Package: sas2r (via r-universe)

November 4, 2026

Title Translate 'SAS' Programs to R with Dependency-Aware Migration

Version 0.5.8

Description Translates 'SAS' programs and projects into R code, orders connected programs by their dependencies, and executes generated code. Compares generated datasets with supplied reference outputs and reports unresolved translation and validation findings. Rule-based translation works offline without a 'SAS' installation. Optional language model translation, review, and repair use a user-configured provider through 'ellmer' and may require an account and API credentials.

License Apache License (>= 2)

URL https://github.com/songyue-chen/sas2r

BugReports https://github.com/songyue-chen/sas2r/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

Depends R (>= 4.1)

Imports callr, cli (>= 3.4), clue, filelock, haven, jsonlite, stats, tibble, utils, vctrs, yaml

Suggests diffdf, dplyr (>= 1.1.0), ellmer (>= 0.4.2), ggplot2, knitr, pdftools, pkgload, ps, rmarkdown, S7, testthat (>= 3.1.7), withr

Config/testthat/edition 3

Config/pak/sysreqs make libx11-dev zlib1g-dev

Repository https://songyue-chen.r-universe.dev

Date/Publication 2026-11-04 04:26:00 UTC

RemoteUrl https://github.com/songyue-chen/sas2r

RemoteRef HEAD

RemoteSha 775e0cf40d9bfeadedf1a4f1a5fc3fba93f86fb2

Contents

analyze_output_order . . . 3
apply_format . . . 3
as_digest_json . . . 5
chr_cmp . . . 5
compare_aligned_outputs . . . 6
compare_datasets . . . 7
compare_profile . . . 8
diff_digest . . . 9
lib_delete . . . 10
lib_exists . . . 11
lib_members . . . 12
lib_read . . . 13
passed . . . 14
print.sas2r_translation . . . 15
print.sas2r_transpilation . . . 15
qc_profile . . . 16
read_comparison_report . . . 17
sas_code . . . 18
sas_compress . . . 18
sas_config . . . 19
sas_display . . . 20
sas_if_else . . . 21
sas_llm . . . 22
sas_llm_models . . . 23
sas_llm_probe . . . 24
sas_merge . . . 25
sas_min . . . 26
sas_missing . . . 27
sas_preflight . . . 28
sas_round . . . 30
sas_sort . . . 31
sas_sum . . . 31
sas_translate . . . 32
sas_write . . . 35
sas2r_fold_names . . . 36
sas2r_libname_assign . . . 37
sas2r_resolve_registry . . . 38
sas2r_runtime . . . 39
sas2r_source_include . . . 41
write_comparison_report . . . 42

Index **43**

analyze_output_order *Analyze dataset row order independently of content equivalence*

Description

Analyze dataset row order independently of content equivalence

Usage

```
analyze_output_order(reference, candidate, pairs, context = NULL)
```

Arguments

reference	Reference data frame or tibble.
candidate	Candidate data frame or tibble.
pairs	Matched pairs tibble (reference_row, candidate_row).
context	Optional alignment context list.

Value

A list containing order analysis metadata.

Examples

```
ref <- data.frame(id = 1:3, v = c("a", "b", "c"))
cand <- data.frame(id = c(3, 1, 2), v = c("c", "a", "b"))
pairs <- tibble::tibble(reference_row = 1:3, candidate_row = c(2L, 3L, 1L))

# Content is equivalent; only the row order differs.
res <- analyze_output_order(ref, cand, pairs,
  context = list(order_contract = list(vars = "id", descending = FALSE)))
res$order_equivalent
res$reason
```

apply_format *Apply a SAS format*

Description

`apply_format()` applies a compiled format catalog entry (as written to a bundle's `_sas2r_formats.R`): exact `values` first, then `ranges` (each a list with `lo`, `hi`, `label`), then `other` for anything unmatched. Missing input stays missing unless the format defines `other`. `sas_put()` is `PUT(x, fmt.)` and does the same. A format name is resolved in the bundle's `.sas2r_formats` catalog. Names are case-insensitive, with an optional trailing period. Character names retain their `$` prefix; numeric and character formats are distinct. Width and decimal specifications are unsupported and raise an error rather than silently changing the requested formatting. An unknown name is an error. Startup loads the catalog before programs and macro functions run; individual formats are not separate R variables.

Usage

```
apply_format(
  x,
  fmt,
  catalog = get0(".sas2r_formats", envir = parent.frame(), inherits = TRUE)
)

sas_put(
  x,
  fmt,
  catalog = get0(".sas2r_formats", envir = parent.frame(), inherits = TRUE)
)
```

Arguments

x	The vector to format.
fmt	A format name, or a compiled format: a list with any of `values` (a named character vector), `ranges`, and `other`; `NULL` formats as character.
catalog	Named list of compiled formats, used only when `fmt` is a name. Defaults to `.sas2r_formats` in the calling environment or its parents. Outside a generated bundle, supply this list explicitly.

Value

A character vector.

See Also

Other runtime helpers: chr_cmp(), lib_delete(), lib_exists(), lib_members(), lib_read(), sas2r_fold_names(), sas2r_libname_assign(), sas2r_resolve_registry(), sas2r_source_include(), sas_compress(), sas_display(), sas_if_else(), sas_merge(), sas_min(), sas_missing(), sas_round(), sas_sort(), sas_sum()

Examples

```
sex <- list(values = c(M = "Male", F = "Female"), other = "Unknown")
apply_format(c("M", "F", "X", NA), sex)
age <- list(ranges = list(list(lo = 0, hi = 17, label = "<18"),
                          list(lo = 18, hi = 200, label = "18+")))
sas_put(c(5, 40, NA), age)
sas_put(c("M", "F"), "$SEX.", catalog = list("$sex" = sex))
```

as_digest_json *Serialize a diff digest to JSON*

Description

Serialize a diff digest to JSON

Usage

```
as_digest_json(digest)
```

Arguments

digest	An object of class sas2r_diff_digest.

Value

A JSON string representing the diff digest.

Examples

```
sas <- data.frame(usubjid = c("01-001", "01-002"), aval = c(1.0, 2.0))
r <- data.frame(usubjid = c("01-001", "01-002"), aval = c(1.0, 2.5))
digest <- diff_digest(compare_datasets(sas, r, keys = "usubjid"))

# The redacted JSON payload that may cross to an LLM.
json <- as_digest_json(digest)
substr(json, 1, 60)
```

chr_cmp *Compare values the way SAS does*

Description

SAS comparison of possibly-missing values. Character operands ignore trailing blanks, and a missing character value ("" or NA) sorts below every non-missing one; numeric missing sorts below every number.

Usage

```
chr_cmp(a, b, op = NULL)
```

Arguments

a, b	Vectors to compare; recycled to a common length.
op	NULL for a three-way comparison, or one of "==", "!=", "<", "<=", ">", ">=". Translate SAS equality (= or EQ) as op = "=="; op = "=" is unsupported. Invalid operators raise an error.

Details

Supply an explicit operator in Boolean conditions. Omitting `op` returns an ordering result: equality is `0`, which R treats as false. To test equality, use `chr_cmp(a, b, op = "==")`.

Value

With `op` `NULL`, an integer vector of `-1`, `0`, `1` (missing equals missing). With an operator, the logical SAS would produce – never `NA`.

See Also

Other runtime helpers: `apply_format()`, `lib_delete()`, `lib_exists()`, `lib_members()`, `lib_read()`, `sas2r_fold_names()`, `sas2r_libname_assign()`, `sas2r_resolve_registry()`, `sas2r_source_include()`, `sas_compress()`, `sas_display()`, `sas_if_else()`, `sas_merge()`, `sas_min()`, `sas_missing()`, `sas_round()`, `sas_sort()`, `sas_sum()`

Examples

```
chr_cmp(c("A", NA, "B  "), c("A", NA, "B"))
chr_cmp(c("A", NA, "B  "), c("A", NA, "B"), op = "==")
chr_cmp(c(61, 70, NA), c(61, 60, NA), op = "==")
chr_cmp(c(1, NA, 3), c(1, 2, 2), op = "<")
```

`compare_aligned_outputs` *Compare aligned outputs under SAS-aware tolerance rules*

Description

Compare aligned outputs under SAS-aware tolerance rules

Usage

```
compare_aligned_outputs(
  reference,
  candidate,
  target,
  context = NULL,
  profile = compare_profile(),
  hashes = list(),
  resource_state = "complete",
  max_examples = OUTPUT_EVIDENCE_MAX_EXAMPLES
)
```

Arguments

reference	Reference data frame or tibble.
candidate	Candidate data frame or tibble.
target	Discovered target metadata row or list.
context	Optional alignment context list.
profile	Comparison profile.
hashes	Hash list.
resource_state	Resource status string.
max_examples	Maximum number of examples to include.

Value

An object of class sas2r_comparison_report.

