Package 'lineager'

August 20, 2026

Title Row-Level Data Provenance and Exclusion Tracking

Version 0.1.1

Description Provides row-level data provenance tracking for analytical pipelines. Tags datasets with unique lineage identifiers that persist through filter, join, and derive operations. Requires documented reasons for every row exclusion, capturing who was removed, why, and at which pipeline stage. Variable derivations are registered as structured specifications linking output variables back to their source. Any row in any downstream dataset can be traced back to its origin via lg_trace(). Generates structured HTML provenance reports suitable for regulatory submissions, internal audit, or analytical documentation. General-purpose: works for clinical data, machine learning pipelines, financial modelling, epidemiology, or any workflow where row-level accountability matters. Optional features support pharmaceutical users including population flag definitions, source-to-analysis variable mapping, and Reviewer's Guide-aligned report output. Complements the 'regulog' package for tamper-evident session-level audit logging. For more details see <https://reprostats.org/lineager/>.

License MIT + file LICENSE

URL https://reprostats.org, https://github.com/repro-stats/lineager

BugReports https://github.com/repro-stats/lineager/issues

Imports dplyr (>= 1.1.0), magrittr (>= 2.0.3)

Suggests admiral, DiagrammeR, ggplot2, haven, knitr, mockery, rmarkdown, testthat (>= 3.0.0), tibble

Encoding UTF-8

Language en-GB

RoxygenNote 7.3.3

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Ndoh Penn [aut, cre] (ORCID: <https://orcid.org/0009-0003-9054-465X>)

Maintainer Ndoh Penn <ndohpenn9@gmail.com>

Repository CRAN

Date/Publication 2026-08-20 14:20:20 UTC

Contents

`lineager-package` *lineager: Row-Level Data Provenance and Exclusion Tracking*

Description

You build a dataset. You filter it, join it, derive new variables, and produce an analysis. Somewhere along the way rows disappear : subjects excluded, records removed, observations dropped. Later, someone asks: *"Show me exactly which records were excluded, why, and what happened to record 01-042 between the source data and this analysis."*

`lineager` makes that question answerable : programmatically, from your existing R pipeline, with no post-hoc documentation.

It tags every row of every dataset with a unique lineage identifier that survives filters, joins, and derivations. Every row removal must carry a documented reason. Variable derivations are registered as structured specifications. And at any point, `lg_trace()` returns any row's complete journey across the entire pipeline : from source to final analysis dataset.

`lineager` is general-purpose. It works for any R pipeline where row-level provenance matters: clinical data, machine learning, financial modelling, epidemiology, or any analytical workflow where

"what was excluded and why" is a question you need to answer. CDISC-specific features (domain codes, population flags, SDTM-to-ADaM variable mapping, Reviewer's Guide output) are available as optional enrichment for pharmaceutical and clinical users.

Workflow

Step 1 : Start a session and tag your source data

```
lg_start(study_id = "TRIAL-001", analysis_id = "primary-efficacy")

# CDISC datasets
dm <- lg_tag(haven::read_sas("sdtm/dm.sas7bdat"),
             dataset_id = "DM", domain = "DM",
             label = "Demographics")

# General-purpose datasets
patients <- lg_tag(patient_df, dataset_id = "patients",
                   label = "Patient registry")
```

Step 2 : Derive variables with documented descriptions

```
adsl <- lg_derive(dm,
  RANDFL = ifelse(ARMCD != "SCRNFAIL", "Y", "N"),
  description = "RANDFL: Y if subject was randomised (ARMCD != 'SCRNFAIL')"
)

lg_spec("ADSL", "RANDFL", "Randomised Flag",
        source_domain = "DM", source_var = "ARMCD",
        derivation    = "Y if ARMCD != 'SCRNFAIL'")
```

Step 3 : Filter with mandatory exclusion reasons

```
adsl_safety <- lg_filter(
  adsl,
  SAFFL == "Y",
  reason      = "Not in safety population (SAFFL != 'Y')",
  reason_code = "NOT_SAFETY",
  population  = "SAFFL"
)
```

