Package: reflowR (via r-universe)

November 3, 2026

Title Themed Extensions for 'workflowr' Research Workflows

Version 0.2.0

Description Extends the 'workflowr' package with themed colour schemes, structured analysis pipeline templates and utility scripts for reproducible research. Wraps wflow_start() from 'workflowr' and adds the nine topic families of the Computational Trauma and Tissue Injury Research (CTTIR) v5 design language as site schemes (Okabe-Ito accents, light and dark surfaces, Inter and JetBrains Mono typography from shared design tokens), three depth presets for the R Markdown pipeline (minimal, standard, extended), CSS/SCSS theming, SVG family icons, 'ggplot2' theme functions and a 'Quarto' report template. All generated projects remain fully compatible with standard 'workflowr' commands. Further interfaces run file-producing analysis steps and imaging workflows of installed package functions as fingerprinted, resumable pipelines with verified receipts, and scaffold and validate strict project profiles.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)

Depends R (>= 4.1.0)

Imports workflowr (>= 1.7.0), cli, digest, fs, glue, rlang, ggplot2 (>= 3.4.0), base64enc, git2r, jsonlite

Suggests codetools, processx (>= 3.9.0), ps (>= 1.9.3), rstudioapi, testthat (>= 3.0.0), knitr, rmarkdown, withr, targets, yaml

VignetteBuilder knitr

Config/testthat/edition 3

Config/Needs/website CTTIR/themakR, pkgdown

URL https://github.com/CTTIR/reflowR, https://cttir.github.io/reflowR/

BugReports https://github.com/CTTIR/reflowR/issues

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake make libgit2-dev libicu-dev libssh2-1-dev libuv1-dev libssl-dev zlib1g-dev

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

Date/Publication 2026-10-29 11:19:01 UTC

RemoteUrl https://github.com/CTTIR/reflowR

RemoteRef HEAD

RemoteSha 5ce581f5e96696f336081a2c2360bbd528f58dad

Contents

reflow_artifact_plan . . . 3
reflow_artifact_plan_run . . . 4
reflow_artifact_plan_run_bounded . . . 5
reflow_artifact_ref . . . 7
reflow_artifact_run . . . 8
reflow_artifact_run_bounded . . . 9
reflow_artifact_select . . . 12
reflow_artifact_selection_verify . . . 13
reflow_artifact_spec . . . 14
reflow_artifact_stage . . . 15
reflow_artifact_target . . . 16
reflow_call_stage . . . 17
reflow_call_writer . . . 18
reflow_figure_plot . . . 19
reflow_figure_recipe . . . 20
reflow_icon_base64 . . . 23
reflow_imaging_plan . . . 23
reflow_imaging_ref . . . 24
reflow_imaging_run . . . 25
reflow_imaging_stage . . . 26
reflow_init . . . 27
reflow_init_strict . . . 29
reflow_preset . . . 31
reflow_presets . . . 31
reflow_preview . . . 32
reflow_profile . . . 32
reflow_scaffold . . . 34
reflow_scheme . . . 35
reflow_schemes . . . 36
reflow_validate_strict . . . 36

Index **38**

`reflow_artifact_plan` *Declare a serial artifact dependency plan*

Description

Declaration order is immaterial: a stable identifier order breaks ties among available stages. This dependency core is not a resource scheduler or a targets adapter. Existing files use `reflow_imaging_input()`; no external execution is inferred from an input path. Plain strings are configuration.

Usage

```
reflow_artifact_plan(...)
```

Arguments

`...`	Deferred `reflow_artifact_stage()` declarations.

Value

A canonical acyclic plan, without executing writers.

Examples

```
plan <- reflow_artifact_plan(
  reflow_artifact_stage(
    "report", "base", "writeLines",
    args = list(text = reflow_artifact_ref("data", "labels", read = TRUE)),
    outputs = list(con = list(path = "report.txt", type = "file")),
    inventory = c(report = "report.txt")
  ),
  reflow_artifact_stage(
    "data", "base", "saveRDS",
    args = list(object = c("alpha", "beta")),
    outputs = list(file = list(path = "labels.rds", type = "file")),
    inventory = c(labels = "labels.rds")
  )
)
# Producers come before their consumers, whatever the declaration order
names(plan$stages)
plan$dependencies
```

`reflow_artifact_plan_run` *Execute or verify an immutable artifact graph generation*

Description

A new generation can reference unchanged nodes in a completed prior generation without copying their files. A node is reused only when its complete resolved definition and producer lineage match. Changed inputs never overwrite a prior generation. Resume requires the same plan, inputs and runtime and rechecks accepted artifacts before dispatching any writer.

Usage

```
reflow_artifact_plan_run(plan, directory, previous = NULL)

reflow_artifact_plan_resume(plan, directory, reconciliations = list())
```

Arguments

`plan`	A `reflow_artifact_plan()`.
`directory`	Fresh generation directory, or existing one for resume.
`previous`	Optional completed generation to verify for in-place reuse.
`reconciliations`	Named list keyed by failed stage identifiers. Each element has exactly `reconciled_attempt` and `reconciliation`, as in `reflow_artifact_resume()`. No process absence is inferred automatically.