Examples

```
ref <- data.frame(usubjid = c("01-001", "01-002"), aval = c(1.0, 2.0))
cand <- data.frame(usubjid = c("01-001", "01-002"), aval = c(1.0, 2.5))
target <- list(target_id = "adam.adsl", logical_dataset = "adam.adsl",
               role = "output", contributing_unit_ids = integer())

report <- compare_aligned_outputs(ref, cand, target = target)
report$report_id
report$examples
```

---

compare_datasets *Compare two datasets under SAS-aware tolerance rules*

Description

A thin wrapper over the shared cell comparator diff_aligned_cells(): rows pair by the given keys (or in row order when keys is NULL), and every cell verdict comes from the same loop the output-review engine uses.

Usage

```
compare_datasets(base, comp, profile = compare_profile(), keys = profile$keys)
```

Arguments

base	The reference dataset (e.g. read from a SAS golden output).
comp	The candidate dataset (e.g. produced by translated R).
profile	A compare_profile().
keys	Key columns for row matching; NULL compares in row order.

Value

An object of class `sas2r_comparison`.

Examples

```
sas <- data.frame(usubjid = c("01-001", "01-002", "01-003"),
                  aval = c(1.0, 2.0, 3.0))
r <- data.frame(usubjid = c("01-001", "01-002", "01-003"),
                aval = c(1.0, 2.0, 3.0 + 1e-10))

# A floating-point difference inside tolerance is not a mismatch.
cmp <- compare_datasets(sas, r, keys = "usubjid")
passed(cmp)

# A genuine difference is reported.
r$aval[3] <- 3.5
passed(compare_datasets(sas, r, keys = "usubjid"))
```

`compare_profile` *Build a comparison tolerance profile*

Description

Every equivalence judgement the comparator makes is parameterized here and nowhere else. Defaults implement the accepted SAS-to-R tolerance rules: SAS null == NA, special missings reported, trailing padding cosmetic, absolute numeric tolerance of 1e-8 and no relative tolerance. Equality uses abs(x-y) <= abs + rel * max(abs(x), abs(y)); relative tolerance can be enabled explicitly for measured quantities.

Usage

```
compare_profile(
  abs = 1e-08,
  rel = 0,
  sas_null_equals_na = TRUE,
  na_tags = c("report", "ignore", "strict"),
  padding = c("cosmetic", "strict"),
  attrs_cosmetic = c("label", "format.sas"),
  overrides = list(),
  keys = NULL
)
```

Arguments

`abs`	Absolute numeric tolerance (scalar >= 0).
`rel`	Relative numeric tolerance (scalar >= 0).

sas_null_equals_na	Whether SAS missing character values (empty string "" and all-space strings) equal R NA (logical).
na_tags	Tagged-NA handling: "report", "ignore", or "strict".
padding	Trailing blank padding handling: "cosmetic" or "strict".
attrs_cosmetic	Character vector of attribute names whose differences are cosmetic.
overrides	Named list of per-variable numeric tolerance overrides.
keys	Character vector of key column names, or NULL.

Value

An object of class sas2r_profile.

Examples

```
# Defaults implement the accepted SAS-to-R tolerance rules.
p <- compare_profile()
p$numeric

# Tighten numeric tolerance and make trailing blanks significant.
strict <- compare_profile(abs = 1e-12, rel = 1e-12, padding = "strict")

# Per-variable tolerance override, plus keys used for row matching.
compare_profile(overrides = list(aval = list(abs = 1e-6)),
                keys = c("usubjid", "paramcd"))
```

---

diff_digest *Build the redacted diff digest (the only comparison artifact that may cross to an LLM). Names, counts, magnitudes, and enumerated hints only – never cell values, never key values.*

Description

Build the redacted diff digest (the only comparison artifact that may cross to an LLM). Names, counts, magnitudes, and enumerated hints only – never cell values, never key values.

Usage

```
diff_digest(x, label = "dataset")
```

Arguments

x	An object of class sas2r_comparison.
label	Label for the dataset comparison.

Value

An object of class sas2r_diff_digest.

Examples

```
sas <- data.frame(usubjid = c("01-001", "01-002"), aval = c(1.0, 2.0))
r <- data.frame(usubjid = c("01-001", "01-002"), aval = c(1.0, 2.5))
cmp <- compare_datasets(sas, r, keys = "usubjid")

# Names, counts and magnitudes only -- never cell values, never key values.
d <- diff_digest(cmp, label = "adsl")
d$vars
d$pattern_hints
```

`lib_delete` *Delete explicitly named datasets from a writable library*

Description

Removes the stored dataset, rather than just an R object. Missing members produce a message and do not stop deletion of other members. Names are matched case-insensitively across rds, xpt and sas7bdat files.

Usage

```
lib_delete(libref, members)
```

Arguments

`libref`	A configured library name, such as `"work"`.
`members`	Character vector of explicit dataset names, without librefs or file extensions. An empty vector does nothing.

Details

Only an explicit vector of ordinary SAS member names is supported. Expand a known list into names before calling; prefix lists, ranges and `_ALL_` are not interpreted. Dynamic selection that cannot be resolved must remain an explicit unsupported operation, never a silent no-op.

This helper deletes from the configured write directory. If a requested member also exists in a different read directory, deletion is unsupported: removing the written copy would expose that original again to `lib_read()`. The whole request is checked before any file is removed, and input library files in a separate read directory are never deleted.

Value

The names of removed members, invisibly.

See Also

Other runtime helpers: `apply_format()`, `chr_cmp()`, `lib_exists()`, `lib_members()`, `lib_read()`, `sas2r_fold_names()`, `sas2r_libname_assign()`, `sas2r_resolve_registry()`, `sas2r_source_include()`, `sas_compress()`, `sas_display()`, `sas_if_else()`, `sas_merge()`, `sas_min()`, `sas_missing()`, `sas_round()`, `sas_sort()`, `sas_sum()`

Examples

```
local({
  folder <- tempfile("work")
  .sas2r_registry <- list(work = list(path = folder, write = "rds"))
  lib_write(data.frame(x = 1), "work", "scratch")
  lib_delete("work", "scratch")
  unlink(folder, recursive = TRUE)
})
```

`lib_exists` *Test whether a library dataset member exists*

Description

Uses the same registry and write/read search order as `lib_read()`, without reading or parsing data. A present unreadable or zero-row dataset exists. Invalid arguments and an unknown library remain errors, not `FALSE`. Supports ordinary RDS, SAS7BDAT and XPT member files, not SAS views or arbitrary external paths. Existence does not establish readability.

Usage

```
lib_exists(libref, member)
```

Arguments

`libref`	A single library name, without a member or path.
`member`	A single dataset member name, without a path or extension.

Value

A single logical value.

See Also

Other runtime helpers: `apply_format()`, `chr_cmp()`, `lib_delete()`, `lib_members()`, `lib_read()`, `sas2r_fold_names()`, `sas2r_libname_assign()`, `sas2r_resolve_registry()`, `sas2r_source_include()`, `sas_compress()`, `sas_display()`, `sas_if_else()`, `sas_merge()`, `sas_min()`, `sas_missing()`, `sas_round()`, `sas_sort()`, `sas_sum()`

Examples

```
local({
  .sas2r_registry <- list(work = list(path = tempdir()))
  lib_exists("work", "not_created")
})
```

`lib_members` *List ordinary dataset members in a configured library*

Description

Lists names without reading dataset rows. Uses the same member resolver and write/read precedence as `lib_exists()` and `lib_read()`. Duplicate storage formats yield one name. Names and extensions must be resolvable on the local filesystem; case is preserved. Unsupported views, name literals and ranges are not expanded. An unknown library or inaccessible configured directory is an error, distinct from an empty library. A not-yet-created write directory is empty; a missing separate read directory remains a configuration error.

Usage

```
lib_members(libref)
```

Arguments

`libref`	A single registered library name, without a member or path.

Details

Listing does not authorize deletion of protected inputs: `lib_delete()` retains its separate-input restriction. This is a runtime helper, not an agent tool or permission to send library contents to a model.

Value

A sorted character vector of ordinary resolvable member names.

See Also

Other runtime helpers: `apply_format()`, `chr_cmp()`, `lib_delete()`, `lib_exists()`, `lib_read()`, `sas2r_fold_names()`, `sas2r_libname_assign()`, `sas2r_resolve_registry()`, `sas2r_source_include()`, `sas_compress()`, `sas_display()`, `sas_if_else()`, `sas_merge()`, `sas_min()`, `sas_missing()`, `sas_round()`, `sas_sort()`, `sas_sum()`

Examples

```
local({
  folder <- tempfile()
  dir.create(folder)
  .sas2r_registry <- list(work = list(path = folder))
  lib_write(data.frame(id = 1), "work", "scratch")
  members <- lib_members("work")
  lib_delete("work", members)
  unlink(folder, recursive = TRUE)
})
```

lib_read *Read and write datasets through librefs*

Description

`lib_read("lib", "member")` reads a dataset from a libref as the registry configures it: the libref's write path is tried first (so a dataset the bundle produced earlier wins), then its read path, for `.rds`, `.sas7bdat`, and `.xpt` in that order. The result carries class `sas2r_dataset`, whose `$` and `[[` methods resolve column names case-insensitively, as SAS variable names do. `lib_write(df, "lib", "member")` writes to the libref's write path in its configured format (`rds` or `xpt`), creating the directory on demand.

