws.s3::get_bucket()] with ‘parse_response = TRUE‘ and converts the parsed list into a data frame while ignoring any ‘Owner‘ fields. Handles "folder" entries by falling back to ‘Name‘ or ‘Prefix‘ when ‘Key‘ is absent. Timestamps are parsed to UTC when present.

Note: this issues a single ListObjects request (no pagination). For >1,000 keys, loop with ‘marker‘/continuation tokens.

## Value

A ‘data.frame‘ with columns: - ‘Key‘ (character) - ‘Size‘ (numeric; ‘NA‘ for common prefixes) - ‘LastModified‘ (POSIXct, UTC; ‘NA‘ if absent) - ‘StorageClass‘ (character; ‘NA‘ if absent)

## See Also

[aws.s3::get_bucket()], [aws.s3::save_object()]

## Examples

```
## Not run:
# Basic
df <- safe_get_bucket_df("my-bucket", prefix = "hrrr/20250809/")

# Newest few objects (sorted by LastModified, tail 6)
newest <- safe_get_bucket_df("my-bucket", prefix = "hrrr/20250809/",
                             order_by_last_modified = TRUE, return_tail = TRUE, tail_n = 6)

## End(Not run)
```

---

# mavtime *Calculate Rolling Averages for Time-Series Data*

---

## Description

The ‘mavtime‘ function calculates rolling averages over a specified time window for time-series data. This implementation leverages an optimized Rcpp function for enhanced performance on large datasets.

## Usage

```
mavtime(
  timevals,
  datavals,
  timewindow,
  timeunits = "min",
  resample = NULL,
  resampleinterval = "1 min"
)
```

### Arguments

| | |
|---|---|
| timevals | A 'POSIXct' vector of timestamps for the data points. |
| datavals | A numeric vector of data values corresponding to the timestamps. |
| timewindow | An integer specifying the size of the rolling window in the units defined by 'timeunits'. |
| timeunits | A character string specifying the time units for the rolling window. Supported values are: '"min"' (default), '"hour"', or '"day"'. |
| resample | A logical indicating whether to resample the data to regular intervals (default: 'TRUE'). |
| resampleinterval | A character string specifying the interval for resampling, e.g., '"1 min"'. |

### Details

This function resamples the data to regular intervals if 'resample = TRUE', ensuring consistent time-series alignment before calculating rolling averages. The rolling averages are computed using a highly optimized Rcpp implementation for performance.

### Value

A numeric vector of rolling averages aligned with the input 'timevals'. Missing values ('NA') are returned if insufficient data exists within the window.

### See Also

'rollapply' from the zoo package for a pure R implementation.

### Examples

```
library(data.table)

# Generate example time-series data
timevals <- seq(from = as.POSIXct("2024-12-01 00:00:00"), by = "1 min", length.out = 100)
datavals <- runif(100, min = 0, max = 100)

# Calculate rolling averages with a 15-minute window
result <- mavtime(timevals, datavals, timewindow = 15, timeunits = "min")
```

---

## mergeDTlist *Merge a list of data.frames/data.tables by keys*

---

### Description

Iteratively merges a list of tabular objects using 'merge(..., all = TRUE)'.

### Usage

```
mergeDTlist(alist, by)
```

### Arguments

| | |
|---|---|
| alist | List of data.frames / data.tables. |
| by | Character vector of key columns passed to ‘merge()‘. |

### Value

A merged data.frame/data.table (depends on inputs).

### Examples

```
a <- data.frame(id = 1:3, x = 11:13)
b <- data.frame(id = 2:4, y = 21:23)
mergeDTlist(list(a, b), by = "id")
```

---

## repvar *Repeat a scalar to length n (or recycle shorter vectors)*

---

### Description

Utility for aligning scalar parameters with vectorized data.

### Usage

```
repvar(var, n)
```

### Arguments

| | |
|---|---|
| var | A vector or ‘NULL‘. |
| n | Integer target length. |

### Value

‘NULL‘ if ‘var‘ is ‘NULL‘; otherwise a vector of length ‘n‘ (or the original vector if already length ‘n‘ or longer). If ‘length(var) < n‘, the value is recycled with a warning.

### Examples

```
repvar(NULL, 5)
repvar(3, 5)
repvar(1:2, 5)
```

## requireRAM *Block Execution Until Available RAM Exceeds Threshold*

### Description

requireRAM pauses execution in a loop until the system reports at least the specified amount of free RAM. Useful for long-running data-intensive tasks that require guaranteed memory availability before proceeding.