Details

Execution is serial and trusted, not sandboxed or resource supervised. Stale locks require external inspection; they are never stolen. Failed child attempts retain the explicit reconciliation contract of `reflow_artifact_resume()`. Atomic writes provide rename visibility, not power-loss durability. Keep prior generations available while referenced.

Value

A graph descriptor containing authenticated node descriptors.

Examples

```
plan <- reflow_artifact_plan(
  reflow_artifact_stage(
    "data", "base", "saveRDS",
    args = list(object = c("alpha", "beta")),
    outputs = list(file = list(path = "labels.rds", type = "file")),
    inventory = c(labels = "labels.rds")
  ),
```

```
  reflow_artifact_stage(
    "report", "base", "writeLines",
    args = list(text = reflow_artifact_ref("data", "labels", read = TRUE)),
    outputs = list(con = list(path = "report.txt", type = "file")),
    inventory = c(report = "report.txt")
  )
)
root <- tempfile("graph-", tmpdir = normalizePath(tempdir(), winslash = "/", mustWork = TRUE))
dir.create(root)
graph <- reflow_artifact_plan_run(plan, file.path(root, "first"))
readLines(file.path(graph$nodes$report$descriptor$bundle, "report.txt"))

# Resume checks the accepted files again without calling any writer
identical(reflow_artifact_plan_resume(plan, file.path(root, "first")), graph)
unlink(root, recursive = TRUE)
```

`reflow_artifact_plan_run_bounded` *Run or resume a serial graph with per-node Linux guardians*

Description

Validate all stage resource policies before any writer, execute pending nodes serially and verify cache hits without new supervisors. Scientific definitions and synchronous graph APIs are unchanged.

Usage

```
reflow_artifact_plan_run_bounded(plan, directory, supervisor_directory,
  resources, runtime_files, previous = NULL, cancel = function() FALSE)
reflow_artifact_plan_resume_bounded(plan, directory, supervisor_directory,
  resources, runtime_files, cancel = function() FALSE, reconciliations = list())
```

Arguments

`plan`	An artifact dependency plan.
`directory`	Graph generation directory.
`supervisor_directory`	Separate policy/control root with an existing parent.
`resources`	Exact named list in canonical stage order; each element has the resource fields documented by `reflow_artifact_run_bounded`.
`runtime_files`	Common named SHA-256 vector covering all writer closures and backend files.
`previous`	Optional completed prior generation for verified in-place reuse.
`cancel`	Function returning one nonmissing logical cancellation value.
`reconciliations`	Explicit per-stage artifact retry reconciliations. Never clears guardian locks.

Details

Control policy and attempt journals are separate from graph inventory. Orphan READY adoption requires a matching successful guardian outcome and fresh locked verification. Missing journals, failed guardian outcomes and changed policies fail closed. Ordinary parent unwind releases its graph/control locks; stale locks after process death require external inspection. Guardian death is unqualified and requires external cleanup. Linux only, trusted descendants, per-process address space, requested threads, and explicit finite deadlines of 1–86400 seconds per node; no graph-wide deadline, parallel scheduler, selector or targets integration.

Value

Verified graph descriptor.