Step 4 : Trace any row and generate the provenance report

```
lg_trace("01-042")

lg_report(
  output  = "outputs/provenance_report.html",
  title   = "Data Provenance Report",
  sponsor = "Example Pharma Ltd",
  author  = "J. Smith, Biostatistician"
)
```

Key functions

Function	Purpose
`lg_start()`	Initialise a provenance session
`lg_end()`	End the session and print a summary
`lg_tag()`	Tag a dataset with row-level lineage IDs
`lg_filter()`	Filter with mandatory exclusion reason
`lg_derive()`	Derive new variables with documented description
`lg_join()`	Tracked join with bilateral row-ID tracing
`lg_population()`	Register a population or cohort definition
`lg_spec()`	Document a source-to-analysis variable derivation
`lg_trace()`	Trace a row's complete lineage journey
`lg_history()`	Retrieve the operation history recorded on a tagged object
`lg_exclusions()`	Retrieve the full exclusion registry
`lg_disposition()`	Grouped exclusion summary table
`lg_operations()`	Full pipeline operation log
`lg_lineage()`	Build a pipeline lineage graph from session operations
`lg_plot()`	Render the lineage graph inline or export as DOT
`lg_report()`	Generate a structured HTML provenance report

The lineage ID

Every row in every tagged dataset carries a `lineage_id` column. For datasets with a `USUBJID` column, the ID embeds the subject identifier for human readability:

```
DM_0001_01-042    <- row 1 from DM, subject 01-042
ADLB_0047_01-042  <- row 47 from ADLB, subject 01-042
```

For datasets without `USUBJID`, a zero-padded sequence is used:

```
patients_000001   <- row 1 from the patients dataset
```

This ID persists through `lg_filter()`, `lg_derive()`, and `lg_join()`, forming the traceable thread connecting any output row back to its origin.

CDISC-specific features

Pharmaceutical and clinical users can additionally use:

- `domain` argument in `lg_tag()` for CDISC domain codes (`"DM"`, `"LB"`, `"AE"`, etc.)
- `lg_population()` to register SAFFL, ITTFL, PPROTFL flag definitions
- `lg_spec()` to document SDTM-to-ADaM variable derivations
- `lg_report()` to generate CDISC Reviewer's Guide-aligned documentation

None of these are required for general use.

Integration with regulog

lineager and regulog are complementary packages. Use regulog to create a tamper-evident audit trail of the session (who ran what, when, and why), and lineager to document the row-level data transformations within that session. The lg_report() output can be referenced in the regulog audit trail via log_action().

Author(s)

Maintainer: Ndoh Penn <ndohpenn9@gmail.com> (ORCID)

See Also

Useful links:

- https://reprostats.org
- https://github.com/repro-stats/lineager
- Report bugs at https://github.com/repro-stats/lineager/issues

---

lg_derive *Derive new variables with documented derivation*

Description

Works exactly like dplyr::mutate() but records a derivation description in the session operation log. Use this when computing ADaM analysis variables from SDTM source variables.

Usage

```
lg_derive(data, ..., description)
```

Arguments

data	An lg_df from lg_tag().
...	Name-value pairs of derivations, passed to dplyr::mutate().
description	Character. **Required.** What is being derived and from what source. E.g. "AVAL: numeric conversion of LBORRES; LBSTRESN used where LBORRES is missing or non-numeric".

Value

An lg_df with derived variables added.

