Package 'rmoriebricklayer'

September 8, 2026

Title Reproducible Data Capsules with Provenance and Fallback

Version 0.3.9

Description Tools for building brick-proof, reproducible, self-contained data capsules.
Resolves open-data sources through the Comprehensive Knowledge Archive Network ('CKAN', <https://ckan.org/>) package_show and package_search endpoints, records and verifies provenance with Secure Hash Algorithm 256 ('SHA-256') digests and Internet Archive 'Wayback Machine' (<https://web.archive.org/>) snapshots, validates downloaded data against a pinned schema, and falls back to schema-driven synthetic data when the real source is unreachable.
Run records are captured in a manifest plus a plain-language summary so any result can be traced back to its inputs. Also ships a small compiled C core (fast summary statistics and a self-contained 'SHA-256') that sibling packages in the 'rmorie' ecosystem reach through 'LinkingTo' for a single, shared numeric and provenance-hashing backend.

License AGPL-3

Encoding UTF-8

Depends R (>= 4.1.0)

Imports methods, stats, utils

Suggests digest, jsonlite, knitr, rmarkdown, stringi, testthat (>= 3.0.0)

VignetteBuilder knitr

URL https://github.com/rootcoder007/rmorie-bricklayer

BugReports https://github.com/rootcoder007/rmorie-bricklayer/issues

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3

SystemRequirements libcurl (deb: libcurl4-openssl-dev, rpm: libcurl-devel)

NeedsCompilation yes

Author Vansh Singh Ruhela [aut, cre] (ORCID:
<https://orcid.org/0009-0004-1750-3592>)

Maintainer Vansh Singh Ruhela <vsruhela@proton.me>

Repository CRAN

Date/Publication 2026-09-08 17:10:02 UTC

Contents

`agent_bundle` *Agent-assisted reproducibility-bundle help*

Description

Forwards a bundle-building request to the `rmorie` command-line agent (optional binary from rmorie-cli), with a rmoriebricklayer-focused preamble. The agent can run R and read/write files to help assemble or repair a brick-proof bundle. See `rmorie::agent` for requirements.

Usage

```
agent_bundle(request, model = NULL, backend = "auto")
```

Arguments

`request`	Character scalar describing the bundle task.
`model`	Optional model id, e.g. `"minimax-m3:cloud"` for an Ollama server or `"claude-sonnet-5"` for Anthropic; defaults to `$RMORIE_AGENT_MODEL` or the CLI's auto-pick.
`backend`	`"auto"` (default), `"ollama"` (server from `$RMORIE_AGENT_OLLAMA_URL`) or `"anthropic"` (`$RMORIE_AGENT_API_KEY`).

Value

Character scalar: the agent's output, or a message if the `rmorie` binary is not installed.

Examples

```
# Routed to the optional rmorie CLI agent when it is installed; with no
# binary on PATH each call returns an install hint instantly (no error,
# no network), so this is safe to execute anywhere.
agent_bundle("scaffold a bundle for analysis.R using the Toronto CKAN dataset")

# Free local/cloud route: an Ollama server (RMORIE_AGENT_OLLAMA_URL, default
# http://localhost:11434) serving the model the MORIE stack uses.
agent_bundle("add a Wayback fallback to my fetch step",
             backend = "ollama", model = "minimax-m3:cloud")

# Anthropic route (needs RMORIE_AGENT_API_KEY); the CLI's default model.
agent_bundle("repair the SHA256 provenance for my capsule",
             backend = "anthropic", model = "claude-sonnet-5")


# With no rmorie binary on PATH the call returns an install hint, not an
# error -- safe to run anywhere:
if (!nzchar(Sys.which("rmorie"))) agent_bundle("hello")
```

apply_schema_validation *Apply Schema Validation, Stopping on Fatal Issues*

Description

Runs `validate_schema()` and acts on the result: fatal issues raise an error via `stop()`, warning-severity issues emit a `warning()`.

Usage

```
apply_schema_validation(df_raw, provenance)
```

Arguments

`df_raw`	The data frame to validate.
`provenance`	A provenance list as returned by `load_provenance()`.

Value

Invisibly, `TRUE` if no issues were found and `FALSE` otherwise. Errors if any fatal issue is present.

Examples

```
prov <- list(schema = list(expected_columns = "id"))
apply_schema_validation(data.frame(id = 1), prov)
# A missing required column is fatal:
try(apply_schema_validation(data.frame(x = 1), prov))
```

ascii_fallback *Use Text As-Is, Falling Back to ASCII When It Cannot Be Represented*

Description

Returns `x` unchanged when it is valid, well-formed text (so legitimate UTF-8 such as an accented name is preserved), and only transliterates to plain ASCII via `to_ascii()` when the text is not valid UTF-8 (an encoding error) or when `force = TRUE` (for ASCII-only destinations such as a package `DESCRIPTION`). This lets author and supervisor names keep their accents wherever UTF-8 is supported while degrading gracefully instead of erroring where it is not.

Usage

```
ascii_fallback(x, force = FALSE)
```

Arguments

x	A character vector.
force	Logical; always transliterate to ASCII (default FALSE).

Value

A character vector: x where it can be represented, ASCII otherwise.