See Also

reflow_artifact_run_bounded, reflow_artifact_plan_run

Examples

```
# Needs Linux with processx, ps, prlimit and nice, and a package library
# without symbolic links. R CMD check links the packages into a temporary
# library when it limits them to the declared dependencies
# (_R_CHECK_SUGGESTS_ONLY_, which --as-cran turns on, or
# _R_CHECK_DEPENDS_ONLY_). A run starts a guardian and a worker process and
# pins every installed file they load, which takes a while.

roots <- c("reflowR", "processx", "ps", "digest", "base")
usable <- identical(Sys.info()[["sysname"]], "Linux") &&
  requireNamespace("processx", quietly = TRUE) &&
  requireNamespace("ps", quietly = TRUE) && ps::ps_is_supported() &&
  nzchar(Sys.which("prlimit")) && nzchar(Sys.which("nice"))
if (usable) {
  # The worker runs at priority 10 with 8 GiB of address space, which an
  # unprivileged session can only request when its own settings allow it.
  space <- grep("^Max address space", readLines("/proc/self/limits"), value = TRUE)
  space <- strsplit(trimws(sub("^Max address space", "", space)), " +")[[1]][1:2]
  usable <- ps::ps_get_nice(ps::ps_handle()) <= 10 && isTRUE(all(
    space == "unlimited" | suppressWarnings(as.numeric(space)) >= 8 * 1024^3
  ))
}
if (usable) {
  # reflowR, processx, ps, digest, the writer package (base) and their imports
  packages <- unique(c(roots, unlist(tools::package_dependencies(
    roots, db = utils::installed.packages(), which = c("Depends", "Imports"),
    recursive = TRUE
  ))))
  usable <- !any(nzchar(Sys.readlink(find.package(packages))))
}
if (usable) {
 root <- tempfile("bounded-", tmpdir = normalizePath(tempdir(), winslash = "/", mustWork = TRUE))
  work <- file.path(root, c("tmp", "cache"))
```

```
  for (path in c(root, work)) dir.create(path)
  plan <- reflow_artifact_plan(reflow_artifact_stage(
    "lines", "base", "writeLines",
    args = list(text = c("alpha", "beta")),
    outputs = list(con = list(path = "lines.txt", type = "file")),
    inventory = c(lines = "lines.txt")
  ))
  resources <- list(
    timeout_seconds = 60, address_space_bytes = 8 * 1024^3, nice = 10L,
    threads = 1L, temp_directory = work[1], cache_directory = work[2],
    rscript = normalizePath(file.path(R.home("bin"), "Rscript")),
    prlimit = normalizePath(Sys.which("prlimit")),
    nice_command = normalizePath(Sys.which("nice"))
  )
  # Pin the backend scripts, every installed file of those packages and the
  # three executables.
  files <- c(
    list.files(system.file("bounded", package = "reflowR"), full.names = TRUE),
    unlist(lapply(find.package(packages), list.files, recursive = TRUE,
                  full.names = TRUE, all.files = TRUE)),
    resources$rscript, resources$prlimit, resources$nice_command
  )
  files <- unique(normalizePath(files, winslash = "/"))
  pins <- vapply(files, digest::digest, character(1), algo = "sha256", file = TRUE)

  # The parent thread settings must equal the requested thread count; they
  # are restored afterwards, also when the run fails.
  threads <- c("OPENBLAS_NUM_THREADS", "OMP_NUM_THREADS", "MKL_NUM_THREADS")
  saved <- Sys.getenv(threads, unset = NA)
  Sys.setenv(OPENBLAS_NUM_THREADS = 1, OMP_NUM_THREADS = 1, MKL_NUM_THREADS = 1)
  tryCatch({
    # One resource list per stage, in the canonical stage order
    graph <- reflow_artifact_plan_run_bounded(plan, file.path(root, "graph"),
                                              file.path(root, "supervisor"),
                                              list(lines = resources), pins)
    print(readLines(file.path(graph$nodes$lines$descriptor$bundle, "lines.txt")))
  }, error = function(e) {
    # A loaded machine can leave the guardian too little time; report it.
    message("The bounded run did not complete here: ", conditionMessage(e))
  }, finally = {
    for (name in threads) {
      if (is.na(saved[[name]])) Sys.unsetenv(name)
      else do.call(Sys.setenv, as.list(saved[name]))
    }
    unlink(root, recursive = TRUE)
  })
}
```

---

reflow_artifact_ref *Reference a declared artifact in a dependency plan*

Description

References identify logical inventory names, not arbitrary output paths.

Usage

```
reflow_artifact_ref(stage, artifact, read = FALSE)
```

Arguments

stage	Producer stage identifier.
artifact	One logical artifact name in the producer inventory.
read	Read the verified artifact as RDS; otherwise pass its path.

Value

A deferred artifact reference.

Examples

```
# The "values" artifact of stage "data", read back as an R object
ref <- reflow_artifact_ref("data", "values", read = TRUE)
str(ref)
```

---

reflow_artifact_run *Execute or verify one declared artifact bundle*

Description

Fresh runs require an absent destination with an existing parent. Resume revalidates a READY bundle without invoking its writer. Failed/interrupted attempts require explicit caller reconciliation; no subprocess absence is inferred from an R error or owner PID. Stale locks require manual inspection. Orphan attempts with missing state always require a new run after external reconciliation; supplying retry arguments does not bypass this refusal. Receipts use atomic rename visibility, not a power-loss durability guarantee. Writers are trusted: observable input changes are detected, but transient restored changes and escaped subprocesses are not prevented.

Usage

```
reflow_artifact_run(spec, directory)

reflow_artifact_resume(
  spec,
  directory,
  reconciled_attempt = NULL,
  reconciliation = NULL
)
```

Arguments

spec	A reflow_artifact_spec().
directory	New run directory, or an existing directory for resume.
reconciled_attempt	Exact latest failed/interrupted attempt identifier.
reconciliation	Nonempty caller assertion explaining that all writer and subprocess activity has stopped. Required together with the identifier before a fresh retry. This is not automatic process verification.

Value

An authenticated descriptor with stable bundle path and file manifest.

Examples

```
spec <- reflow_artifact_spec(
  "base", "writeLines",
  args = list(text = c("alpha", "beta")),
  outputs = list(con = list(path = "lines.txt", type = "file")),
  inventory = c(lines = "lines.txt")
)
run <- tempfile("artifact-", tmpdir = normalizePath(tempdir(), winslash = "/", mustWork = TRUE))
bundle <- reflow_artifact_run(spec, run)
readLines(file.path(bundle$bundle, "lines.txt"))
bundle$inventory

# Resume re-verifies the accepted bundle without calling the writer
identical(reflow_artifact_resume(spec, run), bundle)
unlink(run, recursive = TRUE)
```

---

reflow_artifact_run_bounded

Run an artifact with an optional Linux guardian

Description

Additive optional Linux execution preserving the synchronous APIs.