### Usage

```
requireRAM(threshold = 4000, unit = "MB", wait_time = 10, max_attempts = 10)
```

### Arguments

| | |
|---|---|
| threshold | Numeric. Minimum amount of free memory required (default 4000). |
| unit | Character. Unit for memory threshold (e.g., "MB" or "GB"). |
| wait_time | Numeric. Seconds to wait between consecutive checks (default 10). |
| max_attempts | Integer. Maximum number of check attempts before giving up (default 10). |

### Details

This function relies on check_ram() to query system memory. It will repeatedly sleep for wait_time seconds and retry until the available RAM is sufficient or the maximum number of attempts is reached.

### Value

Invisibly returns TRUE once the available RAM meets or exceeds the threshold.

### Examples

```
## Not run:
# Wait for at least 4 GB free memory before starting heavy computation
requireRAM(threshold = 4000, unit = "MB", wait_time = 10, max_attempts = 20)

## End(Not run)
```

## roundcols *Round selected numeric columns*

### Description

Rounds specified columns in a data.frame or data.table.

### Usage

```
roundcols(x, cols, digits = 2, in_place = FALSE)
```

### Arguments

| | |
|---|---|
| x | A data.frame or data.table. |
| cols | Character vector of column names to round. |
| digits | Integer. Number of decimal places passed to ‘round()‘. |
| in_place | Logical. If ‘TRUE‘ and ‘x‘ is a data.table, modifies ‘x‘ by reference. Otherwise returns a modified copy. |

### Value

The modified object (invisibly returns ‘x‘ when ‘in_place = TRUE‘).

### Examples

```
df <- data.frame(a = 1:5/2.1, b = 6:10/1.59)
roundcols(df, cols = c("a","b"), digits = 2)

## Not run:
library(data.table)
dt <- data.table(a = 1:5/2.1, b = 6:10/1.59)
roundcols(dt, c("a","b"), 2, in_place = TRUE)

## End(Not run)
```

---

## s3_create_dir *Create an "S3 directory" placeholder*

---

### Description

S3 does not have real directories; prefixes ending in ‘/‘ are treated like folders by many tools. This function checks whether any objects already exist under a prefix and, if not, creates a zero-byte placeholder object at that prefix.

### Usage

```
s3_create_dir(bucket_name, directory_name)
```

### Arguments

| | |
|---|---|
| bucket_name | Character scalar. S3 bucket name. |
| directory_name | Character scalar. Prefix to create. A trailing ‘/‘ is added if missing. |

### Value

Invisibly returns ‘TRUE‘ if a placeholder object was created, ‘FALSE‘ if the prefix already had objects.

### Examples

```
## Not run:
s3_create_dir("my-bucket", "some/prefix")
s3_create_dir("my-bucket", "some/prefix/")  # equivalent

## End(Not run)
```

---

## s3_download_file *Download an R object from S3 (stored as RDS)*

---

### Description

Downloads an object from S3 to a temporary file and reads it with ‘readRDS()‘.

### Usage

```
s3_download_file(key, bucket, fileext = ".rds")
```

### Arguments

| | |
|---|---|
| key | Character scalar. S3 object key (path) to download. |
| bucket | Character scalar. S3 bucket name. |
| fileext | Character scalar. Temporary file extension. Defaults to ‘.rds‘. |

### Value

The R object stored in the RDS.

### Examples

```
## Not run:
x <- s3_download_file("path/to/object.rds", bucket = "my-bucket")

## End(Not run)
```

---

## s3_list_objects *List S3 objects with retries and optional ordering*

---

### Description

Wrapper around ‘aws.s3::get_bucket()‘ that:

- supports basic pagination via ‘marker‘
- retries transient failures with backoff
- returns a data frame with ‘Key‘, ‘Size‘, ‘LastModified‘, ‘StorageClass‘
- can order by ‘LastModified‘ and/or return only the tail rows

## Usage

```
s3_list_objects(
  bucket,
  prefix = NULL,
  delimiter = NULL,
  max = NULL,
  marker = NULL,
  ...,
  max_tries = 3,
  wait = 1,
  backoff = 1.6,
  verbose = TRUE,
  order_by_last_modified = FALSE,
  return_tail = FALSE,
  tail_n = 6
)
```

## Arguments

| | |
|---|---|
| bucket | Character scalar. Bucket name. |
| prefix | Character scalar or ‘NULL‘. Prefix filter. If ‘NULL‘, defaults to ‘file.path("goes", <UTC-YYYY-mm-dd>)‘ (preserves original behavior). |
| delimiter | Character scalar or ‘NULL‘. Passed to S3 listing. |
| max | Integer or ‘NULL‘. Max number of keys to return (approximate). |
| marker | Character scalar or ‘NULL‘. Marker for pagination. |
| ... | Additional args passed to ‘aws.s3::get_bucket()‘. |
| max_tries | Integer. Max attempts per page. |
| wait | Numeric. Base wait time (seconds) between retries. |
| backoff | Numeric. Multiplier applied to wait after each retry. |
| verbose | Logical. If ‘TRUE‘, prints retry messages. |
| order_by_last_modified | Logical. If ‘TRUE‘, sorts by ‘LastModified‘. |
| return_tail | Logical. If ‘TRUE‘, returns only the last ‘tail_n‘ rows after optional ordering. |
| tail_n | Integer. Rows to return when ‘return_tail = TRUE‘. |

## Value

A data.frame with columns: ‘Key‘, ‘Size‘, ‘LastModified‘, ‘StorageClass‘.