See Also

lg_filter(), lg_join(), lg_spec()

Examples

```
lg_start()
lb <- lg_tag(
  data.frame(USUBJID = "01-001", LBORRES = "12.4",
             LBSTRESN = 12.4, stringsAsFactors = FALSE),
  dataset_id = "LB", domain = "LB"
)

lb_derived <- lg_derive(
  lb,
  AVAL = dplyr::coalesce(LBSTRESN, suppressWarnings(as.numeric(LBORRES))),
  description = "AVAL: LBSTRESN; numeric LBORRES where LBSTRESN is missing"
)
```

---

lg_disposition *Generate a subject disposition summary*

Description

Produces a CONSORT-style subject disposition table from every documented exclusion in the active session – both `lg_filter()` calls and any `lg_join()` call (`type = "inner"/"right"`) that dropped unmatched rows of `x`. Both are exclusions in the same sense (rows removed from the pipeline with a mandatory documented reason), so both must be reflected here for the totals to match `lg_exclusions()`.

Usage

```
lg_disposition(by = c("reason", "population", "dataset"))
```

Arguments

by	Character. How to group: `"reason"` (default) returns the exact step-by-step funnel. `"population"` groups by population flag. `"dataset"` groups by dataset ID.

Details

With the default `by = "reason"`, this returns one row **per contributing step** (filter or row-dropping join), in the exact chronological order they were executed, with the number of subjects excluded at that step and the number remaining immediately afterward – i.e. the actual funnel.

`by = "population"` and `by = "dataset"` aggregate exclusions that share a population flag or dataset across possibly multiple steps, in the order each group first appears. Note that `lg_join()` has no `population` argument, so join-caused exclusions always fall into the `"(none)"` group under `by = "population"`.

Value

A data.frame. For by = "reason": columns step, reason, n_excluded, n_remaining. For by = "population" or "dataset": columns group, n_excluded, n_remaining.

See Also

lg_exclusions(), lg_trace()

Examples

```
lg_start()
adsl <- lg_tag(
  data.frame(
    USUBJID = sprintf("%02d", 1:5),
    RANDFL = c("Y","Y","N","Y","Y"),
    SAFFL  = c("Y","Y","N","Y","N")
  ),
  dataset_id = "ADSL5"
)
lg_filter(adsl, RANDFL == "Y",
  reason = "Not randomised (RANDFL != 'Y')",
  reason_code = "NOT_RANDOMISED", population = "RANDFL"
)

lg_disposition()
```

---

lg_end *End a lineager provenance session*

Description

Prints a session summary and marks the session inactive. The store is preserved in memory and remains queryable via lg_trace(), lg_exclusions(), and lg_report() until lg_start() is called again.

Usage

```
lg_end()
```

Value

Invisibly NULL.

See Also

lg_start()

Examples

```
lg_start()
lg_end()
```

`lg_exclusions` *Retrieve the exclusion registry*

Description

Returns all exclusions recorded by `lg_filter()` calls during the active session as a flat `data.frame`. This is the data underlying the subject disposition listing : every excluded subject, with their USUBJID, the reason they were excluded, and which population the exclusion relates to.

Usage

```
lg_exclusions(population = NULL, dataset_id = NULL, verbose = TRUE)
```

Arguments

`population`	Character or `NULL`. Filter to a specific population flag (e.g. `"SAFFL"`). `NULL` returns all exclusions.
`dataset_id`	Character or `NULL`. Filter to a specific dataset.
`verbose`	Logical. If `TRUE` (default), prints a count summary.

Value

A `data.frame` with columns: `excl_id`, `op_id`, `dataset_id`, `lid`, `usubjid`, `reason`, `reason_code`, `population`, `excluded_at`.

See Also

`lg_trace()`, `lg_disposition()`

Examples

```
lg_start()
adsl <- lg_tag(
  data.frame(USUBJID = c("01", "02", "03"), RANDFL = c("Y", "N", "Y")),
  dataset_id = "ADSL"
)
lg_filter(adsl, RANDFL == "Y",
  reason = "Not randomised", population = "RANDFL"
)

lg_exclusions()
```

lg_filter *Filter a tagged dataset with mandatory exclusion documentation*

Description

Works exactly like dplyr::filter() but requires a reason for every exclusion. Rows that do not meet the filter conditions are captured in the session exclusion registry with their USUBJID (if present), lineage ID, and the documented reason.