Usage

```
reflow_artifact_run_bounded(spec, directory, supervisor_directory,
  resources, runtime_files, cancel = function() FALSE)
reflow_artifact_resume_bounded(spec, directory, supervisor_directory,
  resources, runtime_files, cancel = function() FALSE,
  reconciled_attempt = NULL, reconciliation = NULL)
```

Arguments

`spec`	A declared artifact specification.
`directory`	Artifact run directory.
`supervisor_directory`	Fresh separate supervisor directory.
`resources`	Exact list: timeout_seconds (finite, 1–86400), address_space_bytes, nice (absolute 0–19), threads (1–2), temp_directory, cache_directory, rscript, prlimit, nice_command. Paths are canonical and link-free.
`runtime_files`	Named SHA-256 vector including installed backend scripts, every file of the backend and declared writer Depends/Imports closure and the three executables. Other runtime files must be explicitly declared; no complete ELF closure is inferred.
`cancel`	Callback returning one nonmissing logical value. Errors cancel.
`reconciled_attempt`, `reconciliation`	Explicit retry assertions as in `reflow_artifact_resume`; no automatic process reconciliation.

Details

Requires processx 3.9.0 or later, ps 1.9.3 or later, Linux /proc and declared nice/prlimit executables. Address-space bounds are per process, thread variables are requests, and this is not a sandbox. Trusted same-UID descendants must retain ownership tokens. Guardian death remains unresolved and requires external cleanup. Unknown cleanup retains locks; no stale lock is stolen. A successful worker is revalidated through a locked READY-only path which cannot invoke a writer. The guardian must report readiness within `timeout_seconds`, but no less than 5 and no more than 60 seconds; a guardian that misses this has started no worker and is stopped. Deadlines exclude up to two additional five-second cleanup intervals. No graph or targets scheduling is performed. Parent OPENBLAS_NUM_THREADS, OMP_NUM_THREADS and MKL_NUM_THREADS must already equal the requested thread count. Child locale, RNG kind and full numerical environment must match the captured definition before dispatch. Parent in-process changes are not silently replayed; use an explicit artifact seed for stochastic writers. RNG state is not copied.

Value

A verified artifact descriptor after successful guardian cleanup.

Examples

```
# Needs Linux with processx, ps, prlimit and nice, and a package library
# without symbolic links. R CMD check links the packages into a temporary
# library when it limits them to the declared dependencies
# (_R_CHECK_SUGGESTS_ONLY_, which --as-cran turns on, or
# _R_CHECK_DEPENDS_ONLY_). A run starts a guardian and a worker process and
# pins every installed file they load, which takes a while.

roots <- c("reflowR", "processx", "ps", "digest", "base")
usable <- identical(Sys.info()[["sysname"]], "Linux") &&
  requireNamespace("processx", quietly = TRUE) &&
```

```
  requireNamespace("ps", quietly = TRUE) && ps::ps_is_supported() &&
  nzchar(Sys.which("prlimit")) && nzchar(Sys.which("nice"))
if (usable) {
  # The worker runs at priority 10 with 8 GiB of address space, which an
  # unprivileged session can only request when its own settings allow it.
  space <- grep("^Max address space", readLines("/proc/self/limits"), value = TRUE)
  space <- strsplit(trimws(sub("^Max address space", "", space)), " +")[[1]][1:2]
  usable <- ps::ps_get_nice(ps::ps_handle()) <= 10 && isTRUE(all(
    space == "unlimited" | suppressWarnings(as.numeric(space)) >= 8 * 1024^3
  ))
}
if (usable) {
  # reflowR, processx, ps, digest, the writer package (base) and their imports
  packages <- unique(c(roots, unlist(tools::package_dependencies(
    roots, db = utils::installed.packages(), which = c("Depends", "Imports"),
    recursive = TRUE
  ))))
  usable <- !any(nzchar(Sys.readlink(find.package(packages))))
}
if (usable) {
 root <- tempfile("bounded-", tmpdir = normalizePath(tempdir(), winslash = "/", mustWork = TRUE))
  work <- file.path(root, c("tmp", "cache"))
  for (path in c(root, work)) dir.create(path)
  spec <- reflow_artifact_spec(
    "base", "writeLines",
    args = list(text = c("alpha", "beta")),
    outputs = list(con = list(path = "lines.txt", type = "file")),
    inventory = c(lines = "lines.txt")
  )
  resources <- list(
    timeout_seconds = 60, address_space_bytes = 8 * 1024^3, nice = 10L,
    threads = 1L, temp_directory = work[1], cache_directory = work[2],
    rscript = normalizePath(file.path(R.home("bin"), "Rscript")),
    prlimit = normalizePath(Sys.which("prlimit")),
    nice_command = normalizePath(Sys.which("nice"))
  )
  # Pin the backend scripts, every installed file of those packages and the
  # three executables.
  files <- c(
    list.files(system.file("bounded", package = "reflowR"), full.names = TRUE),
    unlist(lapply(find.package(packages), list.files, recursive = TRUE,
                  full.names = TRUE, all.files = TRUE)),
    resources$rscript, resources$prlimit, resources$nice_command
  )
  files <- unique(normalizePath(files, winslash = "/"))
  pins <- vapply(files, digest::digest, character(1), algo = "sha256", file = TRUE)

  # The parent thread settings must equal the requested thread count; they
  # are restored afterwards, also when the run fails.
  threads <- c("OPENBLAS_NUM_THREADS", "OMP_NUM_THREADS", "MKL_NUM_THREADS")
  saved <- Sys.getenv(threads, unset = NA)
  Sys.setenv(OPENBLAS_NUM_THREADS = 1, OMP_NUM_THREADS = 1, MKL_NUM_THREADS = 1)
  tryCatch({
```

```
    bundle <- reflow_artifact_run_bounded(spec, file.path(root, "run"),
                                          file.path(root, "supervisor"),
                                          resources, pins)
    print(readLines(file.path(bundle$bundle, "lines.txt")))
  }, error = function(e) {
    # A loaded machine can leave the guardian too little time; report it.
    message("The bounded run did not complete here: ", conditionMessage(e))
  }, finally = {
    for (name in threads) {
      if (is.na(saved[[name]])) Sys.unsetenv(name)
      else do.call(Sys.setenv, as.list(saved[name]))
    }
    unlink(root, recursive = TRUE)
  })
}
```