## Examples

```
## Not run:
df <- s3_list_objects(bucket = "my-bucket", prefix = "data/2026-01-01/")
newest <- s3_list_objects("my-bucket", prefix = "data/", order_by_last_modified = TRUE, return_tail = TRUE, ta

## End(Not run)
```

## s3_upload_file *Save an R object to S3 (as RDS)*

### Description

Serializes an R object via ‘saveRDS()‘ into a temporary file, uploads it to S3, then removes the temporary file.

### Usage

```
s3_upload_file(object, bucket, s3_path, temp_dir = tempdir())
```

### Arguments

| | |
|---|---|
| object | R object to serialize. |
| bucket | Character scalar. Bucket name. |
| s3_path | Character scalar. Destination S3 key (path). |
| temp_dir | Character scalar. Directory to place the temporary file. |

### Value

The result returned by ‘aws.s3::put_object()‘.

### Examples

```
## Not run:
res <- s3_upload_file(list(a = 1), bucket = "my-bucket", s3_path = "tmp/test.rds")

## End(Not run)
```

## sanitize_value *Sanitize a scalar value for SQL-like string construction*

### Description

Converts a single value into a form often used when building SQL statements:

- ‘NA‘ -> ‘"NULL"‘ (character)
- logical -> ‘0/1‘ (integer)
- character/Date/POSIXct -> single-quoted, with internal ‘’‘ escaped
- numeric -> returned as-is

### Usage

```
sanitize_value(x)
```

### Arguments

| | |
|---|---|
| x | A length-1 value. |

## Value

A length-1 value suitable for pasting into query text.

## Examples

```
sanitize_value(NA)
sanitize_value(TRUE)
sanitize_value("O'Reilly")
sanitize_value(as.Date("2026-01-12"))
sanitize_value(3.14)
```

---

`startEnvironment` *Start a Parallel Cluster with a Cross-Platform Watchdog (Temp Stop File)*

---

## Description

'startEnvironment()' sets up a parallel cluster and starts a watchdog script that monitors memory usage on Windows or Linux. It writes out default watchdog scripts if needed.

If 'stop_file' is not provided, a unique temporary filename (with random characters) is generated.

## Usage

```
startEnvironment(
  num_cores = 2,
  threshold = 0.99,
  consecutive_seconds = 15,
  stop_file = tempfile("stop_watchdog_"),
  windows_path = "C:\\Users\\Jordan\\Desktop",
  linux_path = "~/scripts"
)
```

## Arguments

| | |
|---|---|
| `num_cores` | Integer. Number of cores to use for the parallel cluster. Defaults to 2. |
| `threshold` | Numeric. The memory usage threshold (0 to 1) at which the watchdog triggers. Default 0.99. |
| `consecutive_seconds` | Integer. How many consecutive seconds memory must be above threshold before killing. Default 15. |
| `stop_file` | Character. The full path and filename of the stop file to be used by the watchdog. If not specified, a random unique temp file name is created. |
| `windows_path` | Character. The directory for 'watchdog2.ps1' on Windows. Defaults to '"C:\Users\Jordan\Desktop"'. |
| `linux_path` | Character. The directory for 'watchdog2.sh' on Linux. Defaults to '"~/scripts"'. |

### Details

- On Windows, uses a PowerShell script (‘watchdog2.ps1‘). - On Linux, uses a Bash script (‘watchdog2.sh‘). - If the watchdog script doesn’t exist, it is written out. - If ‘stop_file‘ is ‘NULL‘, a unique temp file is created automatically.

The watchdog monitors memory usage. If it stays above ‘threshold‘ for ‘consecutive_seconds‘, it kills the provided worker PIDs. To stop the watchdog gracefully, create the stop file specified by ‘stop_file‘.

### Value

A list: ‘list(cluster = cl, stop_file = stop_file, worker_pids = pids)‘.