Usage

```
lg_filter(data, ..., reason, population = NULL, reason_code = NULL)
```

Arguments

data	An lg_df from lg_tag().
...	Filter conditions, passed to dplyr::filter().
reason	Character. **Mandatory.** Why these rows are being excluded. E.g. "Not randomised (RANDFL != 'Y')".
population	Character or NULL. Which population flag this exclusion relates to (e.g. "SAFFL"). Used to group the exclusion listing.
reason_code	Character or NULL. Short controlled-vocabulary code for this exclusion (e.g. "NOT_RANDOMISED"). Useful for programmatic querying of the exclusion registry.

Details

reason has no default. Undocumented exclusions are a compliance failure : this is enforced at the R level, not by convention.

Value

An lg_df containing only the rows that passed the filter. Excluded rows are recorded in the session store.

See Also

lg_tag(), lg_exclusions(), lg_disposition()

Examples

```
lg_start()
adsl <- lg_tag(
  data.frame(
    USUBJID = c("01", "02", "03"),
    RANDFL = c("Y", "N", "Y"),
    SAFFL = c("Y", "N", "Y")
```

```
  ),
  dataset_id = "ADSL"
)

adsl_rand <- lg_filter(
  adsl,
  RANDFL == "Y",
  reason = "Not randomised (RANDFL != 'Y')",
  reason_code = "NOT_RANDOMISED",
  population = "RANDFL"
)
```

`lg_history` *Retrieve the operation history recorded on a tagged object*

Description

Every `lg_df` accumulates the sequence of `lg_filter()`, `lg_derive()`, and `lg_join()` operations that produced it, in its `lg_history` attribute. `lg_history()` returns that sequence directly rather than requiring `attr(data, "lg_history")`.

Usage

```
lg_history(data)
```

Arguments

`data`	An `lg_df` object.

Details

The returned object prints as a readable summary rather than a raw nested list — when empty, it reports plainly that no operations are recorded for this object yet, rather than printing a bare, uninformative `list()`.

Value

An `lg_history` object (a list of `lg_operation` records applied to this specific object, in the order they were applied; empty if none yet). Iterate over it, or index into it, exactly like a regular list — the class only changes how it prints.

Examples

```
lg_start()
dm <- lg_tag(
  data.frame(USUBJID = c("01", "02"), AGE = c(20L, 15L)),
  dataset_id = "DM"
)
```

```
lg_history(dm) # no operations yet

dm_f <- lg_filter(dm, AGE >= 18L, reason = "Minors excluded")
lg_history(dm_f)
```

`lg_id` *Retrieve lineage IDs from a tagged dataset*

Description

Returns the `lineage_id` vector from an `lg_df` object. Use this instead of accessing the column directly to keep code robust against future internal changes.

Usage

```
lg_id(data)
```

Arguments

`data`	An `lg_df` from `lg_tag()`.

Value

A character vector of lineage IDs, one per row.

Examples

```
lg_start()
dm <- data.frame(USUBJID = c("01-001", "01-002"), AGE = c(34L, 52L))
dm_tagged <- lg_tag(dm, dataset_id = "DM")
lg_id(dm_tagged)
```

`lg_join` *Join two tagged datasets with lineage tracking*

Description

Performs a left, inner, full, or right join and records the operation in the session log. The `lineage_id` column from `x` is preserved. A secondary column records which rows of `y` contributed to each output row, enabling full bilateral tracing.

Usage

```
lg_join(
  x,
  y,
  by,
  type = c("left", "inner", "full", "right"),
  description = NULL
)
```

Arguments

x, y	lg_df objects.
by	Character vector of join keys, passed to the underlying dplyr::left_join() (etc.) call.
type	Join type: "left" (default), "inner", "full", "right".
description	Character or NULL. Description of the join purpose (e.g. "Merge first dose date from EX domain"). For type = "inner" or type = "right", description becomes **mandatory** the moment the join actually drops one or more rows of x (i.e. x rows with no matching y record) : those dropped rows are subjects being silently removed from the pipeline, and per lineager's core design, every exclusion must carry a documented reason. If no rows end up dropped, description stays optional as before.