`reflow_artifact_select` *Select an immutable artifact graph generation*

Description

Serial trusted graph selection using existing graph definitions and historical verification. Unchanged nodes remain in prior generations without payload copies. Linux local filesystem qualification is required; cached references alone are unverified.

Usage

```
reflow_artifact_select(plan, registry, initialize = FALSE, reconciliation = NULL)
```

Arguments

`plan`	A `reflow_artifact_plan`.
`registry`	Canonical registry directory with an existing parent.
`initialize`	Explicitly create an absent registry.
`reconciliation`	NULL or exact list with pending_hash, generation, action, quiescent, reason and reconciliations. Actions: publish_complete, resume_incomplete or close_failed. Quiescent must be TRUE and is an operator assertion, not a detected fact.

Details

Registry locking precedes inspection. Every completed generation remains an integrity obligation, including failed unpublished siblings; verification releases all historical locks before graph operations. A pending transaction blocks ordinary selection. Explicit reconciliation never silently retries an absent or malformed graph. Event staging residues require external forensic recovery. Current-pointer replacement prevents closing that generation failed. Same-directory rename provides visibility, not power-loss durability. This API does not supervise resource use or complete workflow orchestration.

Value

Compact reference with status REFERENCE_REQUIRES_VERIFICATION, or NULL for close_failed.

Examples

```
# The selector is qualified for local Linux file systems only
if (identical(Sys.info()[["sysname"]], "Linux")) {
  plan <- reflow_artifact_plan(reflow_artifact_stage(
    "lines", "base", "writeLines",
    args = list(text = c("alpha", "beta")),
    outputs = list(con = list(path = "lines.txt", type = "file")),
    inventory = c(lines = "lines.txt")
  ))
  root <- tempfile("selection-", tmpdir = normalizePath(tempdir(), winslash = "/", mustWork = TRUE))
  dir.create(root)
  registry <- file.path(root, "registry")
  pointer <- reflow_artifact_select(plan, registry, initialize = TRUE)
  pointer$status

  # An unchanged plan selects the same generation again
  identical(reflow_artifact_select(plan, registry), pointer)
  unlink(root, recursive = TRUE)
}
```

---

reflow_artifact_selection_verify

Verify a cached generation reference

Description

Refuses incomplete or stale selections before using the existing graph resume verification. Verification acquires and releases locks but does not dispatch writers or mutate completed payloads or journals.

Usage

```
reflow_artifact_selection_verify(pointer, plan)
```

Arguments

pointer	A compact artifact selection reference.
plan	The current artifact plan.

Value

The verified graph descriptor for immediate consumption.

Examples

```
# The selector is qualified for local Linux file systems only
if (identical(Sys.info()[["sysname"]], "Linux")) {
  plan <- reflow_artifact_plan(reflow_artifact_stage(
    "lines", "base", "writeLines",
    args = list(text = c("alpha", "beta")),
    outputs = list(con = list(path = "lines.txt", type = "file")),
    inventory = c(lines = "lines.txt")
  ))
 root <- tempfile("selection-", tmpdir = normalizePath(tempdir(), winslash = "/", mustWork = TRUE))
  dir.create(root)
  pointer <- reflow_artifact_select(plan, file.path(root, "registry"),
                                    initialize = TRUE)
  # Verify immediately before reading the selected files
  graph <- reflow_artifact_selection_verify(pointer, plan)
  readLines(file.path(graph$nodes$lines$descriptor$bundle, "lines.txt"))
  unlink(root, recursive = TRUE)
}
```

`reflow_artifact_spec` *Declare one immutable file-producing bundle*

Description

Calls one trusted installed export, substituting only declared top-level output arguments. The writer's return value is discarded. This is not a sandbox, scheduler, native-process supervisor or complete artifact DAG. Plain strings are configuration; files must use `reflow_imaging_input()`. Paths reject symbolic links, special files, reserved Windows components and trailing dots/spaces. Unicode spelling is preserved, not normalized. Parents of output paths are created; declared output directories are writer-created. Installed package resources (including templates), tracked inputs and declared external runtime files are fingerprinted. Undeclared ambient state is not. With `options(reflowR.package_hash_cache = TRUE)` the hashes of installed package files are reused within one R process while a file keeps its device, inode, size, modification and change times; use it only with a library that nothing modifies during the run. `options(reflowR.input_hash_cache = TRUE)` does the same for tracked input files; otherwise inputs are re-read for every fingerprint.