### Examples

```
## Not run:
# Start environment with a random unique stop file name:
env_info <- startEnvironment()

# ... do some computations ...

# Gracefully stop watchdog:
# ?jj::closeEnvironment
closeEnvironment(env_info)
file.create(env_info$stop_file)
jstopCluster(env_info$worker_pids)
# file.create(env_info$stop_file)
# Kill processes if desired:
jstopCluster(env_info$worker_pids)
# Then stop cluster (probably DO NOT need to also do this after calling jstopCluster():
parallel::stopCluster(env_info$cluster)

## End(Not run)
```

---

## syn_create_file_entity

*Create a FileEntity under a parent container*

---

### Description

Typically used after creating a file handle via multipart upload.

### Usage

```
syn_create_file_entity(
  parent_id,
  name,
  dataFileHandleId,
  description = NULL,
  token = NULL,
  verbose = FALSE,
  dry_run = FALSE
)
```

### Arguments

| | |
|---|---|
| parent_id | Parent Synapse ID (Project or Folder). |
| name | FileEntity name. |
| dataFileHandleId | File handle ID returned by multipart upload completion. |
| description | Optional description string. |
| token | Optional token (PAT). |
| verbose | Logical. |
| dry_run | Logical. |

### Value

Parsed JSON list for created entity; contains ‘id‘.

### Examples

```
## Not run:
fe <- syn_create_file_entity("synFolderId", "data.csv", dataFileHandleId = "123456")
fe$id

## End(Not run)
```

---

## syn_create_folder *Create a Folder under a parent container*

---

### Description

Create a Folder under a parent container

### Usage

```
syn_create_folder(
  parent_id,
  name,
  description = NULL,
  token = NULL,
  verbose = FALSE,
  dry_run = FALSE
)
```

### Arguments

| | |
|---|---|
| parent_id | Parent Synapse ID (Project or Folder). |
| name | Folder name (single path segment). |
| description | Optional description string. |
| token | Optional token (PAT). |
| verbose | Logical. |
| dry_run | Logical. |

### Value

Parsed JSON list for created entity; contains ‘id‘.

### Examples

```
## Not run:
f <- syn_create_folder("synProjectId", "my_folder", description = "Created via REST")
f$id

## End(Not run)
```

---

## syn_download_file *Download a Synapse FileEntity to disk*

---

### Description

Fetches the FileEntity to obtain ‘dataFileHandleId‘, resolves a presigned URL via the file handle URL endpoint, and streams the file content to ‘dest_path‘.

### Usage

```
syn_download_file(
  entity_id,
  dest_path,
  overwrite = FALSE,
  token = NULL,
  verbose = FALSE,
  dry_run = FALSE
)
```

### Arguments

| | |
|---|---|
| entity_id | FileEntity synId. |
| dest_path | Destination path (will create parent directories). |
| overwrite | Logical; overwrite existing file if ‘TRUE‘. |
| token | Optional token (PAT). |
| verbose | Logical. |
| dry_run | Logical. |

### Value

‘dest_path‘ (invisibly in dry-run, otherwise returned as a character scalar).

### Examples

```
## Not run:
dest <- tempfile(fileext = ".csv")
syn_download_file("synFileEntityId", dest, overwrite = TRUE, verbose = TRUE)
read.csv(dest)

## End(Not run)
```

`syn_download_file_path` *Download a file using a remote path (relative to a base container)*

## Description

Download a file using a remote path (relative to a base container)

## Usage

```
syn_download_file_path(
  base_id,
  remote_file_path,
  dest_path,
  overwrite = FALSE,
  token = NULL,
  verbose = FALSE,
  dry_run = FALSE
)
```

## Arguments

| | |
|---|---|
| `base_id` | Base container synId. |
| `remote_file_path` | Remote file path like ‘"a/b/c/file.csv"‘. |
| `dest_path` | Local destination path. |
| `overwrite` | Logical; overwrite existing local file. |
| `token` | Optional token (PAT). |
| `verbose` | Logical. |
| `dry_run` | Logical. |

## Value

‘dest_path‘.

## Examples

```
## Not run:
syn_download_file_path(
  base_id = "synProjectId",
  remote_file_path = "zz_smoketests/synapse_rw/test2/New Text Document.txt",
  dest_path = "C:/tmp/downloads/test.txt",
  overwrite = TRUE,
  verbose = TRUE
)

## End(Not run)
```

## syn_ensure_folder_path *Ensure a folder path exists under a base container*

### Description

Ensures each segment exists as a Folder under ‘parent_id‘. Missing folders are created. Existing segments are validated as Folders (not files) by calling [syn_get_entity()] on each resolved synId.

### Usage

```
syn_ensure_folder_path(
  parent_id,
  path,
  token = NULL,
  verbose = FALSE,
  dry_run = FALSE
)
```

### Arguments

| | |
|---|---|
| parent_id | Base container synId (Project or Folder). |
| path | Remote folder path like ‘"a/b/c"‘ (slashes only; leading/trailing ignored). |
| token | Optional token (PAT). |
| verbose | Logical. |
| dry_run | Logical. |

### Value

synId of the deepest folder in the path.