Usage

```
lib_read(libref, member, ...)

lib_write(df, libref, member, ...)
```

Arguments

`libref`	The libref, as a single string.
`member`	The dataset member name, as a single string.
`...`	Not used; present so a rejected alias fails with a clear message.
`df`	The data frame to write.

Details

Exactly one call form is accepted for each. Combined `"lib.member"` strings, single-argument calls, and `dataset=/table=` aliases fail with the canonical form in the message, so a wrong call in an agent translation becomes repair feedback instead of being silently reinterpreted.

The registry is the `.sas2r_registry` list a bundle's `autoexec.R` defines; in a translated program it is loaded by the bootstrap header. To use these helpers interactively, create `runtime <- new.env()` and source that file with `local = runtime, chdir = TRUE`, then call `runtime$lib_read()` or `runtime$lib_write()`.

Value

`lib_read()` returns the dataset with class `sas2r_dataset`; `lib_write()` returns `df`, invisibly.

See Also

Other runtime helpers: `apply_format()`, `chr_cmp()`, `lib_delete()`, `lib_exists()`, `lib_members()`, `sas2r_fold_names()`, `sas2r_libname_assign()`, `sas2r_resolve_registry()`, `sas2r_source_include()`, `sas_compress()`, `sas_display()`, `sas_if_else()`, `sas_merge()`, `sas_min()`, `sas_missing()`, `sas_round()`, `sas_sort()`, `sas_sum()`

Examples

```
local({
  lib <- tempfile("lib"); dir.create(lib)
  .sas2r_registry <- list(
    raw = list(read_path = lib, write_path = lib, engine = "rds", write = "rds")
  )
  lib_write(data.frame(USUBJID = c("01", "02"), AVAL = c(1, 2)), "raw", "dm")
  dm <- lib_read("raw", "dm")
  dm$usubjid          # case-insensitive column access
  unlink(lib, recursive = TRUE)
})
```

---

passed *Did a comparison pass within tolerance?*

Description

Did a comparison pass within tolerance?

Usage

```
passed(x)
```

Arguments

x	An object of class sas2r_comparison.

Value

Logical indicating whether the comparison passed.

Examples

```
sas <- data.frame(usubjid = c("01-001", "01-002"), aval = c(1.0, 2.0))
r <- data.frame(usubjid = c("01-001", "01-002"), aval = c(1.0, 2.0))
passed(compare_datasets(sas, r, keys = "usubjid"))
```

`print.sas2r_translation` *Print a sas2r translation summary*

Description

Print a sas2r translation summary

Usage

```
## S3 method for class 'sas2r_translation'
print(x, ...)
```

Arguments

`x`	A `sas2r_translation` object.
`...`	Additional arguments (ignored).

Value

The input `x`, invisibly.

`print.sas2r_transpilation` *Print a sas2r transpilation summary*

Description

Print a sas2r transpilation summary

Usage

```
## S3 method for class 'sas2r_transpilation'
print(x, ...)
```

Arguments

`x`	A `sas2r_transpilation` object.
`...`	Additional arguments (ignored).

Value

The input `x`, invisibly.

qc_profile *Define reusable dataset quality requirements*

Description

Assign a profile a name in `outputs$profiles`, then select it with `outputs$assertions[["adam.adsl"]]$profile`. Target assertions override whole profile fields. These are explicit study requirements, not an automatic CDISC compliance assessment or evidence of SAS equivalence.

Usage

```
qc_profile(
  required_columns = NULL,
  labels = NULL,
  formats = NULL,
  types = NULL,
  column_order = NULL,
  keys = NULL,
  unique_keys = NULL,
  row_count = NULL,
  min_rows = NULL,
  max_rows = NULL,
  numeric_tolerance = NULL,
  tolerances = NULL
)
```

Arguments

`required_columns`
: Character vector of required column names.

`labels`, `formats`
: Named character vectors of exact required variable labels and `format.sas` attributes. Missing attributes fail the check.

`types`
: Named character vector: `numeric` (integer or double), `double`, `integer`, `character`, `factor` (including ordered factors), `logical`, `Date`, or `POSIXct`.

`column_order`
: Exact ordered vector of all expected column names.

`keys`
: Row alignment columns, matched case-insensitively. Requiring uniqueness also requires nonmissing, nonblank key values.

`unique_keys`
: Whether the combined key must be nonmissing and unique.

`row_count`, `min_rows`, `max_rows`
: Exact, minimum, or maximum row count.

`numeric_tolerance`
: Absolute tolerance for reference comparison. When set, the relative tolerance defaults to zero, as for target assertions.

`tolerances`
: Named variable overrides with `abs` and/or `rel`, using the same policy as `compare_profile()`. Only evaluated with a reference.

Value

A plain list containing only explicitly supplied, non-NULL fields. Omitted fields inherit global settings. Explicit FALSE and empty lists override them. Types describe physical R columns: factors do not satisfy character, even when their values compare equal to character data.

Examples

```
adsl_qc <- qc_profile(
  required_columns = c("USUBJID", "AGE"),
  keys = "USUBJID", unique_keys = TRUE,
  types = c(USUBJID = "character", AGE = "numeric"),
  min_rows = 1, tolerances = list(AGE = list(abs = 0, rel = 0))
)
```

---

read_comparison_report

Read a bounded comparison report from the registry

Description

Read a bounded comparison report from the registry

Usage

```
read_comparison_report(report_id, registry)
```

Arguments

report_id	Report identifier string.
registry	Registry environment or list.

Value

An object of class sas2r_comparison_report.

Examples

```
ref <- data.frame(usubjid = "01-001", aval = 1.0)
cand <- data.frame(usubjid = "01-001", aval = 2.0)
target <- list(target_id = "adam.adsl", logical_dataset = "adam.adsl",
               role = "output", contributing_unit_ids = integer())
report <- compare_aligned_outputs(ref, cand, target = target)

# The registry may be a named list or an environment.
registry <- list()
registry[[report$report_id]] <- report
back <- read_comparison_report(report$report_id, registry)
identical(back$report_id, report$report_id)
```

sas_code *Retrieve staged code for a translated program from the selected bundle*

Description

Recursively inspects the selected bundle directory and resolves by component ID, relative file path, basename, or integer index.

Usage

```
sas_code(x, file = 1L)
```

Arguments

x	A sas2r_translation object.
file	Staged file index or name/component ID.

Value

Character string containing the program code.

Examples

```
sas_dir <- file.path(tempdir(), "sas2r-code-example")
dir.create(sas_dir, showWarnings = FALSE)
writeLines(c("proc sort data=work.dm out=work.dm_sorted;",
             "  by usubjid;",
             "run;"), file.path(sas_dir, "sort.sas"))
res <- sas_translate(sas_dir, out_dir = tempfile("sas2r-out"), execute = FALSE)

# Resolve by index, or by file name / component id.
cat(sas_code(res, 1L))
```

sas_compress *SAS character functions*

Description

sas_compress() is COMPRESS: every character listed in chars is removed literally, including] and -. SAS modifiers are unsupported. sas_substr() is SUBSTR; a NULL len reads to the end of the string. sas_length() is LENGTH: trailing blanks do not count, and a missing or blank value has length 1, as in SAS.

Usage

```
sas_compress(x, chars = " ")

sas_substr(x, pos, len = NULL)

sas_length(x)
```

Arguments

x	A character vector (or a vector coerced to character).
chars	Characters to remove.
pos, len	Start position and length, as in SAS.

Value

`sas_compress()` and `sas_substr()` return character vectors; `sas_length()` an integer vector.

See Also

Other runtime helpers: `apply_format()`, `chr_cmp()`, `lib_delete()`, `lib_exists()`, `lib_members()`, `lib_read()`, `sas2r_fold_names()`, `sas2r_libname_assign()`, `sas2r_resolve_registry()`, `sas2r_source_include()`, `sas_display()`, `sas_if_else()`, `sas_merge()`, `sas_min()`, `sas_missing()`, `sas_round()`, `sas_sort()`, `sas_sum()`

Examples

```
sas_compress("a b c")
sas_substr("abcdef", 2, 3)
sas_length(c("abc  ", "", NA))
```

---

sas_config *Load project configuration (optional by design)*

Description

Looks for `_sas2r.yml` upward from `start`; the `SAS2R_CONFIG` environment variable overrides discovery. No config file is the primary supported case: built-in defaults let a bare SAS script be scanned, assessed, and translated with no setup at all. Every relative configured path – `libraries`, `macros.search_path`, `includes.roots`, `environment.autoexec`, `migration.execution_order`, and output/comparison reference paths – is resolved against the configuration file's own directory, never against the working directory: those roots reach `%include` occurrence identity, which must not depend on where the scan was launched from. One anchoring rule governs these paths, so a `~`-prefixed entry is treated as already carrying its own base everywhere rather than in some keys only, because R expands `~` against the user's home directory and prefixing a base onto it would corrupt the path.