Usage

```
reflow_artifact_spec(
  package,
  fun,
  args = list(),
  outputs,
  inventory,
  allow_empty = character(),
  seed = NULL,
  packages = character(),
  runtime_files = character()
)
```

Arguments

package, fun	Installed package and exported writer function.
args	Uniquely named ordinary arguments, excluding output arguments.
outputs	Named list keyed by output argument, each an exact list with path (relative to the bundle) and type (file or directory).
inventory	Nonempty named character vector of logical artifact names mapped to exact relative file paths. Unexpected directories also fail.
allow_empty	Logical artifact names permitted to contain zero bytes.
seed	Explicit nonnegative integer seed, or NULL for a nonrandom writer.
packages	Additional installed packages to fingerprint.
runtime_files	Existing external runtime files to fingerprint. The caller must ensure the writer actually dispatches these binaries/resources.

Value

A versioned declaration; no writer is invoked.

Examples

```
spec <- reflow_artifact_spec(
  "utils", "write.csv",
  args = list(x = data.frame(id = 1:3, value = c(2.5, 3.1, 4.7)),
              row.names = FALSE),
  outputs = list(file = list(path = "table.csv", type = "file")),
  inventory = c(table = "table.csv")
)
spec$outputs
spec$inventory
```

---

reflow_artifact_stage *Declare a deferred artifact writer*

Description

This additive API permits reflow_artifact_ref() in ordinary arguments. Static declarations are validated immediately; future artifact paths are resolved only after their producers have been authenticated. Scientific methods remain in trusted installed exports. No writer runs here.

Usage

```
reflow_artifact_stage(
  id,
  package,
  fun,
  args = list(),
```

```
  outputs,
  inventory,
  allow_empty = character(),
  seed = NULL,
  packages = character(),
  runtime_files = character()
)
```

Arguments

id	Stage identifier, unique within a plan even when letter case is ignored.
package, fun, args, outputs, inventory, allow_empty, seed, packages, runtime_files	See reflow_artifact_spec().

Value

A deferred writer declaration.

Examples

```
stage <- reflow_artifact_stage(
  "data", "base", "saveRDS",
  args = list(object = c(alpha = 1.5, beta = 2.5)),
  outputs = list(file = list(path = "values.rds", type = "file")),
  inventory = c(values = "values.rds")
)
stage$id
stage$declaration$inventory
```

---

reflow_artifact_target

Always verify a compact artifact selection in targets

Description

Creates an optional targets RDS target with an always cue. Every make calls the selector; the stored value is only a compact reference. No graph payloads are copied. A targets cache read alone does not verify current files: consumers must call reflow_artifact_selection_verify immediately before use. Plan expressions must not embed raw payloads.

Usage

```
reflow_artifact_target(name, plan_command, registry)
```

Arguments

name	A plain target name.
plan_command	One unevaluated language object constructing the plan.
registry	Canonical selector registry path, initialized before execution.

Value

A targets target object.

Examples

```
if (requireNamespace("targets", quietly = TRUE)) {
 registry <- file.path(normalizePath(tempdir(), winslash = "/", mustWork = TRUE), "registry")
  # The plan expression is stored unevaluated and runs at every make
  target <- reflow_artifact_target("selection", quote(my_plan()), registry)
  target$command$expr
}
```

---

reflow_call_stage *Declare a stage that calls one installed export*

Description

Builds the reflow_artifact_stage() for reflow_call_writer(): the stage saves the value of fun as the artifact value (value.rds) and a tab-separated copy as table (value.tsv, empty for values that are not plain data frames). The package of fun and any packages are fingerprinted. Refer to the value downstream with reflow_artifact_ref(id, "value", read = TRUE).

Usage

```
reflow_call_stage(id, fun, args = list(), seed = NULL, packages = character())
```

Arguments

id	Stage identifier, unique within a plan even when letter case is ignored.
fun	One string "package::function" naming an exported function.
args	Uniquely named list of arguments; may contain reflow_artifact_ref() and reflow_imaging_input() values at any depth.
seed	Explicit nonnegative integer seed, or NULL for a nonrandom call.
packages	Additional installed packages to fingerprint, such as optional packages that fun uses.

Value

A deferred writer declaration.

Examples

```
stage <- reflow_call_stage("top", "utils::head", list(x = 1:10, n = 3))
stage$declaration$inventory
```

`reflow_call_writer` *Call one installed export and save its value*

Description

A generic artifact writer: calls exactly one exported function of an installed package with the given arguments and saves the returned object as RDS. Use it as the writer of a `reflow_artifact_stage()` (see `reflow_call_stage()`) so that configured calculations need no writer code of their own. Artifact references in `args` are resolved by the executor before this function runs; with `read = TRUE` they arrive as objects.