### Examples

```
## Not run:
folder_id <- syn_ensure_folder_path("synProjectId", "analysis/run_01")
folder_id

## End(Not run)
```

## syn_get_annotations *Get annotations2 for an entity*

### Description

Get annotations2 for an entity

### Usage

```
syn_get_annotations(entity_id, token = NULL, verbose = FALSE, dry_run = FALSE)
```

### Arguments

| | |
|---|---|
| entity_id | synId. |
| token | Optional token (PAT). |
| verbose | Logical. |
| dry_run | Logical. |

### Value

Parsed JSON list containing ‘etag‘ and ‘annotations‘.

### Examples

```
## Not run:
ann <- syn_get_annotations("synEntityId")
names(ann$annotations)

## End(Not run)
```

---

## syn_get_child_by_name *Get a child entity by name under a parent container*

---

### Description

Wraps ‘POST /repo/v1/entity/child‘.

### Usage

```
syn_get_child_by_name(
  parent_id,
  name,
  include_types = NULL,
  token = NULL,
  verbose = FALSE,
  dry_run = FALSE
)
```

### Arguments

| | |
|---|---|
| parent_id | Parent synId. |
| name | Child name. |
| include_types | Optional vector of type filters (e.g., ‘"folder"‘, ‘"file"‘). |
| token | Optional token (PAT). |
| verbose | Logical. |
| dry_run | Logical. |

### Value

Parsed JSON list including ‘id‘ and (depending on response) type metadata.

## Examples

```
## Not run:
hit <- syn_get_child_by_name("synParent", "my_folder", include_types = "folder")
hit$id

## End(Not run)
```

---

`syn_get_entity` *Get a Synapse entity*

---

## Description

Get a Synapse entity

## Usage

```
syn_get_entity(entity_id, token = NULL, verbose = FALSE, dry_run = FALSE)
```

## Arguments

| | |
|---|---|
| `entity_id` | Synapse ID (e.g., ‘"syn123"‘). |
| `token` | Optional token (PAT). |
| `verbose` | Logical; emit request progress messages. |
| `dry_run` | Logical; do not execute. |

## Value

Parsed JSON list describing the entity.

## Examples

```
## Not run:
e <- syn_get_entity("syn123")
e$concreteType

## End(Not run)
```

---

`syn_lookup_child_id` *Lookup a child synId by parent + name*

---

## Description

Convenience wrapper returning just the child ‘id‘ or ‘NULL‘ if not found.

### Usage

```
syn_lookup_child_id(
  parent_id,
  entity_name,
  token = NULL,
  verbose = FALSE,
  dry_run = FALSE
)
```

### Arguments

| | |
|---|---|
| parent_id | Parent synId. |
| entity_name | Child name. |
| token | Optional token (PAT). |
| verbose | Logical. |
| dry_run | Logical. |

### Value

Child synId character scalar, or ‘NULL‘.

### Examples

```
## Not run:
cid <- syn_lookup_child_id("synParent", "child_name")

## End(Not run)
```

---

## syn_request *Perform a Synapse REST request*

---

### Description

Low-level HTTP wrapper around Synapse endpoints using **httr2**.

### Usage

```
syn_request(
  method,
  path,
  query = NULL,
  body = NULL,
  body_json = TRUE,
  headers = list(),
  token = NULL,
  retry = TRUE,
  verbose = FALSE,
  dry_run = FALSE
)
```

### Arguments

| | |
|---|---|
| method | HTTP method (e.g., '"GET"', '"POST"', '"PUT"'). |
| path | Path starting with '/' (e.g., '"/repo/v1/entity/syn123"'). |
| query | Optional named list of query parameters. |
| body | Optional request body. |
| body_json | If 'TRUE', encode body as JSON and set content-type accordingly. |
| headers | Optional named list of extra headers. |
| token | Optional Synapse token (PAT). If 'NULL', env vars are used. |
| retry | Logical; retry transient failures when 'TRUE'. |
| verbose | Logical; emit request/response progress messages. |
| dry_run | Logical; do not execute the request, return a description instead. |

### Details

- Adds Authorization header using a PAT (see [syn_resolve_token()]). - Retries transient failures (network, HTTP 429, HTTP 5xx) with exponential backoff. - Parses JSON responses when content-type indicates JSON; otherwise returns text.

### Value

For JSON: a list (parsed JSON). For non-JSON: a character string body.