Details

Only unmatched rows of x are exclusion-tracked (since x is treated as the primary, subject-carrying dataset in lineager's model). Unmatched rows of y dropped by "left" or "inner" joins are not separately logged as exclusions of y's own dataset : if y-side row loss also needs documented tracking for your use case, log it explicitly with lg_filter() on y before joining.

Value

An lg_df with the joined result. A lineage_id_y column is added recording the contributing row IDs from y, matching prior versions of lineager. If x already carries a lineage_id_y column from an earlier join in the same chain (e.g. joining a third dataset onto the result of a previous lg_join() call), this join's own y-tracing column is instead named lineage_id_y__<op_id> (e.g. lineage_id_y__op_0003) so it cannot silently collide with – or overwrite – the earlier join's tracing column. A message is printed whenever this fallback naming is used.

See Also

lg_derive(), lg_filter()

Examples

```
lg_start()

adsl <- lg_tag(
```

```
  data.frame(USUBJID = c("01", "02"), TRT01P = c("Active", "Placebo")),
  dataset_id = "ADSL"
)
ex_summary <- lg_tag(
  data.frame(USUBJID = c("01", "02"), EXSTDTC_min = c("2026-01-01", "2026-01-03")),
  dataset_id = "EX_SUMM"
)

adsl_ex <- lg_join(adsl, ex_summary, by = "USUBJID",
                   description = "First dose date from EX domain")
```

`lg_lineage` *Build a pipeline lineage graph from the active session*

Description

Constructs a visual representation of the full pipeline : every tagged dataset, every `lg_derive()`, `lg_join()`, and `lg_filter()` operation, and every exclusion branch : as a list of nodes and edges with a Graphviz DOT string.

Usage

```
lg_lineage(rankdir = c("TB", "LR"))
```

Arguments

`rankdir`	Character. Layout direction: `"TB"` (top to bottom, default) or `"LR"` (left to right).

Details

Render with `lg_plot()` for inline display in RStudio or a knitr document, or write the DOT string to a file and render externally with Graphviz.

Value

An `lg_lineage` object (list) with components:

`nodes` Named list of node metadata.

`edges` Named list of edge metadata.

`dot` Character string. Graphviz DOT representation.

`rankdir` The layout direction used.

See Also

`lg_plot()`, `lg_operations()`, `lg_report()`

Examples

```
lg_start()
patients <- data.frame(
  USUBJID = c("P01", "P02", "P03", "P04", "P05"),
  eligible = c(TRUE, FALSE, TRUE, TRUE, FALSE),
  age = c(34L, 17L, 52L, 29L, 61L),
  stringsAsFactors = FALSE
)
pts <- lg_tag(patients, dataset_id = "PATIENTS")
pts <- lg_derive(pts,
  adult = age >= 18L,
  description = "adult flag from age"
)
lg_filter(pts, eligible & adult,
  reason = "Ineligible or under 18"
)

lin <- lg_lineage()
print(lin)
lg_end()
```

---

lg_operations *Retrieve the operation log as a data frame*

Description

Retrieve the operation log as a data frame

Usage

```
lg_operations(verbose = TRUE)
```

Arguments

verbose	Logical. Print count summary. Default TRUE.

Value

A data.frame of all recorded operations, with columns op_id, op_type, dataset_id, description, population (NA for non-FILTER operations), rows_in, rows_out, rows_excluded (rows_in - rows_out when not directly recorded), and timestamp.

lg_plot *Render a lineage graph*

Description

Renders the lineage graph returned by lg_lineage() as an interactive inline widget (using DiagrammeR if installed), or writes the DOT source to a file for rendering with Graphviz externally.