Examples

```
# By default valid UTF-8 is preserved (accents kept where supported).
ascii_fallback("\u00c1ngela")              # "\u00c1ngela"

# force = TRUE always transliterates (for ASCII-only destinations
# such as a package DESCRIPTION).
ascii_fallback("\u00c1ngela", force = TRUE)  # "Angela"

# Plain ASCII is returned unchanged either way.
ascii_fallback("plain name")

# Vectorised; each element handled independently.
ascii_fallback(c("caf\u00e9", "resume"), force = TRUE)
```

---

bricklayer_fetch *Fetch a URL to disk with an Internet Archive fallback (C++/libcurl)*

Description

The shared data-fetch foundation of the morie ecosystem. Downloads url to dest; if the live download fails (404, network), it retries the Wayback Machine snapshot – so a rotated or removed source file (CIHI, open-data portals) stays retrievable. Backed by the package's C++ rmbl_fetch_with_fallback kernel (libcurl), which **rmorie** and **morie** share through LinkingTo.

Usage

```
bricklayer_fetch(url, dest, wayback = "", timeout = 120L)
```

Arguments

url	Live source URL.
dest	Destination file path.
wayback	Optional explicit Wayback snapshot URL. "" (default) auto-resolves one via wayback_snapshot_url.
timeout	Per-request timeout, seconds.

Value

Invisibly, one of "live", "wayback", or throws on total failure.

Examples

```
# Inputs are validated before any network access:
try(bricklayer_fetch("", tempfile()))          # empty url -> error


# Downloads from the live web service; try() keeps the example graceful
# when neither the live URL nor its Wayback fallback is reachable.
dst <- tempfile(fileext = ".xlsx")

# Live download; auto-resolves a Wayback snapshot only if the live URL fails.
try(bricklayer_fetch(
 "https://www.cihi.ca/sites/default/files/document/hospital-beds-2024-2025-data-tables-en.xlsx",
  dst))

# Pin an explicit Wayback snapshot to fall back to, and a shorter timeout.
try(bricklayer_fetch(
  "https://example.org/rotated-file.csv", tempfile(fileext = ".csv"),
  wayback = "https://web.archive.org/web/2024id_/https://example.org/rotated-file.csv",
  timeout = 60))

# The return value tells you which source served the file.
status <- try(bricklayer_fetch("https://cloud.r-project.org/", tempfile()))
status   # "live" or "wayback"
```

---

bricklayer_fetch_parse_siu

Fetch and parse one SIU director's report

Description

Convenience: `bricklayer_fetch_siu()` then `bricklayer_parse_siu()`. Fails gracefully – returns `NULL` with a message when the report cannot be retrieved.

Usage

```
bricklayer_fetch_parse_siu(drid, lang = c("en", "fr"))
```

Arguments

`drid`	Director's-report id (the `drid=` query parameter).
`lang`	`"en"` (default) or `"fr"`.

Value

A named character vector of parsed fields, or `NULL` when the fetch fails.

Examples

```
f <- try(bricklayer_fetch_parse_siu(648), silent = TRUE)
if (is.character(f)) f[["police_service"]]
```

`bricklayer_fetch_siu` *Fetch an Ontario SIU director's report by drid*

Description

Downloads the HTML of one Ontario Special Investigations Unit (SIU) director's report to `dest`, via `bricklayer_fetch` (live URL with a Wayback Machine fallback). This is the fetch step of the open SIU corpus pipeline: pair it with the SIU parser in **rmorie** (or the standalone `siu` C++ package) to rebuild the full director's-report corpus yourself, then audit it with the multi-agent panel.

Usage

```
bricklayer_fetch_siu(
  drid,
  dest,
  lang = c("en", "fr"),
  wayback = "",
  timeout = 120L
)
```

Arguments

`drid`	Director's-report id – the `drid=` query parameter (integer or integer-like scalar).
`dest`	Destination file path for the fetched HTML.
`lang`	`"en"` (default) or `"fr"`.
`wayback`	Optional explicit Wayback snapshot URL (passed through to `bricklayer_fetch`); `""` lets it discover one.
`timeout`	Request timeout in seconds. Default 120.

Value

`"live"` or `"wayback"` (invisibly), as `bricklayer_fetch`.

See Also

`bricklayer_fetch`

Examples

```
# Downloads from the live SIU web service; try() keeps the example
# graceful when the service (and its Wayback fallback) is unreachable.
# Fetch report drid 648 to a temp file.
dest <- tempfile(fileext = ".html")
try(bricklayer_fetch_siu(648, dest))
```

`bricklayer_json_from_json` *Parse JSON into R objects (jsonlite's fromJSON, natively)*

Description

Parse JSON into R objects (jsonlite's fromJSON, natively)

Usage

```
bricklayer_json_from_json(
  txt,
  simplifyVector = TRUE,
  simplifyDataFrame = simplifyVector,
  simplifyMatrix = simplifyVector,
  flatten = FALSE,
  bigint_as_char = FALSE,
  simplify = NULL,
  ...
)
```

Arguments

`txt`	JSON text, a file path, or an http(s) URL.
`simplifyVector`, `simplifyDataFrame`, `simplifyMatrix`, `flatten`	as in jsonlite.
`bigint_as_char`	integers beyond 2^53 come back as strings.
`simplify`	legacy: `FALSE` turns every simplification off.
`...`	ignored, for call compatibility.

Value

an R object.

Examples

```
bricklayer_json_from_json('[{"a":1,"b":"x"},{"a":2,"b":"y"}]')
bricklayer_json_from_json('[[1,2],[3,4]]')
```

`bricklayer_json_to_json` *Encode an R object as JSON (jsonlite's toJSON, natively)*

Description

Every option of jsonlite's `toJSON()` with the same default and the same bytes out: `dataframe`, `matrix`, `Date`, `POSIXt`, `factor`, `complex`, `raw`, `null`, `na`, `auto_unbox`, `digits` (`I(n)` for significant digits, `NA` for 15), `pretty` (TRUE = 2 spaces, or a width), `force`, plus `rownames`, `keep_vec_names`, `json_verbatim`, `always_decimal`, `time_format`, `UTC`, `no_dots`, `hms`.