Usage

```
sas_config(path = NULL, start = ".")
```

Arguments

path	Path to a configuration file. Defaults to NULL (use discovery).
start	Directory from which to search upwards for _sas2r.yml. Defaults to ".".

Details

A caller-supplied configuration list has no file of its own, so sas_project() resolves whatever is still relative against the project root. Both bases are facts about the project's layout, and both travel with the checkout.

YAML boolean settings use true and false. Tokens such as N, Y, yes, and no remain strings, preserving clinical metadata keys. Quote string metadata values that look numeric, such as SAS format "8.".

migration.execution_order optionally lists every executable root program exactly once, in its intended order within one shared WORK session. Reads use the latest preceding completed write; a later writer cannot supply an earlier read. List roots only: called macros, startup and included files retain their existing roles. The list orders the scanned source scope; path and recursive in sas_preflight() or sas_translate() still select that scope. Omitting the order retains inferred dependencies. Changing the order on a previously scanned project requires rescanning the sources.

Value

A sas2r_config object containing libraries, macro_search_path, include_roots, migration, optional normalized llm, output_review, outputs, source, and raw.

Examples

```
# Read the demo project configuration shipped with the package.
cfg <- sas_config(system.file("examples", "migration-demo", "_sas2r.yml",
                              package = "sas2r"))
class(cfg)
names(cfg$libraries)

# With no explicit file, search from a known project directory.
cfg <- sas_config(start = system.file("examples", "migration-demo", package = "sas2r"))
```

---

sas_display *Display values the way SAS prints them*

Description

Numeric missing is ".", character missing is blank, and numbers are never in scientific notation.

Usage

```
sas_display(x)
```

Arguments

x	A vector.

Value

A character vector.

See Also

Other runtime helpers: apply_format(), chr_cmp(), lib_delete(), lib_exists(), lib_members(), lib_read(), sas2r_fold_names(), sas2r_libname_assign(), sas2r_resolve_registry(), sas2r_source_include(), sas_compress(), sas_if_else(), sas_merge(), sas_min(), sas_missing(), sas_round(), sas_sort(), sas_sum()

Examples

```
sas_display(c(1, NA, 1e6))
sas_display(c("a", NA))
```

---

sas_if_else *IF/THEN/ELSE assignment with SAS semantics*

Description

A missing condition is false in SAS. The result keeps the class (Date, labelled) of the branch that carries it, which ifelse() would strip.

Usage

```
sas_if_else(cond, yes, no)
```

Arguments

cond	A logical vector.
yes, no	Values for true and false conditions; recycled.

Value

A vector of the common length.

See Also

Other runtime helpers: apply_format(), chr_cmp(), lib_delete(), lib_exists(), lib_members(), lib_read(), sas2r_fold_names(), sas2r_libname_assign(), sas2r_resolve_registry(), sas2r_source_include(), sas_compress(), sas_display(), sas_merge(), sas_min(), sas_missing(), sas_round(), sas_sort(), sas_sum()

Examples

```
sas_if_else(c(TRUE, NA, FALSE), "yes", "no")
sas_if_else(c(TRUE, FALSE), as.Date("2026-01-01"), NA)
```

---

sas_llm *Construct an LLM adapter from a project's* llm: *configuration*

Description

The supported way to obtain the llm argument that sas_translate() accepts. The adapter is backed by public ellmer APIs for one of the twelve registered provider ids. Credential values may be held in memory by adapters and redactors. Keep adapters private; reports redact registered secret values. Constructing an adapter contacts no network – use sas_llm_probe() to validate connectivity and model capabilities, and sas_llm_models() to see the inventory visible to the configured identity.

Usage

```
sas_llm(config)
```

Arguments

config	An llm: mapping, a sas2r_config from sas_config(), or an already-normalized LLM list.

Value

A sas2r_llm adapter.

See Also

sas_llm_probe(), sas_llm_models()

Examples

```
# Constructing an adapter contacts no network.
if (requireNamespace("ellmer", quietly = TRUE)) {
  llm <- sas_llm(list(provider = "anthropic", model = "claude-sonnet-4-6"))
  class(llm)
}
```

```
## Not run:
# Choose one of sas2r's registered providers.
llm <- sas_llm(list(provider = "openai", model = "gpt-4.1"))
res <- sas_translate("path/to/sas", llm = llm)

## End(Not run)
```

`sas_llm_models` *List models visible to the configured provider identity*

Description

Uses the provider's read-only inventory route when ellmer exposes one. An unavailable or non-authoritative inventory is reported explicitly and is never represented as an empty list of models. Bedrock inference-profile and custom-runtime selectors require probe-based access validation because the foundation-model inventory does not enumerate them safely.

Usage

```
sas_llm_models(config)
```

Arguments

`config`	An `llm:` mapping, a `sas2r_config`, or a normalized LLM list.

Value

A `sas2r_llm_model_inventory` record.

Examples

```
## Not run:
# Contacts the provider's read-only inventory route; needs credentials.
sas_llm_models(list(provider = "anthropic", model = "claude-sonnet-4-6"))

# Or straight from a project configuration.
sas_llm_models(sas_config("path/to/_sas2r.yml"))

## End(Not run)
```

`sas_llm_probe` *Probe LLM connectivity and configured settings with a structured ping*

Description

Validates authentication, endpoint reachability, and structured-output support and forwarding of explicit parameters on the configured model. The ping uses the configured output ceiling, or 2048 tokens when reasoning effort is configured, otherwise 32. It does not perform the negative control used by automatic translation startup to detect ignored reasoning settings, and does not populate that startup cache. The probe may invoke provider-managed authentication, including device login. A missing or expired ambient session raises a classed condition naming the command to run instead.

Usage

```
sas_llm_probe(
  llm,
  max_retries = 2L,
  log_dir = tempdir(),
  on_charge = NULL,
  tier = "cheap",
  can_attempt = NULL,
  usage_budget = NULL,
  audit_context = list()
)
```

Arguments

`llm`	A `sas2r_llm` adapter from `sas_llm()`, or an `llm:` mapping.
`max_retries`	Maximum retry attempts for a retryable failure.
`log_dir`	Directory the audit log is written to. Defaults to the R session's temporary directory; supply a directory to retain the log.
`on_charge`	Deprecated compatibility callback.
`tier`	Tier whose model is probed (`"cheap"` by default).
`can_attempt`	Optional predicate consulted before each attempt.
`usage_budget`	Optional shared usage budget. Pass one with a `ledger_path` to persist the attempt; omitted, a transient budget is created and the call is audited but not ledgered.
`audit_context`	Optional named list merged into the audit context.

Details

The ping carries no tools, so it does **not** exercise tool calling. A model that answers the probe may still lack tool support; that is caught at the first tool-using phase, not here.

An attempt that reaches the provider is written to the audit log. A call refused before transport – by the capability gate, or by an exhausted budget – returns `FALSE` without an audit entry. Reaching the

durable usage ledger additionally requires `usage_budget` to carry a `ledger_path`, as it does when a translation run probes through its own shared budget; a standalone call with no `usage_budget` gets a transient budget with none, so it is audited but not ledgered.

Value

`TRUE` when the model answered the ping, `FALSE` on a definitive nonanswer. Access, authentication, and transport failures are raised as classed conditions with secrets redacted.

See Also

`sas_llm()`, `sas_llm_models()`

Examples

```
## Not run:
# Sends a structured ping to validate connectivity and model access.
llm <- sas_llm(list(provider = "anthropic", model = "claude-sonnet-4-6"))
sas_llm_probe(llm)

## End(Not run)
```

---

sas_merge *Merge two datasets the way MERGE ... BY does*

Description

`MERGE a b; BY by;` with `IN=` semantics selected by `keep`: `"both"` (`in_a and in_b`), `"left"` (`in_a`), `"right"` (`in_b`), `"left_only"`, `"right_only"`, `"full"`. Where both datasets carry a non-key column the later dataset's value wins, columns follow statement order, and rows follow SAS BY ordering. Many-to-many keys, and duplicate keys with shared non-key columns, are refused: those cases require SAS observation-by-observation semantics that this join-based helper does not implement. Do not pre-create missing columns on one input when the other input supplies them: that invents an overlap. Preserve genuine overlapping source columns; their unsupported repeated-key semantics must not be silently dropped. Values from an input can be retained within a BY group, but reset to missing at the next BY group. If an input has no row for a key, its unique columns are missing for that key; never fill them from a preceding key.