### Examples

```
## Not run:
tok <- syn_resolve_token()
ent <- syn_request("GET", "/repo/v1/entity/syn123", token = tok)
str(ent)

## End(Not run)
```

---

## syn_resolve_entity_by_path

*Resolve an entity synId from a remote path under a base container*

---

### Description

- If 'remote_path' ends with '/', the last segment must resolve to a Folder. - Otherwise the last segment may resolve to either a Folder or FileEntity.

### Usage

```
syn_resolve_entity_by_path(
  base_id,
  remote_path,
  token = NULL,
  verbose = FALSE,
  dry_run = FALSE
)
```

### Arguments

| | |
|---|---|
| base_id | Base container synId (Project or Folder). |
| remote_path | Remote path like ‘"a/b/file.csv"‘ or ‘"a/b/folder/"‘. |
| token | Optional token (PAT). |
| verbose | Logical. |
| dry_run | Logical. |

### Details

Folder-ness checks are enforced by calling [syn_get_entity()] on intermediate segments.

### Value

synId of the resolved entity.

### Examples

```
## Not run:
eid <- syn_resolve_entity_by_path("synProjectId", "a/b/c/myfile.csv")
eid

## End(Not run)
```

---

## syn_resolve_token *Resolve a Synapse Personal Access Token (PAT)*

---

### Description

Returns a token from (in order): 1) ‘token‘ argument (if a long-ish scalar string) 2) ‘Sys.getenv("SYNAPSE_PAT")‘ (preferred) 3) ‘Sys.getenv("SYNAPSE_AUTH_TOKEN")‘ (fallback)

### Usage

```
syn_resolve_token(token = NULL)
```

### Arguments

| | |
|---|---|
| token | Optional token string. |

### Value

A character scalar token.

### Examples

```
## Not run:
Sys.setenv(SYNAPSE_PAT = "YOUR_PAT")
tok <- syn_resolve_token()

## End(Not run)
```

## syn_set_annotations *Set (merge) annotations2 on an entity*

### Description

Fetches current annotations2, merges keys provided in 'annotations' (overwriting those keys), and PUTs the updated annotations back using the current 'etag'.

### Usage

```
syn_set_annotations(
  entity_id,
  annotations,
  token = NULL,
  verbose = FALSE,
  dry_run = FALSE
)
```

### Arguments

| | |
|---|---|
| entity_id | synId. |
| annotations | Named list of annotations. Values may be scalar or vectors. |
| token | Optional token (PAT). |
| verbose | Logical. |
| dry_run | Logical. |

### Value

Parsed JSON list from the PUT request.

### Examples

```
## Not run:
syn_set_annotations("synEntityId", list(stage = "raw", tags = c("a","b")))

## End(Not run)
```

## syn_try_get_child_by_name *Try to get a child entity by name (returns NULL on 404)*

### Description

Try to get a child entity by name (returns NULL on 404)

### Usage

```
syn_try_get_child_by_name(
  parent_id,
  name,
  include_types = NULL,
  token = NULL,
  verbose = FALSE,
  dry_run = FALSE
)
```

### Arguments

| | |
|---|---|
| parent_id | Parent synId. |
| name | Child name. |
| include_types | Optional vector of type filters (e.g., ‘"folder"‘, ‘"file"‘). |
| token | Optional token (PAT). |
| verbose | Logical. |
| dry_run | Logical. |

### Value

Parsed JSON list, or ‘NULL‘ if not found.

### Examples

```
## Not run:
hit <- syn_try_get_child_by_name("synParent", "maybe_exists", include_types = c("folder","file"))
if (is.null(hit)) message("Not found")

## End(Not run)
```

---

## syn_update_file_entity

*Update a FileEntity to point to a new file handle*

---

### Description

Update a FileEntity to point to a new file handle

### Usage

```
syn_update_file_entity(
  entity_id,
  new_dataFileHandleId,
  token = NULL,
  verbose = FALSE,
  dry_run = FALSE
)
```

### Arguments

| | |
|---|---|
| entity_id | FileEntity synId. |
| new_dataFileHandleId | New file handle ID. |
| token | Optional token (PAT). |
| verbose | Logical. |
| dry_run | Logical. |

### Value

Parsed JSON list for updated entity.