Usage

```
bricklayer_json_to_json(
  x,
  dataframe = c("rows", "columns", "values"),
  matrix = c("rowmajor", "columnmajor"),
  Date = c("ISO8601", "epoch"),
  POSIXt = c("string", "ISO8601", "epoch", "mongo"),
  factor = c("string", "integer"),
  complex = c("string", "list"),
  raw = c("base64", "hex", "mongo", "int", "js"),
  null = c("list", "null"),
  na = c("null", "string"),
  auto_unbox = FALSE,
  digits = 4,
  pretty = FALSE,
  force = FALSE,
  ...
)
```

Arguments

`x`	the object to encode.
`dataframe`, `matrix`, `Date`, `POSIXt`, `factor`, `complex`, `raw`, `null`, `na`, `auto_unbox`, `digits`, `pretty`, `force`, `...`	as in jsonlite.

Value

a length-one character vector of class `json`.

Examples

```
bricklayer_json_to_json(list(a = 1:3, b = "x"), auto_unbox = TRUE)
bricklayer_json_to_json(data.frame(id = 1:2, v = c(1.5, NA)), pretty = TRUE)
```

bricklayer_parse_siu *Parse an SIU director's report into the schema fields*

Description

Deterministic, offline extraction of every `bricklayer_siu_schema()` field (plus `_language`) from report HTML. Fields the report does not state come back as `""`.

Usage

```
bricklayer_parse_siu(html)
```

Arguments

`html`	A length-1 character vector of raw report HTML, or the path to a saved report file (e.g. from `bricklayer_fetch_siu()`).

Value

A named character vector: the 16 schema fields plus `_language`.

Examples

```
f <- bricklayer_parse_siu(system.file("extdata",
                                      "siu_synthetic_report.html",
                                      package = "rmoriebricklayer"))
f[["number_of_subject_officers"]]
```

bricklayer_siu_iso_date *Convert a human-readable SIU report date to ISO format*

Description

`"January 5, 2023"` (or `"January 5 2023"`) becomes `"2023-01-05"`; unparseable input becomes `""`.

Usage

```
bricklayer_siu_iso_date(x)
```

Arguments

`x`	A character vector of human-readable dates.

Value

A character vector of YYYY-MM-DD strings (or "").

Examples

```
bricklayer_siu_iso_date(c("January 5, 2023", "not a date"))
```

`bricklayer_siu_resolve_so` *Resolve the subject-official count from SIU report text*

Description

Deterministic, reproducible extraction of the subject-official (SO) count for reports where a model panel (or a human) is unsure. The standard SIU privacy boilerplate is stripped first, then rules apply most-specific first: highest SO #N ordinal; spelled-out plural; singular subject official present (1); witness-official-only (0, a real answer); otherwise unresolved (NA).

Usage

```
bricklayer_siu_resolve_so(text)
```

Arguments

text	A length-1 character vector of plain report text (see bricklayer_siu_text()).

Details

bricklayer is the foundation layer: this function is the pure rule set. Reports already in the panel-reviewed corpus should never be re-derived – use rmorie::morie_siu_resolve_so(), which returns the verified corpus value first and only falls back to these rules for unreviewed reports.

Value

A list with count (integer, NA when unresolved) and reason (the human-readable evidence).

Examples

```
bricklayer_siu_resolve_so(
  "Subject Officials\nSO #1 Interviewed\nSO #2 Declined interview")
```

bricklayer_siu_schema *The panel-reviewed SIU report field schema*

Description

The sixteen fields extracted from every Special Investigations Unit director's report. Count-type fields (`is_count = TRUE`) count distinct entities and zero is a real answer – a witness-official-only investigation has zero subject officials.

Usage

```
bricklayer_siu_schema()
```

Value

A data.frame with columns `name`, `is_count`, and `description`.

Examples

```
bricklayer_siu_schema()
```

bricklayer_siu_text *Convert SIU report HTML to plain text*

Description

Convert SIU report HTML to plain text

Usage

```
bricklayer_siu_text(html)
```

Arguments

`html`	A length-1 character vector of raw report HTML.

Value

A length-1 character vector of plain text.

Examples

```
bricklayer_siu_text("<p>Number of SIU Investigators assigned: 3</p>")
```

capture_environment *Capture the Analysis Environment for a Manifest*

Description

Records the facts a replicator needs to rebuild the session: R version, platform, operating system, a UTC timestamp, and the versions of the requested packages.

Usage

```
capture_environment(packages = loadedNamespaces())
```

Arguments

packages	Character vector of package names to record. Defaults to every currently loaded namespace.

Value

A list with r_version, platform, os, captured_utc, and packages (a named character vector of versions).

Examples

```
# Record specific packages' versions alongside the session facts.
env <- capture_environment(c("stats", "utils"))
env$r_version
env$os
env$packages          # named character vector of versions

# Default captures every currently loaded namespace.
names(capture_environment())[1:4]
```

cite_capsule *Generate a Data Citation From Provenance*

Description

Builds a ready-to-paste data citation (plain text and BibTeX @misc) from a provenance object's dataset and resource blocks, using publisher, resource name, source system, retrieval date, license, the pinned URL, and a DOI when one is recorded (dataset$doi).

Usage

```
cite_capsule(provenance)
```

Arguments

provenance	A provenance list as returned by load_provenance().

Value

A list with text and bibtex character scalars, or NULL if provenance is NULL.