Usage

```
lg_plot(lineage, output = NULL)
```

Arguments

lineage	An lg_lineage object from lg_lineage().
output	Character or NULL. File path for DOT output (e.g. "pipeline.dot"). When NULL (default), renders inline using DiagrammeR::grViz() if available, otherwise prints the DOT source to the console.

Value

The lg_lineage object, invisibly.

See Also

lg_lineage()

Examples

```
lg_start()
pts <- lg_tag(
  data.frame(
    USUBJID = c("P01", "P02"),
    eligible = c(TRUE, FALSE),
    stringsAsFactors = FALSE
  ),
  dataset_id = "PATIENTS"
)
lg_filter(pts, eligible, reason = "Not eligible")
lin <- lg_lineage()
lg_plot(lin)
lg_end()
```

`lg_population` *Document and apply a population flag*

Description

Population flags (SAFFL, ITTFL, PPROTFL, and custom flags) are first-class objects in `lineager`. Every flag must carry its inclusion criteria, exclusion criteria, and plain-English definition : the information needed to reconstruct the Reviewer's Guide population section automatically.

Usage

```
lg_population(
  data,
  flag_var,
  label,
  definition,
  incl_criteria,
  excl_criteria = NULL,
  included_value = "Y"
)
```

Arguments

`data`	An `lg_df` containing the flag variable.
`flag_var`	Character. The flag variable name (e.g. `"SAFFL"`).
`label`	Character. Human label (e.g. `"Safety Analysis Flag"`).
`definition`	Character. Plain-English definition for regulatory reviewers (e.g. `"All randomised subjects who received at least one dose of study medication"`).
`incl_criteria`	Character vector of inclusion criteria as R expressions or plain English. At least one required.
`excl_criteria`	Character vector of explicit exclusion criteria. `NULL` if there are none beyond failing inclusion.
`included_value`	The value of `flag_var` that denotes inclusion. Defaults to `"Y"` (the CDISC convention), but `lineager` is general-purpose : if your flag is a logical column, pass `included_value = TRUE`; for any other custom coding, pass the actual included-value directly. Using the wrong value here silently produces incorrect included/excluded counts (e.g. a logical `TRUE/FALSE` flag compared against `"Y"` will count every row as excluded).

Details

The flag variable must already exist in `data`. `lg_population()` documents it; it does not compute it. Compute the flag first with `lg_derive()`, then call `lg_population()` to register its definition.

Value

data, invisibly (for pipe use).

See Also

lg_filter(), lg_disposition(), lg_report()

Examples

```
lg_start()
adsl <- lg_tag(
  data.frame(
    USUBJID = c("01", "02", "03"),
    RANDFL = c("Y", "N", "Y"), EXOCCUR = c("Y", "N", "Y"),
    SAFFL = c("Y", "N", "Y")
  ),
  dataset_id = "ADSL"
)

lg_population(
  adsl,
  flag_var = "SAFFL",
  label = "Safety Analysis Flag",
  definition = "All randomised subjects who received at least one dose",
  incl_criteria = c("RANDFL == 'Y'", "EXOCCUR == 'Y'"),
  excl_criteria = "No study drug administered (EXOCCUR != 'Y')"
)
```

---

lg_report *Generate a CDISC Reviewer's Guide-aligned provenance report*

Description

Compiles all provenance collected during the active session into a structured, self-contained HTML document suitable for inclusion in a regulatory submission package.

Usage

```
lg_report(
  format = "html",
  output = NULL,
  title = "Data Provenance Report",
  study_id = .lg$study_id,
  sponsor = NULL,
  author = NULL,
  date = Sys.Date()
)
```

Arguments

format	Character. Output format: "html" (default). PDF requires Quarto CLI and a LaTeX installation.
output	Character or NULL. Output file path. If NULL, returns the report as a character string (HTML) without writing to disk.
title	Character. Report title.
study_id	Character or NULL. Study identifier for the report header.
sponsor	Character or NULL. Sponsor name.
author	Character or NULL. Analyst name.
date	Date or Character. Report date. Defaults to today.