Usage

```
sas_merge(
  a,
  b,
  by,
  keep = c("full", "both", "left", "right", "left_only", "right_only"),
  ...
)
```

Arguments

a, b	Data frames, in statement order.
by	Character vector of BY variables.
keep	Which rows to keep; see Description.
...	Ignored.

Value

The merged data frame.

See Also

Other runtime helpers: apply_format(), chr_cmp(), lib_delete(), lib_exists(), lib_members(), lib_read(), sas2r_fold_names(), sas2r_libname_assign(), sas2r_resolve_registry(), sas2r_source_include(), sas_compress(), sas_display(), sas_if_else(), sas_min(), sas_missing(), sas_round(), sas_sort(), sas_sum()

Examples

```
a <- data.frame(id = c(1, 2), x = c("a", "b"))
b <- data.frame(id = c(2, 3), y = c("B", "C"))
sas_merge(a, b, by = "id", keep = "full")
sas_merge(a, b, by = "id", keep = "both")
events <- data.frame(id = c(1, 1, 2), event = c("a", "b", "c"))
subjects <- data.frame(id = c(1, 2), base = c(10, 20))
sas_merge(events, subjects, by = "id", keep = "left") # three events remain
```

---

sas_min *Row-wise MIN and MAX with SAS missing semantics*

Description

Row-wise across the arguments, ignoring missing values; an observation whose arguments are all missing yields missing, and a single argument passes through elementwise – SAS MIN(x) is never a column aggregate.

Usage

```
sas_min(...)

sas_max(...)
```

Arguments

...	Numeric vectors, recycled to a common length.

Value

A numeric vector of that length.

See Also

Other runtime helpers: apply_format(), chr_cmp(), lib_delete(), lib_exists(), lib_members(), lib_read(), sas2r_fold_names(), sas2r_libname_assign(), sas2r_resolve_registry(), sas2r_source_include(), sas_compress(), sas_display(), sas_if_else(), sas_merge(), sas_missing(), sas_round(), sas_sort(), sas_sum()

Examples

```
sas_min(c(1, NA, 3), c(2, NA, 1))
sas_max(c(1, NA, 3), c(2, NA, 1))
```

---

sas_missing *SAS missing values and Boolean conditions*

Description

Character missing is NA or all spaces. Numeric missing is NA (including tagged missing values). A missing or zero numeric condition is false. Character conditions convert to numeric; nonnumeric text is false.

Usage

```
sas_missing(x)

sas_true(x)
```

Arguments

x	A vector.

Value

A logical vector with no missing values.

See Also

Other runtime helpers: apply_format(), chr_cmp(), lib_delete(), lib_exists(), lib_members(), lib_read(), sas2r_fold_names(), sas2r_libname_assign(), sas2r_resolve_registry(), sas2r_source_include(), sas_compress(), sas_display(), sas_if_else(), sas_merge(), sas_min(), sas_round(), sas_sort(), sas_sum()

Examples

```
sas_missing(c("", " ", NA, "A"))
sas_true(c(NA, 0, 1, -1))
```

`sas_preflight` *Inspect a migration and optionally diagnose findings*

Description

Scans sources and resolves library bindings with the same project scanner used by `sas_translate()`. Reads directory listings, not dataset contents. Does not execute programs, read dataset contents, or create output directories, scan caches, or usage ledgers. With a configured LLM, actionable findings can receive one bounded advisory diagnosis. Use `diagnose = "off"` for offline inspection without constructing an adapter or making a model call. Deterministic support findings are advisory: an AI translation may handle deferred constructs, and runtime checks can still defer supported shapes.

Usage

```
sas_preflight(
  path,
  out_dir = NULL,
  config = NULL,
  outputs = NULL,
  budget_usd = Inf,
  budget_mode = "stop",
  pricing_source = "catalog",
  pricing_rates = NULL,
  usage_limits = NULL,
  recursive = FALSE,
  max_parallel_translations = NULL,
  diagnose = c("auto", "off"),
  llm = NULL
)
```

Arguments

`path`	Path to a SAS file or directory containing SAS files, or a `sas2r_project`.
`out_dir`	Planned migration output root. `NULL` reports a temporary output root, as used by `sas_translate()`, without creating it.
`config`	Optional configuration list, YAML path, or `sas2r_config` object. With a reused project, a plain list replaces only its supplied top-level fields; omitted fields inherit, and explicit `NULL` resets a field. A YAML path or `sas2r_config` object supplies a complete configuration, as does a plain list with a source path. Changing library or source-search settings requires rescanning the source path. Reference paths in YAML resolve from its directory; paths in R lists resolve from the project directory.
`outputs`	Optional character vector or list specifying output contract overrides. Reference paths supplied here resolve from the calling working directory. Output overrides are retained in the returned project for reuse.
`budget_usd`	Dollar threshold for LLM spend. Defaults to Inf (unlimited).

budget_mode
: Dollar enforcement mode: "stop", "observe", "soft", or "strict". Defaults to "stop".

pricing_source
: Cost provenance source ("catalog", "adapter", "organization", "external"). Defaults to "catalog".

pricing_rates
: Optional organization pricing table.

usage_limits
: Optional named list of non-dollar request limits: `max_calls`, `max_retries`, `max_tool_calls`, `max_wall_time` (seconds), `max_request_bytes`, `max_request_chars`, `max_input_tokens`, and `max_output_tokens`. Each defaults to Inf. Zero prevents the corresponding request activity; unknown names and invalid values raise a configuration error.

recursive
: Logical; whether to scan subdirectories recursively. Defaults to FALSE.

max_parallel_translations
: Concurrent component workflow limit used when planning translation. Overrides `migration.max_parallel_translations`. Invalid combinations are reported by static checks; automatic diagnosis can explain these errors when a valid diagnosis budget is available.

diagnose
: `"auto"` (default) diagnoses actionable findings when an LLM is configured; `"off"` always stays offline. Advice never clears findings.

llm
: Optional `sas2r_llm` adapter for diagnosis, overriding `llm:` configuration. Missing configuration is reported, not treated as a failure.

Details

With `diagnose = "auto"`, a configured model may receive bounded SAS statements, source paths and findings, subject to budget admission; dataset contents are not sent. Configured adapters use one transport attempt capped at 120 seconds; custom supplied adapters retain their transport contract. Advice can suggest source/configuration corrections or an issue draft for a suspected sas2r bug. It does not edit sources or submit issues.

Configure `migration.execution_order` to inspect a declared shared-session sequence; see `sas_config()`. `pipeline$order_source` reports `"configured"` or `"inferred"`. A permanent input with no preceding known or possible writer still receives the normal availability check, even if a later program writes it. Unknown macro effects can leave dataset state deferred.

Value

A `sas2r_preflight` list with `sources`, point-of-use `libraries`, `inputs` (available, missing, unresolved, no_producer, deferred, backward_dependency, created_if_missing, generated, or environment), `references`, `unsupported`, scanner `findings`, shared `readiness` warnings, `outputs`, `destinations`, `budget`, `next_actions`, `diagnosis`, and actual `model_calls`. Diagnosis failures preserve the static result. Inspection errors retain their original condition with optional advice attached as `$diagnosis`. `called_macros` lists reachable search-path macro definitions and their standalone R and interface-test paths. `pipeline` reconciles every scanned file with the translation schedule and bundle execution roles, listing intentional exclusions and blocking gaps. Also includes `status`, `configured_libraries`, `schedule`, explanatory `notes`, `max_parallel_translations`, and the scanned `project`, reusable by `sas_translate()` while sources are unchanged. `ready_for_translation` means no known missing inputs or unresolved scan findings; it does not mean execution or reference validation has passed.

Examples

```
source <- tempfile(fileext = ".sas")
writeLines("data out; set raw.dm; run;", source)
check <- sas_preflight(source, diagnose = "off")
check$inputs
check$model_calls
unlink(source)
```

`sas_round` *Round the way SAS does*

Description

SAS `ROUND` rounds half away from zero, to a rounding unit; R's `round()` is banker's rounding.

Usage

```
sas_round(x, unit = 1)
```

Arguments

`x`	A numeric vector.
`unit`	The rounding unit (SAS's second argument).

Value

`x` rounded to the nearest multiple of `unit`.

See Also

Other runtime helpers: `apply_format()`, `chr_cmp()`, `lib_delete()`, `lib_exists()`, `lib_members()`, `lib_read()`, `sas2r_fold_names()`, `sas2r_libname_assign()`, `sas2r_resolve_registry()`, `sas2r_source_include()`, `sas_compress()`, `sas_display()`, `sas_if_else()`, `sas_merge()`, `sas_min()`, `sas_missing()`, `sas_sort()`, `sas_sum()`

Examples

```
sas_round(c(0.5, 1.5, -0.5))
sas_round(12.345, 0.01)
```

sas_sort *Sort a dataset the way PROC SORT does*

Description

Stable, with missing values lowest under SAS collation; `by` is matched to column names case-insensitively, and variables named in `descending` are sorted in reverse.