Examples

```
prov <- list(
  captured_at_utc = "2026-06-23T04:41:40Z",
  dataset = list(publisher = "Ontario Ministry of the Solicitor General",
                 licence_short = "OGL-Ontario",
                 package_slug = "data-on-inmates-in-ontario"),
  resource = list(name = "Restrictive Confinement - Detailed Dataset",
                  direct_url = "https://data.ontario.ca/example.csv")
)
cit <- cite_capsule(prov)

# Plain-text citation ready to paste.
cat(cit$text)

# BibTeX @misc entry for LaTeX bibliographies.
cat(cit$bibtex)

# NULL provenance returns NULL (composes safely).
cite_capsule(NULL)
```

---

core_normal_pdf *Normal density (C backend)*

Description

Vectorised over x; mean and sd are length-1.

Usage

```
core_normal_pdf(x, mean = 0, sd = 1)
```

Arguments

x	Numeric vector of quantiles.
mean	Distribution mean (length-1, default 0).
sd	Distribution standard deviation (length-1, default 1, > 0).

Value

A numeric vector the length of x. Equivalent to stats::dnorm(x, mean, sd).

Examples

```
# Standard normal density at a few quantiles.
core_normal_pdf(c(-1, 0, 1))

# Peak of the standard normal is 1/sqrt(2*pi) at x = 0.
core_normal_pdf(0)

# Shift and scale via `mean` and `sd`.
core_normal_pdf(5, mean = 5, sd = 2)       # peak of N(5, 2)
core_normal_pdf(c(0, 5, 10), mean = 5, sd = 2)

# Identical to stats::dnorm().
all.equal(core_normal_pdf(-2:2, 0, 1), stats::dnorm(-2:2, 0, 1))
```

---

core_sha256 *SHA-256 hex digest (C backend)*

Description

Hashes character or raw input with the self-contained SHA-256 in the `rmoriebricklayer` core. For a character vector each element is hashed as its UTF-8/native bytes; for a raw vector the raw bytes are hashed. This is the same routine sibling packages use for provenance via `LinkingTo: rmoriebricklayer`.

Usage

```
core_sha256(x)
```

Arguments

`x`	A character vector or a raw vector.

Value

A character vector of 64-character lowercase hex digests (one per element for character input; length-1 for raw input).

Examples

```
# Hash a character scalar (NIST test vector for "abc").
core_sha256("abc")
# ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad

# Vectorised over character input: one digest per element.
core_sha256(c("abc", "def"))

# Raw input hashes the bytes directly; identical to the character form.
identical(core_sha256("abc"), core_sha256(charToRaw("abc")))
```

```
# Fingerprint an arbitrary object via its serialization.
core_sha256(serialize(list(a = 1L, b = "x"), NULL))
```

`download_data` *Download a File*

Description

Thin wrapper around `utils::download.file()` that returns the target path invisibly so it composes in pipelines.

Usage

```
download_data(url, target_path, mode = "wb", quiet = FALSE)
```

Arguments

`url`	URL to download.
`target_path`	Destination path on disk.
`mode`	Write mode passed to `utils::download.file()`; defaults to `"wb"` (binary) for cross-platform safety.
`quiet`	Logical; suppress progress output. Defaults to `FALSE`.

Value

The `target_path`, returned invisibly.

Examples

```
# try(): a live download must fail gracefully on an offline check machine.
dest <- try(download_data("https://cloud.r-project.org/",
                          tempfile(fileext = ".html"), quiet = TRUE))
if (!inherits(dest, "try-error")) file.exists(dest)
```

friendly_download *Download a File With Diagnostic Error Messages*

Description

Wraps `utils::download.file()` and, on failure, prints plain-language guidance for the most common academic and corporate network problems (rate limiting, TLS-inspection VPNs, DNS failures, timeouts, HTTP 403). Optionally retries from a Wayback Machine snapshot URL.

Usage

```
friendly_download(url, target_path, attempt_wayback = NULL)
```

Arguments

`url`	URL to download.
`target_path`	Destination path on disk.
`attempt_wayback`	Wayback Machine snapshot URL tried as a fallback if the primary download fails. When `NULL` (the default) a snapshot is resolved automatically via `wayback_snapshot_url()`; pass an explicit URL to override the lookup, or `""` to disable the fallback entirely.

Value

`TRUE` if either the primary download or the Wayback fallback succeeds, otherwise `FALSE`.

Examples

```
ok <- friendly_download("https://cloud.r-project.org/",
                        tempfile(fileext = ".html"),
                        attempt_wayback = "")  # disable the fallback
ok
```

load_provenance *Load a Pinned Data-Provenance Record*

Description

Reads a `data_provenance.json` file describing a project's pinned data source: the CKAN endpoint, resource name pattern, expected SHA256, Wayback snapshot, schema, and synthetic-data recipe.

Usage

```
load_provenance(path)
```

Arguments

path	Path to the provenance JSON file.

Value

The parsed provenance as a nested list (via jsonlite::fromJSON() with simplifyVector = FALSE), or NULL if the file does not exist.

Examples

```
prov_file <- tempfile(fileext = ".json")
writeLines('{"dataset": {"title": "demo"}, "sha256": "abc"}', prov_file)
prov <- load_provenance(prov_file)
prov$dataset$title
load_provenance(file.path(tempdir(), "no-such-file.json"))  # NULL
```

---

make_manifest *Construct a Reproducibility Manifest*

Description

Creates an empty manifest object that accumulates cross-check entries via record() and is later serialized with write_manifest_json().

Usage

```
make_manifest(meta, environment = TRUE)
```

Arguments

meta	A named list of run metadata (e.g. project, author, run_at, synthetic).
environment	Logical; when TRUE (the default) the manifest also records the analysis environment via capture_environment() (R version, platform, OS, UTC timestamp, loaded package versions).