### Examples

```
## Not run:
syn_update_file_entity("synFileEntityId", new_dataFileHandleId = "999999")

## End(Not run)
```

---

## syn_upload_file *Upload a local file to Synapse (multipart) and create/update a FileEntity*

---

### Description

Implements Synapse multipart upload: 1) Determine upload destination storage location for ‘parent_id‘ 2) Initiate multipart upload 3) Upload parts to presigned S3 URLs and ‘add‘ each part 4) Complete multipart upload to obtain a file handle ID 5) Create a new FileEntity under ‘parent_id‘ or update an existing file with the same name 6) Optionally set annotations (annotations2)

### Usage

```
syn_upload_file(
  local_path,
  parent_id,
  name = NULL,
  contentType = NULL,
  md5 = TRUE,
  overwrite = FALSE,
  description = NULL,
  annotations = NULL,
  token = NULL,
  verbose = FALSE,
  dry_run = FALSE
)
```

### Arguments

| | |
|---|---|
| local_path | Path to an existing local file. |
| parent_id | Parent container synId (Project or Folder). |
| name | Optional name for the FileEntity (defaults to ‘basename(local_path)‘). |
| contentType | Optional MIME type (defaults to [tools::file_ext()] guess). |
| md5 | Logical; compute MD5 for local file and parts (requires **digest**). |
| overwrite | Logical; if a file with the same name exists, update it. |
| description | Optional description for the FileEntity. |
| annotations | Optional named list of annotations (applied via annotations2). |
| token | Optional Synapse token (PAT). |
| verbose | Logical; emit progress. |
| dry_run | Logical; do not execute network requests. |

### Value

An object of class ‘"syn_upload_result"‘ with fields: - ‘file_entity_id‘ - ‘file_handle_id‘ - ‘upload_id‘ - ‘md5_hex‘ (file MD5 if computed) - ‘size_bytes‘ - ‘etags‘ (list of part ETags when available)

### Examples

```
## Not run:
tmp <- tempfile(fileext = ".csv")
write.csv(data.frame(x=1:3, y=letters[1:3]), tmp, row.names = FALSE)

res <- syn_upload_file(
  local_path = tmp,
  parent_id  = "synProjectOrFolder",
  overwrite  = TRUE,
  annotations = list(stage = "raw", kind = "csv"),
  verbose = TRUE
)
res$file_entity_id

## End(Not run)
```

---

## syn_upload_file_path *Upload a file to a remote folder path (relative to a base container)*

---

### Description

Convenience wrapper: - Ensures ‘remote_folder_path‘ exists under ‘base_id‘ (creating folders as needed) - Uploads ‘local_path‘ to the resulting folder via [syn_upload_file()]

## Usage

```
syn_upload_file_path(
  local_path,
  base_id,
  remote_folder_path,
  name = NULL,
  contentType = NULL,
  overwrite = FALSE,
  description = NULL,
  annotations = NULL,
  token = NULL,
  verbose = FALSE,
  dry_run = FALSE
)
```

## Arguments

| | |
|---|---|
| local_path | Local file path. |
| base_id | Base container synId (Project or Folder). |
| remote_folder_path | Remote folder path relative to ‘base_id‘, e.g. ‘"a/b/c"‘. |
| name | Optional FileEntity name override. |
| contentType | Optional MIME type override. |
| overwrite | Logical; overwrite existing FileEntity with same name. |
| description | Optional description. |
| annotations | Optional named list for annotations2. |
| token | Optional token (PAT). |
| verbose | Logical. |
| dry_run | Logical. |

## Value

‘"syn_upload_result"‘ (see [syn_upload_file()]).

## Examples

```
## Not run:
res <- syn_upload_file_path(
  local_path = "C:/data/example.csv",
  base_id = "synProjectId",
  remote_folder_path = "zz_smoketests/synapse_rw/test2",
  overwrite = TRUE,
  annotations = list(stage = "smoke", kind = "csv"),
  verbose = TRUE
)
res$file_entity_id

## End(Not run)
```

## timed *timed: simple script timer*

### Description

Start/stop a single unnamed timer for the whole script. Prints start/finish timestamps and elapsed time. Optionally returns the timestamp/elapsed.

### Usage

```
timed(action, label = NULL, round = 2, ret = FALSE)
```

### Arguments

| | |
|---|---|
| round | Integer(1). Decimal places for rounding elapsed time. Default 2. |
| ret | Logical(1). If TRUE, returns start time (on "start") or elapsed difftime (on "end"/"stop"). Default FALSE. |
| what | One of "start", "end", or "stop". |

### Value

Invisibly NULL by default; if ‘ret = TRUE‘, a POSIXct (start) or difftime (end/stop).

### Examples

```
timed("start"); Sys.sleep(0.2); timed("end")
```

## to_camel_case *Convert strings to lowerCamelCase*

### Description

Converts arbitrary strings to lowerCamelCase by:

1. replacing non-alphanumeric characters with spaces
2. splitting on whitespace
3. lowercasing the first token
4. TitleCasing subsequent tokens
5. concatenating tokens

### Usage

```
to_camel_case(string)
```

### Arguments

| | |
|---|---|
| string | Character vector. |

## Value

Character vector of the same length, converted to lowerCamelCase.