Details

The report covers:

- **Dataset inventory** : all tagged datasets, row counts, sources
- **Subject disposition** : CONSORT-style disposition table from all lg_filter() calls
- **Population flags** : definitions, criteria, and counts for all lg_population() registrations
- **Variable derivations** : SDTM-to-ADaM mappings from lg_spec() registrations
- **Operation log** : full sequence of pipeline operations
- **Exclusion listing** : every excluded subject with reason and population

Value

The output file path (if output is specified) or the HTML string (if output is NULL), invisibly.

See Also

lg_start(), lg_exclusions(), lg_disposition()

Examples

```
lg_start(study_id = "TRIAL-001", analysis_id = "primary")

# ... tagging, filtering, deriving, spec registration ...

lg_report(
  output   = tempfile(fileext = ".html"),
  title    = "Data Provenance Report: TRIAL-001",
  sponsor  = "Example Pharma Ltd",
  author   = "J. Smith, Biostatistician"
)
```

lg_spec *Document an SDTM-to-ADaM variable derivation*

Description

Records a structured derivation specification linking an ADaM analysis variable back to its SDTM source. These specs are the basis for the variable derivation section of the CDISC Reviewer's Guide, auto-generated by `lg_report()`.

Usage

```
lg_spec(
  adam_dataset,
  adam_var,
  label,
  source_domain,
  source_var,
  derivation,
  conditions = NULL
)
```

Arguments

`adam_dataset`	Character. ADaM dataset name (e.g. `"ADLB"`).
`adam_var`	Character. ADaM variable name (e.g. `"AVAL"`).
`label`	Character. Variable label.
`source_domain`	Character. Source SDTM domain (e.g. `"LB"`).
`source_var`	Character. Source SDTM variable (e.g. `"LBSTRESN"`).
`derivation`	Character. Plain-English description of how the ADaM variable is derived from the source.
`conditions`	Character vector or `NULL`. Conditions under which this derivation applies. `NULL` means it applies unconditionally.

Value

Invisibly `NULL`.

See Also

`lg_derive()`, `lg_report()`

Examples

```
lg_start()

lg_spec(
  adam_dataset = "ADLB",
  adam_var = "AVAL",
  label = "Analysis Value",
  source_domain = "LB",
  source_var = "LBSTRESN",
  derivation = "LBSTRESN; numeric conversion of LBORRES where LBSTRESN is missing",
  conditions = "LBSTAT != 'NOT DONE'"
)
```

---

lg_start *Start a lineager provenance session*

Description

Initialises the session store. Call once at the top of your analysis script, before any lg_tag(), lg_filter(), or lg_derive() calls. Resets any prior session state.

Usage

```
lg_start(study_id = NULL, analysis_id = NULL)
```

Arguments

study_id	Character or NULL. Optional study identifier included in reports.
analysis_id	Character or NULL. Optional analysis identifier.

Value

Invisibly NULL.

See Also

lg_end(), lg_tag(), lg_report()

Examples

```
lg_start(study_id = "TRIAL-001", analysis_id = "primary-efficacy")
lg_end()
```

lg_tag *Tag a dataset to begin lineage tracking*

Description

Assigns a unique lineage identifier (`lineage_id`) to every row and registers the dataset in the active session store. This is the entry point to `lineager` : all other functions require a tagged data frame.