Value

A manifest list with elements meta, an empty results list, and (when requested) environment.

Examples

```
# Minimal manifest, no environment capture.
man <- make_manifest(list(project = "demo-study", author = "A. Author"),
                     environment = FALSE)
names(man)          # "meta" "results"
man$meta$project

# With environment = TRUE it also records R version / platform / packages.
```

```
full <- make_manifest(list(project = "demo"), environment = TRUE)
names(full)         # adds "environment"
full$environment$r_version
```

`make_synthetic_column` *Generate One Synthetic Column From a Spec*

Description

Builds a single synthetic data column according to a column spec drawn from a provenance synthetic recipe. Supported `type`s are `"sample"` (categorical, optionally weighted), `"bernoulli"` (two-label draw with optional per-row base rate), `"poisson"` (counts with a floor), `"id_pattern"` (templated IDs, optionally per-year sequenced), and `"sequence"` (a running integer sequence).

Usage

```
make_synthetic_column(spec, n, ctx = list(), base_p = NULL)
```

Arguments

`spec`	A list describing the column; recognised fields depend on `spec$type` (e.g. `values`, `weights`, `p`, `p_with_baserate`, `labels`, `lambda`, `min`, `pattern`, `year_col`, `from`).
`n`	Number of values to generate.
`ctx`	Named list of already-generated columns, letting later columns (such as `id_pattern` with a `year_col`) reference earlier ones. Defaults to an empty list.
`base_p`	Optional numeric vector of per-row latent propensities used by the `"bernoulli"` type to add row-level variation.

Value

A vector of length `n` for the requested column type. Errors on an unknown `type`.

Examples

```
set.seed(1)
# "sample": categorical draw, optionally weighted.
make_synthetic_column(list(type = "sample", values = list("a", "b"),
                           weights = list(0.7, 0.3)), 5)

# "bernoulli": two-label draw at probability p.
make_synthetic_column(list(type = "bernoulli", p = 0.5,
                           labels = list("Yes", "No")), 5)

# "poisson": counts with a floor via `min`.
make_synthetic_column(list(type = "poisson", lambda = 3, min = 1), 5)

# "id_pattern": templated IDs (the {seq:05d} token is zero-padded).
```

```
make_synthetic_column(list(type = "id_pattern",
                           pattern = "case-{seq:05d}"), 3)

# "sequence": a running integer sequence from `from`.
make_synthetic_column(list(type = "sequence", from = 100), 4)

# An unknown type errors.
try(make_synthetic_column(list(type = "nope"), 3))
```

`make_synthetic_csv` *Generate a Synthetic CSV From a Schema Recipe*

Description

Generates a reproducible synthetic data set from the `schema$synthetic_recipe` block of a provenance object and writes it to a CSV. Columns are produced in declaration order so later columns can reference earlier ones, a shared per-row latent propensity drives any Bernoulli columns, and an optional row-replication block expands per-person rows.

Usage

```
make_synthetic_csv(schema, out_path, n_rows = NULL, seed = NULL)
```

Arguments

`schema`	The synthetic recipe (a list with `columns`, and optional `n_rows`, `seed`, and `row_replication`).
`out_path`	Path where the CSV is written.
`n_rows`	Number of rows (persons, if replicating) to generate. Defaults to `schema$n_rows`, then 50000.
`seed`	Random seed for reproducibility. Defaults to `schema$seed`, then 91735246.

Value

Invisibly, a list with `path`, `rows` (rows written), and `seed` used.

Examples

```
recipe <- list(
  n_rows = 20, seed = 42,
  columns = list(
    year   = list(type = "sample", values = list(2024, 2025)),
    alert  = list(type = "bernoulli", p = 0.2),
    visits = list(type = "poisson", lambda = 3, min = 1),
    id     = list(type = "id_pattern", pattern = "p-{seq:05d}")
  )
)
out <- tempfile(fileext = ".csv")
res <- make_synthetic_csv(recipe, out)
```

```
res$rows                        # 20
res$seed                        # 42 (reproducible)

# The written CSV round-trips and has the declared columns.
df <- utils::read.csv(out)
dim(df)
names(df)

# `n_rows` overrides the recipe's own row count.
make_synthetic_csv(recipe, tempfile(fileext = ".csv"), n_rows = 5)$rows
```

`record` *Record a Cross-Check Result in a Manifest*

Description

Appends one named cross-check entry to a manifest, classifying it as `PASS`, `DIFFER`, or `INFO`, printing a formatted line to the console, and returning the updated manifest.

Usage

```
record(
  manifest,
  name,
  observed,
  expected,
  tol = 1e-04,
  group = "general",
  synthetic = FALSE
)
```

Arguments

`manifest`	A manifest as returned by `make_manifest()`.
`name`	Unique name for this cross-check; used as the result key.
`observed`	The observed value (numeric or otherwise).
`expected`	The expected value to compare against.
`tol`	Numeric tolerance; a numeric pair within `tol` is `PASS`. Defaults to `0.0001`.
`group`	Optional grouping label for the entry. Defaults to `"general"`.
`synthetic`	Logical; if `TRUE` the entry is marked `INFO` because comparison against synthetic data is not meaningful.

Value

The updated manifest, returned so calls can be chained.