Usage

```
reflow_call_writer(fun, args = list(), path, tsv = NULL)
```

Arguments

`fun`	One string `"package::function"` naming an exported function.
`args`	Uniquely named list of arguments for `fun`.
`path`	Output path of the RDS file.
`tsv`	Optional output path of the tab-separated copy, or NULL.

Details

The function is trusted and is not sandboxed. It must not write files of its own. Declare its package in the stage `packages` so that its installed files are fingerprinted; `reflow_call_stage()` does this.

A tab-separated copy is written when `tsv` is a path. A data frame whose columns are all atomic vectors is written with a header row, `NA` for missing values and double quotes only around fields that contain a tab, line break or double quote. Any other value gives an empty file, so the declared inventory stays fixed; declare that artifact in `allow_empty`. The RDS file is the exact value; the table is a readable copy.

Value

The output paths, invisibly.

Examples

```
out <- tempfile(fileext = ".rds")
reflow_call_writer("utils::head", list(x = datasets::mtcars, n = 3), out)
readRDS(out)
```

`reflow_figure_plot` *Build a Plot from a Finalized-Value Figure Recipe*

Description

Build a Plot from a Finalized-Value Figure Recipe

Usage

```
reflow_figure_plot(data, recipe, annotation_data = NULL)
```

Arguments

`data`	Data frame with plain finite numeric plotted values and nonblank character keys/categories. Complete row keys must be unique. Extra columns are ignored without modifying caller input. Repeated display positions remain distinct keyed rows. Numeric points require nonempty data and preserve input row order, with complete colour/facet levels and fixed numeric scales.
`recipe`	A recipe from `reflow_figure_recipe()`.
`annotation_data`	NULL for existing recipes. Wrapped numeric points require a separate frame with exactly one uniquely keyed row per declared facet. Selected key, facet and label columns must be plain nonmissing character vectors without attributes. Keys and facet values must each be unique; extra columns are ignored. Row order, literal newlines and supplied status/NA text are preserved; no statistics are computed.

Details

Numeric points draw the supplied coordinates without aggregation, filtering, fitting, jitter or automatic offsets. Text nudges affect annotations only. Device-specific visible-label coverage, clipping and minimum exported font sizes require separate render and visual qualification. No files are written.

Value

Versioned list with plot (ggplot, or NULL for explicitly empty legacy input), status, rows, selected_columns, payload_sha256 and recipe_sha256. Numeric points additionally return annotation_records (zero when absent) and visible_annotations = `NA_integer_`: retained records do not establish device-visible labels. Clipping and check_overlap can suppress visibility. The payload hash binds key-sorted selected values and mapping, row keys, and supplied midpoint/reference as applicable. Styling, level orders, numeric axis declarations, colour maps and unverified provenance bind the recipe hash. Extra unselected status columns are not authenticated by these hashes. Wrapped numeric points additionally return annotation_payload_sha256 for key-sorted selected annotation values and declared key/facet/label columns. Canonical payload hashes do not record input row order; drawing preserves it and order equality requires separate diagnostics. Hashes use R version-2 serialization, not a cross-language protocol.

Examples

```
scores <- data.frame(
  id = c("r1", "r2", "r3", "r4"),
  sample = c("s1", "s2", "s1", "s2"),
  marker = c("CD3", "CD3", "CD8", "CD8"),
  score = c(-1.2, 0.4, 0.8, 1.5)
)
recipe <- reflow_figure_recipe(
  "tile_heatmap",
  mapping = c(x = "sample", y = "marker", value = "score"),
  row_key = "id",
  levels = list(x = c("s1", "s2"), y = c("CD3", "CD8")),
  midpoint = 0,
  labels = list(title = "Finalized scores", fill = "z-score")
)
figure <- reflow_figure_plot(scores, recipe)
figure$rows
figure$payload_sha256
figure$plot
```

`reflow_figure_recipe` *Declare a Finalized-Value Figure Recipe*

Description

Supports tile heatmaps, effect points and supplied numeric points. Values, categorical orders and scales are supplied by the caller. No observations are filtered, aggregated, fitted, jittered or interpreted as independent units.

Usage

```
reflow_figure_recipe(
  type, mapping, row_key, levels,
  facets = list(), labels = list(), midpoint = NULL, reference = NULL,
  style = list(), status = "available", provenance = character(),
  display_labels = list(), numeric_axes = NULL, colour_values = NULL,
  annotation = NULL, scatter = NULL,
  tile = NULL
)
```

Arguments

`type`	`tile_heatmap`, `effect_points`, `numeric_points` or `numeric_tiles`.
`mapping`	Named character vector of data column names. Heatmaps require `x`, `y`, `value`; effects and numeric points require `x`, `y`, `colour`. Numeric tiles require `x`, `y`, `fill`.
`row_key`	Nonempty character vector of complete unique-key columns.