Usage

```
sas_sort(df, by, descending = character(), ...)
```

Arguments

`df`	A data frame.
`by`	Character vector of BY variables.
`descending`	Character vector of those BY variables to sort descending.
`...`	Ignored.

Value

`df` reordered.

See Also

Other runtime helpers: `apply_format()`, `chr_cmp()`, `lib_delete()`, `lib_exists()`, `lib_members()`, `lib_read()`, `sas2r_fold_names()`, `sas2r_libname_assign()`, `sas2r_resolve_registry()`, `sas2r_source_include()`, `sas_compress()`, `sas_display()`, `sas_if_else()`, `sas_merge()`, `sas_min()`, `sas_missing()`, `sas_round()`, `sas_sum()`

Examples

```
sas_sort(data.frame(id = c(2, NA, 1)), by = "id")
sas_sort(data.frame(id = c(2, NA, 1)), by = "id", descending = "id")
```

sas_sum *Row-wise SUM and MEAN with SAS missing semantics*

Description

SAS `SUM()` and `MEAN()` are row-wise across their arguments: each observation gets the sum or mean of its non-missing arguments, and only an observation whose arguments are all missing yields missing. Column aggregation (`PROC MEANS`, `PROC SQL`) is emitted inline by those emitters and never routed through these.

Usage

```
sas_sum(...)

sas_mean(...)
```

Arguments

...	Numeric vectors, recycled to a common length.

Value

A numeric vector of that length.

See Also

Other runtime helpers: apply_format(), chr_cmp(), lib_delete(), lib_exists(), lib_members(), lib_read(), sas2r_fold_names(), sas2r_libname_assign(), sas2r_resolve_registry(), sas2r_source_include(), sas_compress(), sas_display(), sas_if_else(), sas_merge(), sas_min(), sas_missing(), sas_round(), sas_sort()

Examples

```
sas_sum(c(1, NA, 2), c(1, NA, NA))
sas_mean(c(1, NA, 3), c(3, NA, NA))
```

---

sas_translate *Translate SAS programs or project to R with dependency-aware orchestration*

Description

Exposes the single authoritative SAS-to-R migration pipeline. SAS source is the only required input. Operates on a single SAS file (wrapped as a one-module project) or a directory of SAS files with identical internal sequencing: initializes/reconciles run -> discovers dependency graph and output contracts -> baseline generation and immediate program repair -> full bundle attempt execution and output-driven repair -> deterministic gate selection and reporting.

Usage

```
sas_translate(
  path,
  out_dir = NULL,
  config = NULL,
  execute = TRUE,
  max_program_repair_rounds = 1L,
  max_bundle_repair_rounds = NULL,
  outputs = NULL,
  agent_evidence = c("code_only", "bounded"),
```

```
  llm = NULL,
  budget_usd = Inf,
  budget_mode = "stop",
  pricing_source = "catalog",
  pricing_rates = NULL,
  usage_limits = NULL,
  recursive = FALSE,
  resume = FALSE,
  keep_raw_attempts = FALSE,
  max_bundle_repairs_per_component = 2L,
  max_parallel_translations = NULL
)
```

Arguments

path	Path to a SAS file or directory containing SAS files, or a sas2r_project.
out_dir	Output directory path for generated R bundle, attempts, and reports. Defaults to a temporary directory. Relative paths resolve from the calling working directory before workers start. Source parsing caches use the R session's temporary directory, not the input project. Wrap the call in suppressMessages() to silence routine console output, or use options(sas2r.progress = FALSE) to hide progress only. Returned results, saved reports and execution logs remain available.
config	Optional configuration list, YAML path, or sas2r_config object. With a reused project, a plain list replaces only its supplied top-level fields; omitted fields inherit, and explicit NULL resets a field. A YAML path or sas2r_config object supplies a complete configuration, as does a plain list with a source path. Changing library or source-search settings requires rescanning the source path. Reference paths in YAML resolve from its directory; paths in R lists resolve from the project directory.
execute	Logical; whether to run meaningful program smoke and full bundle execution. Defaults to TRUE.
max_program_repair_rounds	Maximum repair rounds per component in immediate loop. Defaults to 1L.
max_bundle_repair_rounds	Optional overall cap on bundle fixer calls. Defaults to NULL, allowing each component its own bounded repair allowance. An explicit zero disables bundle repair.
outputs	Optional character vector or list specifying output contract overrides. Reference paths supplied here resolve from the calling working directory. Output overrides are retained in the returned project for reuse.
agent_evidence	Agent evidence policy ("code_only" or legacy "bounded"). Both exclude record previews and raw runtime messages. Focused investigation may receive approved aggregate patterns; fixers remain reference-blind.
llm	Optional sas2r_llm instance for agent-assisted translation, review, and repair. For parallel execution, a custom adapter needs a self-contained, zero-argument

function in `attr(llm, "parallel_factory")` that returns a fresh adapter. Without a factory, execution uses one workflow and reports why. Capture only small settings: caller-global objects are not transported, and the combined encoded adapter startup data is limited to 100,000 bytes per worker.

`budget_usd` Dollar threshold for LLM spend. Defaults to Inf (unlimited).

`budget_mode` Dollar enforcement mode: "stop", "observe", "soft", or "strict". Defaults to "stop".

`pricing_source` Cost provenance source ("catalog", "adapter", "organization", "external"). Defaults to "catalog".

`pricing_rates` Optional organization pricing table.

`usage_limits` Optional named list of non-dollar request limits: `max_calls`, `max_retries`, `max_tool_calls`, `max_wall_time` (seconds), `max_request_bytes`, `max_request_chars`, `max_input_tokens`, and `max_output_tokens`. Each defaults to Inf. Zero prevents the corresponding request activity; unknown names and invalid values raise a configuration error.

`recursive` Logical; whether to scan subdirectories recursively. Defaults to FALSE.

`resume` Logical; reuse saved translation revisions and completed reviews when source, input data, configuration, helpers, and worker prompts still match. Full-bundle execution and output checks use fresh attempts; passing or deferred smoke results can be reused when their full context matches. Changing only the parallel translation limit does not invalidate reuse. Missing or edited revision files trigger regeneration. Compatible version-8 checkpoints are imported; other incompatible checkpoints regenerate with an explanation before new provider calls. Defaults to FALSE.

`keep_raw_attempts`
Logical; retain raw outputs from unselected attempts, including isolated component smoke outputs and their replay scripts. These are partial debugging artifacts, not validated final outputs. Defaults to FALSE.

`max_bundle_repairs_per_component`
Maximum bundle fixer calls per component across the run. Defaults to 2L; separate from immediate program repairs.

`max_parallel_translations`
Maximum concurrent SAS program or macro translation workflows. Overrides `migration.max_parallel_translations` in `_sas2r.yml`; defaults to 1. Model waiting can overlap on fewer CPUs. Local execution and repair remain serial. Values above 1 cannot be combined with `llm.max_tries` above 1; preflight and translation stop with a configuration error before provider calls.

Details

Missing inputs, called macros, includes and uncertain dependencies are reported while available source continues translating. Unavailable dependencies defer affected execution; they do not authorize inventing missing source or data. Component failures preserve useful work and allow other components to proceed. Unusable configuration, inconsistent pipeline coverage and critical provider or accounting failures still stop the run. Use `sas_preflight()` to inspect readiness and configure `macros.search_path` for external macro definitions.

Value

An object of S3 class "sas2r_translation" containing $run_id, $out_dir, $bundle_dir, $outputs_dir, $status, $status_reason, $graph_path, $output_contracts_path, $report_path, $report_json_path, $component_evidence, $output_assessments, $diagnostics, $repair_history, $usage, and $project. $outputs_dir contains selected contract deliverables with their library and relative-path layout (for example datasets/work/out.rds, datasets/adam/adsl.rds, and tlf/outputs/table.html); it is NULL when execution is disabled or no selected deliverables were produced. The JSON report includes separate target/reference/review coverage and effective limits.

Examples

```
# Translate a small SAS program with the deterministic rule-based engine.
# Neither an LLM nor a SAS installation is required.
sas_dir <- file.path(tempdir(), "sas2r-translate-example")
dir.create(sas_dir, showWarnings = FALSE)
writeLines(c(
  "data work.adsl;",
  "  set adam.dm;",
  "  where age >= 18;",
  "  bmi = weight / (height * height);",
  "run;",
  "",
  "proc sort data=work.adsl out=work.adsl_sorted;",
  "  by usubjid;",
  "run;"
), file.path(sas_dir, "adsl.sas"))

# execute = FALSE translates without running the generated bundle.
res <- sas_translate(sas_dir, out_dir = tempfile("sas2r-out"), execute = FALSE)
res$status
cat(sas_code(res))

## Not run:
# Run the generated bundle and let the agents translate, review, and repair
# the patterns the deterministic rules cannot prove.
llm <- sas_llm(list(provider = "anthropic", model = "claude-sonnet-4-6"))
res <- sas_translate("path/to/sas/project", llm = llm, recursive = TRUE)
sas_write(res, "path/to/r/bundle")

## End(Not run)
```

---

sas_write *Write translated code artifacts and outputs to a destination directory*

Description

Copies the selected editable bundle (including failed code), with its programs/, macros/, runtime/, autoexec.R, run.R, and README. Saved automated deliverables go to saved-outputs/ and reports to report/. Existing manual output/ files are not copied; configuration edits are kept. Input

libraries remain external dependencies documented in the README. Run `Rscript run.R` from the exported folder, or `source("run.R", chdir = TRUE)`. Warns if status is `blocked` or `needs_review`.