Examples

```
man <- make_manifest(list(project = "demo"), environment = FALSE)

# Within tolerance -> PASS.
man <- record(man, "mean_matches", observed = 1.0001, expected = 1,
              tol = 0.001)
man$results$mean_matches$status      # "PASS"

# Outside tolerance -> DIFFER.
man <- record(man, "sd_matches", observed = 2.5, expected = 2.0, tol = 0.01)
man$results$sd_matches$status        # "DIFFER"

# Synthetic data -> INFO (comparison not meaningful).
man <- record(man, "synthetic_row", observed = 5, expected = 5,
              synthetic = TRUE)
man$results$synthetic_row$status     # "INFO"

# Calls chain: record() returns the mutated manifest.
length(man$results)                  # 3
```

`resolve_via_arcgis` *Resolve a Query URL via ArcGIS FeatureServer Metadata*

Description

Verifies that an ArcGIS FeatureServer layer still exists by fetching its `f=json` metadata, then returns a paged GeoJSON query URL for the full layer. ArcGIS FeatureServer layers back the Toronto Police Service open-data portal used across the MORIE family.

Usage

```
resolve_via_arcgis(provenance)
```

Arguments

`provenance`	A provenance list as returned by `load_provenance()`. Must contain `dataset$arcgis_layer_url`, a FeatureServer layer root such as `"https://services.arcgis.com/.../Assault_Open_Data/Featur`

Value

The layer query URL (`where=1=1`, all fields, GeoJSON) as a character string, or `NULL` if the field is missing, the request fails, or the layer metadata reports an error.

Examples

```
# Missing fields return NULL rather than erroring:
resolve_via_arcgis(list())

prov <- list(dataset = list(arcgis_layer_url = paste0(
```

```
  "https://services.arcgis.com/S9th0jAJ7bqgIRjw/arcgis/rest/services/",
  "Neighbourhood_Crime_Rates_Open_Data/FeatureServer/0")))
resolve_via_arcgis(prov)
```

`resolve_via_ckan` *Resolve a Download URL via CKAN package_show*

Description

Queries the CKAN `package_show` endpoint recorded in a provenance object and returns the URL of the first resource whose name matches the provenance's name-match pattern. CKAN powers data.ontario.ca, data.gov.uk, data.gov, and most government open-data portals, so this recovers the current download URL even if the underlying resource UUID has been replaced.

Usage

```
resolve_via_ckan(provenance)
```

Arguments

`provenance`	A provenance list as returned by `load_provenance()`. Must contain `dataset$ckan_api_endpoint` and `resource$name_match_pattern`.

Value

The matched resource URL as a character string, or `NULL` if the endpoint is missing, the request fails, CKAN reports failure, or no resource name matches.

Examples

```
# Missing fields return NULL rather than erroring:
resolve_via_ckan(list())

prov <- list(
  dataset  = list(ckan_api_endpoint = paste0(
    "https://data.ontario.ca/api/3/action/package_show",
    "?id=ontario-public-library-statistics")),
  resource = list(name_match_pattern = "2014")
)
resolve_via_ckan(prov)
```

resolve_via_ckan_search *Resolve a Download URL via CKAN package_search*

Description

Fallback for `resolve_via_ckan()` when the dataset slug has changed. Derives the CKAN portal base URL from the provenance's `package_show` endpoint, runs a `package_search` query (from `resource$search_query`, or derived from the name-match pattern), and returns the URL of the first matching resource, preferring CSV format when specified.

Usage

```
resolve_via_ckan_search(provenance)
```

Arguments

provenance	A provenance list as returned by `load_provenance()`. Uses `resource$search_query`, `resource$name_match_pattern`, `resource$format`, and `dataset$ckan_api_endpoint`.

Value

The matched resource URL as a character string, or `NULL` if no query or base URL can be derived, the request fails, or nothing matches.

Examples

```
# Missing fields return NULL rather than erroring:
resolve_via_ckan_search(list())

prov <- list(
  dataset  = list(ckan_api_endpoint = paste0(
    "https://data.ontario.ca/api/3/action/package_show",
    "?id=ontario-public-library-statistics")),
  resource = list(name_match_pattern = "2014",
                  search_query = "public library statistics")
)
resolve_via_ckan_search(prov)
```

resolve_via_socrata *Resolve a Download URL via the Socrata Metadata API*

Description

Verifies that a Socrata dataset still exists by fetching its `api/views` metadata, then returns the canonical CSV export URL. Socrata powers the Calgary, Chicago, and NYC open-data portals used across the MORIE family.

Usage

```
resolve_via_socrata(provenance)
```

Arguments

`provenance`	A provenance list as returned by `load_provenance()`. Must contain `dataset$socrata_domain` (e.g. `"data.cityofchicago.org"`) and `dataset$socrata_id` (the 4x4 dataset id, e.g. `"ijzp-q8t2"`).

Value

The CSV export URL as a character string, or `NULL` if the fields are missing, the request fails, or the metadata reports an error.

Examples

```
# Missing fields return NULL rather than erroring:
resolve_via_socrata(list())

prov <- list(dataset = list(socrata_domain = "data.cityofchicago.org",
                            socrata_id     = "ijzp-q8t2"))
resolve_via_socrata(prov)
```

rmbl_core_stats *Fast summary statistics (C backend)*

Description

Thin R wrappers over the `rmoriebricklayer` compiled core – the same kernels that sibling packages reach through `LinkingTo: rmoriebricklayer`. NA/NaN values propagate (there is no `na.rm`); call `stats::na.omit()` first if you need NA handling.

Usage

```
core_mean(x)

core_var(x)

core_cor(x, y)
```

Arguments

x, y	Numeric vectors (coerced with `as.numeric()`).

Value

`core_mean()`, `core_var()` and `core_cor()` return a length-1 numeric. `core_var()` uses the `n - 1` (sample) denominator, matching `stats::var()`.