Usage

```
lg_tag(
  data,
  dataset_id,
  domain = NULL,
  label = NULL,
  source = NULL,
  overwrite = FALSE
)
```

Arguments

`data`	A `data.frame` or `tibble`.
`dataset_id`	Character. Short identifier for this dataset, e.g. `"LB"`, `"ADLB"`, `"ADSL"`. Used as the prefix in lineage IDs and in report output.
`domain`	Character or `NULL`. CDISC domain code if applicable (e.g. `"DM"`, `"LB"`, `"AE"`). Used for SDTM-to-ADaM mapping and Reviewer's Guide output.
`label`	Character or `NULL`. Human-readable label for the dataset (e.g. `"Laboratory test results"`). Used in reports.
`source`	Character or `NULL`. Source file or system description.
`overwrite`	Logical. If `dataset_id` is already registered in this session, `lg_tag()` errors by default : any `lg_df` object still held from the previous registration would silently stop being traceable via `lg_trace()` the moment the registration is replaced. Set `overwrite = TRUE` to explicitly allow re-tagging (e.g. intentionally re-running a step) and acknowledge that the prior object is no longer traceable.

Details

The `lineage_id` column is added at position 1 and is preserved through `lg_filter()`, `lg_derive()`, and `lg_join()` operations. It allows every row in any downstream dataset to be traced back to its origin.

Value

An `lg_df` object : a `data.frame` with a `lineage_id` column and lineage metadata stored in attributes.

See Also

lg_filter(), lg_derive(), lg_trace()

Examples

```
lg_start()

dm <- data.frame(
  USUBJID = c("01-001", "01-002", "01-003"),
  AGE     = c(34L, 52L, 47L),
  SEX     = c("M", "F", "M")
)

dm_tagged <- lg_tag(dm, dataset_id = "DM", domain = "DM",
                    label = "Demographics")
dm_tagged
```

---

lg_trace *Trace a subject's complete lineage journey*

Description

Given a USUBJID (or a `lineage_id` value), returns the complete history of that subject across all tagged datasets and operations in the session: which datasets they appear in, which operations they passed through or were excluded by, and which population flags apply to them.

Usage

```
lg_trace(usubjid, verbose = TRUE)
```

Arguments

usubjid	Character. The subject identifier to trace. Must match a value of USUBJID in at least one tagged dataset.
verbose	Logical. If TRUE (default), prints a formatted trace to the console.

Details

This is the key regulatory tracing capability : a reviewer can ask "show me everything that happened to subject 01-042" and get a complete, programmatically generated answer.

Value

A list (invisibly) with components:

`usubjid` The traced subject ID.

`datasets` Character vector of dataset IDs the subject appears in.

`operations` Data frame of operations applied to datasets containing this subject.

`exclusions` Data frame of exclusion records for this subject, or a zero-row data frame if none.

`populations` Named list of population flag values for this subject across all registered populations.

See Also

`lg_exclusions()`, `lg_disposition()`

Examples

```
lg_start()
adsl <- lg_tag(
  data.frame(
    USUBJID = c("01", "02", "03"),
    RANDFL = c("Y", "N", "Y")
  ),
  dataset_id = "ADSL"
)
lg_filter(adsl, RANDFL == "Y",
  reason = "Not randomised", population = "RANDFL"
)

lg_trace("02")
```

`[.lg_df` *Subset an* `lg_df`*, preserving lineage attributes*

Description

Subset an `lg_df`, preserving lineage attributes

Usage

```
## S3 method for class 'lg_df'
x[i, j, drop = FALSE]
```

Arguments

`x`	An `lg_df` object.
`i`	Row index, as in `[.data.frame`.
`j`	Column index, as in `[.data.frame`.
`drop`	Ignored : `lg_df` subsetting always behaves as though `drop = FALSE`. See Details.

Details

`[.lg_df` deliberately forces `drop = FALSE`, unlike base `[.data.frame`. This means single-column subsetting (e.g. `df[, "col"]`) returns a one-column `lg_df`/`data.frame` rather than a bare vector, so the lineage attributes are never silently lost through ordinary subsetting. Use `df[[col]]` or `lg_id(df)` when a plain vector is what you actually want.

Value

An `lg_df` with lineage attributes preserved (or a plain `data.frame`/vector for subsetting operations where preservation is not applicable, matching normal `[.data.frame` fallback behaviour).

Index