## Examples

```
to_camel_case(c("hello world", "some_value", "  already  camel  "))
```

---

`trimallcols` *Trim whitespace from all character columns*

---

## Description

Applies ‘trimws()‘ to all character columns in a data.frame/data.table.

## Usage

```
trimallcols(data)
```

## Arguments

| | |
|---|---|
| `data` | A data.frame or data.table. |

## Value

The trimmed object (copy).

## Examples

```
d <- data.frame(a = c("  x", "y  "), b = 1:2, stringsAsFactors = FALSE)
trimallcols(d)
```

---

`unit` *Lightweight unit getter/setter (conversion + optional rounding)*

---

## Description

Provides a simple way to read and set a ‘"unit"‘ attribute on numeric vectors. The replacement form (‘unit(x) <- value‘) can also convert values between supported units (handled by the package-internal `.convert_units()`) and optionally round the result.

## Usage

```
unit(x)

unit(x) <- value
```

## Arguments

| | |
|---|---|
| x | A numeric vector (or column) to query or tag with a "unit" attribute. |
| value | Character scalar describing the assignment. See the accepted forms above. Examples: "degF", "degC -> degF", "degC\|degF\|1", "degF (2)", "degF; round=1". |

## Details

**Accepted assignment forms for value:**

- "degF" — set/convert to degF using the current attribute as the source (if present).
- "degC -> degF" — explicitly convert from degC to degF.
- "degC|degF" — shorthand for the arrow form above.
- Optional rounding may be requested (highest precedence first): "...; round=1", "...; digits=1", trailing "|1", or trailing "(1)"; e.g., "degC|degF|1" or "degF (2)".

If an explicit source unit is provided on the left side (e.g., "degC -> degF"), it is treated as authoritative. To prevent accidental re-interpretation of already-tagged data, a source-mismatch policy is applied:

- options(jj.unit.on_src_mismatch = "warn_noop") (default) — warn and do nothing when *declared source* differs from attr(x,"unit").
- "error" — stop with an error.
- "convert" — trust the declared source and convert anyway (original permissive behavior).

Rounding is applied *after* conversion. Global default rounding can be set via options(jj.unit.round_digits = K). This default is only applied when a conversion actually occurred; inline digits (round= / digits= / |K / (K)) always apply. When rounding occurs, the setter also records attr(x, "unit_digits") = K.

To avoid silently mislabeling temperatures, you may forbid "tag-only" temperature assignments (no known source) by enabling: options(jj.unit.disallow_temp_tag_only = TRUE). With this option set, attempting unit(x) <- "degF" on an untagged vector will error; use "src|dst" instead.

## Value

- unit(x) returns the current unit attribute (character scalar) or NULL.
- unit(x) <- value returns the modified vector x, with values converted if needed, optionally rounded, and attr(x, "unit") set to the target unit. When rounding is applied, attr(x, "unit_digits") is also set.

## Conversions

The actual numeric conversions are performed by the internal .convert_units(x, from, to). Typical pairs supported in this package include temperature (degC, degF, K/degK), pressure (hPa, Pa), wind speed (m/s, mph, kt), precipitation depth (kg/m^2, mm, in), radiation (W/m^2), and relative humidity forms (where supported by your map).

## Examples

```
x <- c(25, 26, 27)
unit(x) <- "degC"        # tag as degC
unit(x)                  # "degC"

# Convert using current attribute as the source
unit(x) <- "degF"        # C -> F (if attr is "degC")

# Explicit source -> target
unit(x) <- "degF -> degC"

# Shorthand with rounding
unit(x) <- "degC|degF|1"   # C -> F, then round(., 1)
unit(x) <- "degF (2)"      # tag/convert to F, then round(., 2)

# Safer everyday pattern (idempotent):
y <- c(0, 5, 10); unit(y) <- "degC"
unit(y) <- "degF"          # converts once; calling again is a no-op

# data.table in-place usage
if (requireNamespace("data.table", quietly = TRUE)) {
  library(data.table)
  DT <- data.table(ta = c(25, 26, 27))
  unit(DT$ta) <- "degC"
  DT[, ta := { unit(ta) <- "degC -> degF; round=1"; ta }]
  unit(DT$ta)                    # "degF"
  attr(DT$ta, "unit_digits")     # 1
}

# Policies (optional):
# options(jj.unit.on_src_mismatch = "warn_noop")  # default
# options(jj.unit.on_src_mismatch = "error")
# options(jj.unit.on_src_mismatch = "convert")
#
# options(jj.unit.round_digits = 1L)              # global rounding (after conversions)
# options(jj.unit.disallow_temp_tag_only = TRUE)  # forbid tag-only for temperatures
```

# Index