Examples

```
## core_mean(): sample mean (NA/NaN propagate; no na.rm)
core_mean(1:10)                 # 5.5
core_mean(c(2.5, 3.5))          # 3
core_mean(c(1, 2, NA))          # NA -- call stats::na.omit() first if needed
core_mean(stats::na.omit(c(1, 2, NA)))

## core_var(): n-1 (sample) variance, matching stats::var()
core_var(c(2, 4, 4, 4, 5, 5, 7, 9))
all.equal(core_var(1:10), stats::var(1:10))   # agrees with base R

## core_cor(): Pearson correlation of two equal-length vectors
core_cor(1:10, (1:10)^2)        # strong positive, near 0.97
core_cor(1:10, 10:1)            # perfect negative: -1
```

---

sha256_file *Compute a File's SHA256 Digest*

Description

Returns the SHA256 digest of a file as a lowercase hex string, using the digest package. Used to record and verify data provenance.

Usage

```
sha256_file(path)
```

Arguments

path	Path to the file to hash.

Value

The SHA256 digest as a character string.

Examples

```
f <- tempfile()
writeLines("hello capsule", f)
sha256_file(f)

# Deterministic: the same bytes always yield the same digest.
identical(sha256_file(f), sha256_file(f))

# Any change to the file changes the digest (tamper-evidence).
before <- sha256_file(f)
writeLines("hello capsule (edited)", f)
after <- sha256_file(f)
before == after      # FALSE

# Provenance pin: record a digest, verify it later.
pinned <- sha256_file(f)
stopifnot(sha256_file(f) == pinned)
```

---

to_ascii *Transliterate Text to Plain ASCII*

Description

Converts a character vector to plain 7-bit ASCII, transliterating accented or non-Latin characters to their nearest ASCII equivalent (for example, an accented capital A becomes a plain "A"). Falls back to dropping any character that has no transliteration. This is the deterministic "fallback" used by `ascii_fallback()`.

Usage

```
to_ascii(x)
```

Arguments

x	A character vector.

Value

A character vector containing only ASCII characters.

Examples

```
# Latin accents fold to their nearest ASCII letter.
to_ascii("Prof. \u00c1ngela Zorro Medina")  # "Prof. Angela Zorro Medina"

# Vectorised over the input.
to_ascii(c("Se\u00e1n", "Zo\u00eb", "na\u00efve"))

# Non-Latin scripts are romanised when stringi is available.
if (requireNamespace("stringi", quietly = TRUE))
  to_ascii("\u041c\u043e\u0441\u043a\u0432\u0430")  # "Moskva" (Cyrillic)

# Either way the result is guaranteed pure 7-bit ASCII (never "?").
all(charToRaw(to_ascii("caf\u00e9")) < 128)
```

`validate_schema` *Validate a Data Frame Against a Provenance Schema*

Description

Checks a raw data frame against the `schema` block of a provenance object: required columns, row-count bounds, and allowed categorical value sets. Returns the issues found rather than raising, so the caller decides how to react.

Usage

```
validate_schema(df_raw, provenance)
```

Arguments

`df_raw`	The data frame to validate.
`provenance`	A provenance list as returned by `load_provenance()`. The `schema` block may contain `expected_columns`, `structural_invariants` (`min_data_rows`, `max_data_rows`), and `expected_value_sets` (a named list of allowed values per column).

Value

A named list of issues; each issue is a list with `severity` (`"fatal"` or `"warning"`) and a human-readable `message`. A zero-length list means the data frame is clean.

Examples

```
prov <- list(schema = list(
  expected_columns      = c("id", "year"),
  structural_invariants = list(min_data_rows = 1),
  expected_value_sets   = list(year = 2020:2025)
))
df <- data.frame(id = 1:3, year = c(2020, 2021, 2030))
issues <- validate_schema(df, prov)
names(issues)  # flags the out-of-set year value
```

verify_capsule *Re-Verify an Entire Reproducible Data Capsule*

Description

Runs the full custody chain over a capsule directory in one call: the provenance manifest is readable, the pinned data file exists and matches its recorded `sha256` (and `size_bytes` / row count where recorded), the schema still validates, a recorded analysis script still matches its pinned hash, and every numeric cross-check stored in a results manifest still reproduces its recorded `PASS`/`DIFFER` status from its own `observed`/`expected`/`tol` fields.

Usage

```
verify_capsule(
  capsule_dir,
  provenance_file = "data_provenance.json",
  data_file = NULL,
  manifest_file = NULL,
  script_file = NULL
)
```

Arguments

`capsule_dir`	Directory containing the capsule.
`provenance_file`	Provenance JSON filename inside `capsule_dir` (default `"data_provenance.json"`).
`data_file`	Data filename inside `capsule_dir`. Defaults to the provenance's `resource$filename`.
`manifest_file`	Optional results-manifest JSON (as written by `write_manifest_json()`) inside `capsule_dir`; checked when present.
`script_file`	Optional analysis-script filename inside `capsule_dir`; compared against the manifest's recorded `meta$script_sha256` when both are present.

Details

Entirely offline: nothing is downloaded and nothing is written.

Value

A list with `ok` (logical scalar: every check passed) and `checks` (data.frame with columns `check`, `ok`, `detail`).

Examples

```
dir <- file.path(tempdir(), "capsule-example")
dir.create(dir, showWarnings = FALSE)
write.csv(data.frame(id = 1:3), file.path(dir, "d.csv"), row.names = FALSE)
prov <- list(resource = list(filename = "d.csv",
```

```
                              sha256 = sha256_file(file.path(dir, "d.csv"))))
jsonlite::write_json(prov, file.path(dir, "data_provenance.json"),
                     auto_unbox = TRUE)
verify_capsule(dir)$ok
```

`verify_sha256` *Verify a File's SHA256 Against an Expected Digest*

Description

Computes the SHA256 digest of a file (via the digest package) and compares it to the expected value pinned in provenance.