Usage

```
sas_write(x, dir, overwrite = FALSE)
```

Arguments

`x`	A `sas2r_translation` object.
`dir`	Target directory path to write translated files.
`overwrite`	Replace an existing sas2r export. Non-export directories must be empty. Replacing an export updates only its previously listed files; unrelated files are preserved and conflicting new paths are refused. This also replaces your edits to previously exported files. Export to a new directory to preserve an edited bundle; edits are not merged or backed up.

Value

The target directory path, invisibly.

Examples

```
sas_dir <- file.path(tempdir(), "sas2r-write-example")
dir.create(sas_dir, showWarnings = FALSE)
writeLines(c("proc sort data=work.dm out=work.dm_sorted;",
             "  by usubjid;",
             "run;"), file.path(sas_dir, "sort.sas"))
res <- sas_translate(sas_dir, out_dir = tempfile("sas2r-out"), execute = FALSE)

# Writing an unverified translation warns; the bundle is still written.
dest <- tempfile("sas2r-bundle")
suppressWarnings(sas_write(res, dest))
list.files(dest, recursive = TRUE)
```

`sas2r_fold_names` *Fold column names to lower case*

Description

SAS resolves variable names case-insensitively; R does not. Deterministic translations fold every frame to lower-case names at entry so lower-case source references (`aval`) find upper-case data columns (`AVAL`) the way SAS would.

Usage

```
sas2r_fold_names(df)
```

Arguments

df	A data frame.

Value

df with lower-case column names.

See Also

Other runtime helpers: apply_format(), chr_cmp(), lib_delete(), lib_exists(), lib_members(), lib_read(), sas2r_libname_assign(), sas2r_resolve_registry(), sas2r_source_include(), sas_compress(), sas_display(), sas_if_else(), sas_merge(), sas_min(), sas_missing(), sas_round(), sas_sort(), sas_sum()

Examples

```
names(sas2r_fold_names(data.frame(USUBJID = 1, AVAL = 2)))
```

---

sas2r_libname_assign *Bind or clear a libref at its point of use*

Description

The registry a bundle starts with is only a seed – the libraries the configuration declares, plus work. Every LIBNAME the SAS program executes changes the registry where that statement stood, so a libref that is rebound or cleared part-way through behaves at run time the way it does in SAS. sas2r_libname_assign() is LIBNAME libref '<path>'; sas2r_libname_clear() is LIBNAME libref CLEAR. Relative assignment paths use .sas2r_execution_root from autoexec.R, initially the source project directory. Registry seed paths still use the bundle folder. Reassigning a known physical library preserves its separate write directory. Newly assigned libraries write under the execution output libraries/ folder unless write_path is explicitly supplied.

Usage

```
sas2r_libname_assign(
  libref,
  read_path,
  write_path = read_path,
  engine = "sas7bdat",
  write = "rds"
)

sas2r_libname_clear(libref)
```

Arguments

libref	The libref (case-insensitive).
read_path, write_path	Directories the libref reads from and writes to.
engine	Format of members read: "sas7bdat", "xpt", or "rds".
write	Format members are written in: "rds" or "xpt".

Value

sas2r_libname_assign() returns the new registry entry, invisibly; sas2r_libname_clear() returns NULL, invisibly.

See Also

Other runtime helpers: apply_format(), chr_cmp(), lib_delete(), lib_exists(), lib_members(), lib_read(), sas2r_fold_names(), sas2r_resolve_registry(), sas2r_source_include(), sas_compress(), sas_display(), sas_if_else(), sas_merge(), sas_min(), sas_missing(), sas_round(), sas_sort(), sas_sum()

Examples

```
local({
  .sas2r_registry <- list()
  sas2r_libname_assign("raw", tempdir(), engine = "rds")
  names(.sas2r_registry)
  sas2r_libname_clear("raw")
  names(.sas2r_registry)
})
```

---

sas2r_resolve_registry

Resolve a bundle's libref registry against its folder

Description

autoexec.R lists each libref's directories the way a person writes them: an absolute path is used as is, and a relative path means inside the bundle folder. This turns the relative ones into absolute paths against root, creates every write directory, and returns the registry ready for lib_read() and lib_write(). autoexec.R calls it once, after the helpers are loaded; call it yourself only when building a registry by hand.

Usage

```
sas2r_resolve_registry(registry, root)
```

Arguments

registry	The .sas2r_registry list as written in autoexec.R.
root	The bundle folder, as an absolute path.

Details

A <FILL> placeholder left in a path – the commented entries autoexec.R carries for a library nothing binds – is refused with the libref named, rather than becoming a directory called <FILL...>.

Value

The registry, with every read_path and write_path absolute.

See Also

Other runtime helpers: apply_format(), chr_cmp(), lib_delete(), lib_exists(), lib_members(), lib_read(), sas2r_fold_names(), sas2r_libname_assign(), sas2r_source_include(), sas_compress(), sas_display(), sas_if_else(), sas_merge(), sas_min(), sas_missing(), sas_round(), sas_sort(), sas_sum()

Examples

```
root <- tempfile("bundle"); dir.create(root)
reg <- sas2r_resolve_registry(
  list(work = list(read_path = "work", write_path = "work",
                   engine = "rds", write = "rds")),
  root)
reg$work$write_path      # <root>/work, and the directory now exists
```

---

sas2r_runtime *The runtime every translated program carries*

Description

Every program sas2r translates calls a small runtime – lib_read(), lib_write(), sas_sort(), sas_merge(), chr_cmp(), the format machinery and the rest. Since sas2r 0.2.0 that runtime is exported package code with a help page per helper, **and** every bundle still carries its own copy as sas2r-helpers.R, rendered from the same source files (R/runtime-*.R) so the two are identical function for function. A bundle always loads its own runtime copy, so it can run without sas2r installed. Results still depend on input data, R and dependency versions; recheck outputs when those change.

The vignette vignette("runtime-helpers", package = "sas2r") explains how to run programs, how to use the runtime interactively, and the stability policy behind it.

How the runtime reaches a program

One guarded line at the top of each generated program sources `autoexec.R` into the program's own environment, the way a SAS session runs its autoexec first. Run `Rscript run.R` from the bundle folder to execute programs in dependency order. For individual programs, create `runtime <- new.env()` and use `source("<bundle>/autoexec.R", local = runtime, chdir = TRUE)`. Source required upstream programs into that same environment before `source("<bundle>/programs/adsl.R", local = runtime)`. `autoexec.R` is the bundle's counterpart of a SAS autoexec: generated by the translation, maintained by a person afterwards. Its LIBRARIES section says where each libref reads from and writes to (a relative path means inside the bundle folder); its LOAD section sources `runtime/sas2r-helpers.R` (this runtime), resolves those paths with `sas2r_resolve_registry()`, and sources `runtime/_sas2r_formats.R` (the compiled format catalog).

To use the runtime at the console, load it into `runtime` as above and call helpers through that environment, for example `runtime$lib_read("adam", "adsl")`. Attach sas2r when you want the help pages.

The helpers

Data access `lib_read()`, `lib_write()`, `sas2r_libname_assign()`, `sas2r_libname_clear()`, `sas2r_resolve_registry()`, `sas2r_fold_names()`; datasets read through `lib_read()` carry class `sas2r_dataset`, whose `$` and `[[` methods resolve column names case-insensitively.

SAS value semantics `chr_cmp()`, `sas_if_else()`, `sas_sum()` and `sas_mean()`, `sas_min()` and `sas_max()`, `sas_round()`, `sas_length()`, `sas_substr()`, `sas_compress()`, `sas_display()`; and two operators the package defines but does not export: `a %notin% b` (`NOT IN` with SAS missing semantics – missing values match missing values in the list; character comparison ignores trailing blanks) and `a %+% b` (`||` concatenation where a missing operand contributes nothing rather than `"NA"`; SAS pads fixed-width operands, this does not).

Sorting and merging `sas_sort()`, `sas_merge()`.

Formats `apply_format()`, `sas_put()`.

Includes `sas2r_source_include()`.