Usage

```
verify_sha256(path, expected_sha)
```

Arguments

`path`	Path to the file to hash.
`expected_sha`	The expected SHA256 digest, as a lowercase hex string.

Value

A list with `actual` (computed digest), `expected` (the value passed in), and `match` (logical; `TRUE` if they are identical).

Examples

```
f <- tempfile()
writeLines("hello capsule", f)

# Matching digest -> match TRUE.
chk <- verify_sha256(f, sha256_file(f))
chk$match          # TRUE

# A wrong expected digest -> match FALSE, with both values reported.
bad <- verify_sha256(f, "0000000000000000000000000000000000000000000000000000000000000000")
bad$match          # FALSE
bad$actual         # the real digest
```

wayback_snapshot_url *Resolve a Wayback Machine snapshot URL*

Description

Queries the Internet Archive availability API (`http://archive.org/wayback/available`) for the closest archived snapshot of `url` and returns a directly-downloadable snapshot URL, or `NULL` if no snapshot exists or the lookup fails. This is the shared fetch failsafe the wider morie package family relies on: callers attempt the live source first and fall back to this snapshot when the source is unreachable.

Usage

```
wayback_snapshot_url(url, timestamp = NULL)
```

Arguments

`url`	The original source URL to look up.
`timestamp`	Optional 14-digit `YYYYMMDDhhmmss` target; the API returns the snapshot closest to it. Defaults to the most recent.

Value

A character scalar snapshot URL, or `NULL`.

Examples

```
wayback_snapshot_url("https://www.r-project.org/")
```

wayback_snapshot_url_native *Resolve a Wayback Machine snapshot URL (C++/libcurl)*

Description

Queries the Internet Archive "available" API for the closest archived snapshot of `url`. C++ backend; supersedes the older jsonlite-based resolver (kept internally for offline use).

Usage

```
wayback_snapshot_url_native(url, timeout = 30L)
```

Arguments

`url`	URL to resolve.
`timeout`	Request timeout, seconds.

Value

The https snapshot URL, or NULL if none is archived.

Examples

```
# Input is validated before any network call:
try(wayback_snapshot_url_native(""))     # empty url -> error


# Uses the live Wayback service; degrades gracefully offline: any
# network failure returns NULL rather than erroring.
# Closest archived snapshot of a live page (or NULL if none archived).
wayback_snapshot_url_native("https://www.r-project.org/")

# A shorter timeout for a quick lookup.
wayback_snapshot_url_native("https://cloud.r-project.org/", timeout = 10)

# A never-archived URL returns NULL rather than erroring.
wayback_snapshot_url_native("https://example.invalid/never-archived")
```

---

write_manifest_json *Write a Manifest to JSON*

Description

Serializes a manifest to a pretty-printed JSON file with the native JSON codec (bricklayer_json_to_json()); no jsonlite needed.

Usage

```
write_manifest_json(manifest, path)
```

Arguments

manifest	A manifest as returned by make_manifest() / built up with record().
path	Destination path for the JSON file.

Value

The path, returned invisibly.

Examples

```
man <- make_manifest(list(project = "demo"), environment = FALSE)
man <- record(man, "row_count", observed = 20, expected = 20)
path <- write_manifest_json(man, tempfile(fileext = ".json"))
file.exists(path)

# Round-trips back through jsonlite.
back <- bricklayer_json_from_json(path, simplifyVector = FALSE)
back$results$row_count$status        # "PASS"
```

`write_summary_txt` *Write a Plain-Language Run Summary*

Description

Writes a human-readable `SUMMARY.txt` into the output directory, covering run metadata, the exact absolute paths used, result counts, the files produced, and optional notes, contact, and licence lines.

Usage

```
write_summary_txt(
  manifest,
  output_dir,
  paths,
  what_was_done = NULL,
  contact = NULL,
  licence = NULL
)
```

Arguments

`manifest`	A manifest as returned by `make_manifest()`; its `meta` supplies project/author/run details.
`output_dir`	Directory to write `SUMMARY.txt` into and to list produced files from.
`paths`	A named list of absolute paths to report (e.g. `capsule`, `input`, `results`, `analysis_script`, `provenance`).
`what_was_done`	Optional character vector of bullet points describing what the run did.
`contact`	Optional contact string appended to the summary.
`licence`	Optional licence string appended to the summary.

Value

The path to the written `SUMMARY.txt`, returned invisibly.

Examples

```
man <- make_manifest(list(project = "demo", author = "A. Author"),
                     environment = FALSE)
man <- record(man, "row_count", observed = 20, expected = 20)
out <- file.path(tempdir(), "demo-run")
dir.create(out, showWarnings = FALSE)
s <- write_summary_txt(man, out, paths = list(results = out))
readLines(s)[7:8]
```

`write_text_fallback` *Write Text as UTF-8, Falling Back to ASCII on an Encoding Error*

Description

Writes `text` to `path` as UTF-8. If the write raises an encoding error (for example a destination or locale that cannot represent the characters), it retries with an ASCII transliteration produced by `to_ascii()` so capsule generation never fails on a non-ASCII name.

Usage

```
write_text_fallback(text, path)
```

Arguments

`text`	A character vector of lines to write.
`path`	Destination file path.

Value

Invisibly, `path`.

Examples

```
p <- write_text_fallback(c("line one", "line two"),
                         tempfile(fileext = ".txt"))
readLines(p)
```

Index