Internal helpers

Generated code calls these; they are not exported, not meant for direct use, and may change with the generator: `sas2r_registry_env()` (where the `.sas2r_registry` list lives – lexically where the runtime was loaded, then active local frames; package helpers exclude the global workspace), `sas2r_lib_entry()` (the registry entry for a libref), `sas2r_lib_member_path()` (the file a member resolves to, refusing any name that is not a plain member name), `sas2r_assignment_path()` (resolve a LIBNAME path against the execution root), `sas2r_libref_stop()` (classed libref errors), `split_ds()` (split a two-level SAS name after macro substitution).

Versioning and stability

The runtime follows the sas2r version, and every vendored copy states the version that generated it. Existing helper contracts are stable, but correctness fixes may change erroneous results. A helper is retired only after a replacement ships and the old name has warned for at least one minor version. Tests hold the source files, the vendored template, the help pages, and the lint allowlist in sync. A separate `sas2r.runtime` package is a future option when usage justifies it.

See Also

sas_translate(), sas_write(), sas_code()

---

sas2r_source_include *Run an included module where the %INCLUDE stood*

Description

Each included SAS file is translated once into its own staged module; this executes that module at one include site, into the calling program's own environment, exactly where the %INCLUDE stood.

Usage

```
sas2r_source_include(relative_path, envir = parent.frame())
```

Arguments

relative_path	The module's path relative to the staged module root.
envir	The environment to run it in; the caller's by default.

Details

The path is relative to the staged module root, and never to the working directory. The editable bundle sets this to programs/ in autoexec.R; flat execution snapshots use the nearest ancestor holding autoexec.R. Absolute paths and . or .. components are rejected, and the file the joined path actually resolves to is then required to lie under the root: rejecting the components alone would still let a symlinked directory inside the bundle read outside it, so the confinement is checked on the resolved path, where a hand-edited bundle cannot talk its way out.

Value

The resolved path of the module, invisibly.

See Also

Other runtime helpers: apply_format(), chr_cmp(), lib_delete(), lib_exists(), lib_members(), lib_read(), sas2r_fold_names(), sas2r_libname_assign(), sas2r_resolve_registry(), sas_compress(), sas_display(), sas_if_else(), sas_merge(), sas_min(), sas_missing(), sas_round(), sas_sort(), sas_sum()

Examples

```
## Not run:
# inside a translated program, where the SAS source had %include 'inc/prep.sas';
sas2r_source_include("inc/prep.R")

## End(Not run)
```

`write_comparison_report` *Write a PROC-COMPARE-style markdown report or versioned JSON comparison report*

Description

Write a PROC-COMPARE-style markdown report or versioned JSON comparison report

Usage

```
write_comparison_report(
  x,
  file = NULL,
  base_label = "SAS (base)",
  comp_label = "R (compare)",
  run_id = "default",
  dir = NULL,
  ...
)
```

Arguments

`x`	An object of class `sas2r_comparison` or `sas2r_comparison_report`.
`file`	Path to the output markdown or JSON file.
`base_label`	Label for the base dataset.
`comp_label`	Label for the compare dataset.
`run_id`	Run ID for output review directory structure.
`dir`	Base directory for `.sas2r` output review files.
`...`	Additional options.

Value

Path to the written file, invisibly.

Examples

```
sas <- data.frame(usubjid = c("01-001", "01-002"), aval = c(1.0, 2.0))
r <- data.frame(usubjid = c("01-001", "01-002"), aval = c(1.0, 2.5))
cmp <- compare_datasets(sas, r, keys = "usubjid")

# PROC COMPARE-style markdown.
md <- file.path(tempdir(), "sas2r-comparison.md")
write_comparison_report(cmp, file = md)
cat(head(readLines(md), 6), sep = "\n")
```

Index

∗ **runtime helpers**
  apply_format, 3
  chr_cmp, 5
  lib_delete, 10
  lib_exists, 11
  lib_members, 12
  lib_read, 13
  sas2r_fold_names, 36
  sas2r_libname_assign, 37
  sas2r_resolve_registry, 38
  sas2r_source_include, 41
  sas_compress, 18
  sas_display, 20
  sas_if_else, 21
  sas_merge, 25
  sas_min, 26
  sas_missing, 27
  sas_round, 30
  sas_sort, 31
  sas_sum, 31
[[<-.sas2r_dataset *(*sas2r_runtime*)*, 39
$<-.sas2r_dataset *(*sas2r_runtime*)*, 39

analyze_output_order, 3
apply_format, 3, *6*, *10–13*, *19*, *21, 22*, *26, 27*, *30–32*, *37–39*, *41*
apply_format(), *40*
as_digest_json, 5

chr_cmp, *4*, 5, *10–13*, *19*, *21, 22*, *26, 27*, *30–32*, *37–39*, *41*
chr_cmp(), *39, 40*
compare_aligned_outputs, 6
compare_datasets, 7
compare_profile, 8
compare_profile(), *7*, *16*

diff_digest, 9

lib_delete, *4*, *6*, 10, *11–13*, *19*, *21, 22*, *26, 27*, *30–32*, *37–39*, *41*
lib_delete(), *12*
lib_exists, *4*, *6*, *10*, 11, *12, 13*, *19*, *21, 22*, *26, 27*, *30–32*, *37–39*, *41*
lib_exists(), *12*
lib_members, *4*, *6*, *10, 11*, 12, *13*, *19*, *21, 22*, *26, 27*, *30–32*, *37–39*, *41*
lib_read, *4*, *6*, *10–12*, 13, *19*, *21, 22*, *26, 27*, *30–32*, *37–39*, *41*
lib_read(), *10–12*, *38–40*
lib_write *(*lib_read*)*, 13
lib_write(), *38–40*

passed, 14
print.sas2r_translation, 15
print.sas2r_transpilation, 15

qc_profile, 16

read_comparison_report, 17

sas2r_fold_names, *4*, *6*, *10–13*, *19*, *21, 22*, *26, 27*, *30–32*, 36, *38, 39*, *41*
sas2r_fold_names(), *40*
sas2r_helpers *(*sas2r_runtime*)*, 39
sas2r_libname_assign, *4*, *6*, *10–13*, *19*, *21*, *22*, *26, 27*, *30–32*, 37, *37*, *39*, *41*
sas2r_libname_assign(), *40*
sas2r_libname_clear *(*sas2r_libname_assign*)*, 37
sas2r_libname_clear(), *40*
sas2r_resolve_registry, *4*, *6*, *10–13*, *19*, *21, 22*, *26, 27*, *30–32*, *37*, 38, *38*, *41*
sas2r_resolve_registry(), *40*
sas2r_runtime, 39
sas2r_source_include, *4*, *6*, *10–13*, *19*, *21*, *22*, *26, 27*, *30–32*, *37–39*, 41
sas2r_source_include(), *40*
sas_code, 18
sas_code(), *41*
sas_compress, *4*, *6*, *10–13*, 18, *21, 22*, *26, 27*, *30–32*, *37–39*, *41*

sas_compress(), *40*
sas_config, 19
sas_config(), *22*, *29*
sas_display, *4*, *6*, *10–13*, *19*, 20, *22*, *26, 27*, *30–32*, *37–39*, *41*
sas_display(), *40*
sas_if_else, *4*, *6*, *10–13*, *19*, 21, *21*, *26, 27*, *30–32*, *37–39*, *41*
sas_if_else(), *40*
sas_length *(*sas_compress*)*, 18
sas_length(), *40*
sas_llm, 22
sas_llm(), *24, 25*
sas_llm_models, 23
sas_llm_models(), *22*, *25*
sas_llm_probe, 24
sas_llm_probe(), *22*
sas_max *(*sas_min*)*, 26
sas_max(), *40*
sas_mean *(*sas_sum*)*, 31
sas_mean(), *40*
sas_merge, *4*, *6*, *10–13*, *19*, *21, 22*, 25, *27*, *30–32*, *37–39*, *41*
sas_merge(), *39, 40*
sas_min, *4*, *6*, *10–13*, *19*, *21, 22*, 26, *26, 27*, *30–32*, *37–39*, *41*
sas_min(), *40*
sas_missing, *4*, *6*, *10–13*, *19*, *21, 22*, *26*, 27, *27*, *30–32*, *37–39*, *41*
sas_preflight, 28
sas_preflight(), *20*, *34*
sas_put *(*apply_format*)*, 3
sas_put(), *40*
sas_round, *4*, *6*, *10–13*, *19*, *21, 22*, *26, 27*, 30, *31, 32*, *37–39*, *41*
sas_round(), *40*
sas_sort, *4*, *6*, *10–13*, *19*, *21, 22*, *26, 27*, *30*, 31, *32*, *37–39*, *41*
sas_sort(), *39, 40*
sas_substr *(*sas_compress*)*, 18
sas_substr(), *40*
sas_sum, *4*, *6*, *10–13*, *19*, *21, 22*, *26, 27*, *30*, 31, *31*, *37–39*, *41*
sas_sum(), *40*
sas_translate, 32
sas_translate(), *20*, *22*, *28*, *41*
sas_true *(*sas_missing*)*, 27
sas_write, 35
sas_write(), *41*
write_comparison_report, 42