`levels` Named list of explicit unique character levels: x/y for heatmaps, y/colour for effects, colour for numeric points, plus facet_rows/facet_columns for requested facets. Nonempty data require exact coverage of observed levels, except numeric tiles: levels$fill may contain unused palette categories.

`facets` Named list with optional rows/columns column names. Heatmap and effect facets use free y scales and space. Numeric-point facets use fixed scales and space in both directions. Facets do not aggregate values.

`labels` Named list of optional title/x/y/fill/colour/caption text; numeric points also accept subtitle. Explicit NULL removes a label; empty text remains a distinct label.

`midpoint` Required finite scalar for available heatmaps; empty heatmaps may supply NULL. Must be NULL for effects and numeric points.

`reference` Finite vertical reference for effects; NULL for heatmaps. Numeric points accept NULL or a finite horizontal reference within y limits.

`style` Named list of presentation overrides: base_size, family, low, mid, high, point_size, tile_linewidth, x_angle, strip_y_angle (default -90). Optional positive finite text_size overrides only root text, preserving geom defaults and explicit child text sizes. Omission preserves legacy recipes. No expressions allowed.

`status` Explicit available or empty for heatmaps/effects, agreeing with supplied rows. Numeric points require available and nonempty data.

`provenance` Named character vector of caller declarations, not verified evidence. It is excluded from the scientific payload hash.

`display_labels` Optional complete named plaintext maps for categorical x/y axes or facet_rows/facet_columns. Numeric points permit only facet maps; numeric axis labels come from numeric_axes. Map keys are original levels. Newlines and repeated display text are allowed without merging categories. Omitted roles use identity labels; colour labels are not mapped here.

`numeric_axes` Numeric points only: named x/y lists, each containing limits (two strictly increasing finite values), breaks (unique increasing finite values within limits), and an equally long character labels vector. Both axes are required. Coordinates outside the declared limits are refused.

`colour_values` Numeric points or tiles: complete named colour vector whose unique names exactly match `levels$colour` (points) or `levels$fill` (tiles). Colour order follows those levels.

`annotation` Numeric points only: NULL or a named list with column, nudge_x, nudge_y, size_pt, check_overlap and show_legend. Nudges must be finite; size_pt must be finite and at least 7; both policies must be logical scalars. Text size uses ggplot2's 72.27 points per inch conversion. Annotation records are retained; clipping and overlap suppression can hide text. Actual visible label coverage and exported glyph sizes require device and visual checks.

`scatter` Numeric points only: NULL preserves the existing fixed-scale recipe. A plain list of ncol (integer-valued 1..100), point_alpha (0..1), and corner selects free-y facet_wrap with one facets$rows column and declared facet order. Corner

contains row_key, column, finite hjust/vjust, size_pt at least7, and positive lineheight. Separate annotation_data supplies exactly one uniquely keyed plaintext row per facet. Display anchors are negative/positive infinity; point coordinates remain finite. Reference and row-linked annotation must be NULL. Both numeric_axes limits remain strict finite input envelopes. Paired NULL breaks/labels request automatic ticks in this mode only. The x viewing range uses its limits; y trains per panel with ordinary expansion, not a common y viewing limit. No values or statistics are changed.

`tile` Numeric tiles only: a list with finite positive width and height, and legend equal to "observed" or "declared". Optional legend_text_size_pt is NULL or a positive finite scalar overriding only legend text. Omitted or NULL values preserve the default tile recipe. Optional axis_text_x_inherit_blank is NULL or a plain logical scalar. TRUE preserves blank-element inheritance for numeric x-axis text; omitted, NULL or FALSE retain the previous plotting default. Omitted and NULL values preserve recipe bytes. Other types are unaffected. Requires mapping x/y/fill, levels$fill and a complete named colour_values palette. Coordinates stay numeric, borders are absent and aspect ratio is one. Axes train on tile extents with ordinary expansion. Observed legends include present categories in declared order; declared legends include unused levels without adding tiles. Singleton input is supported, empty input refused. Facets, numeric_axes, annotation, scatter, reference, midpoint and display_labels are unsupported. Status is available. Labels accept title/subtitle/x/y/fill/caption. No categories are inferred.

Details

Numeric-only options must remain NULL for existing types. Their default omission preserves existing recipe structure and behaviour. Numeric coordinates are supplied directly: no aggregation, filtering, jitter, automatic offsets or implicit reordering is performed. Plotted rows retain input order; categorical display order follows the declared levels.

Value

A versioned recipe; no data are processed or files written.

Examples

```
recipe <- reflow_figure_recipe(
  "tile_heatmap",
  mapping = c(x = "sample", y = "marker", value = "score"),
  row_key = "id",
  levels = list(x = c("s1", "s2"), y = c("CD3", "CD8")),
  midpoint = 0,
  labels = list(title = "Finalized scores", fill = "z-score")
)
recipe$type
recipe$mapping
```

reflow_icon_base64 *Get Base64-Encoded SVG Icon for a Scheme*

Description

Reads the CTTIR v5 family icon from the package's `inst/icons/` directory and returns it as a base64-encoded data URI suitable for embedding in HTML. The icon is the v5 hex on a light surface: a tinted face, the family accent as rim and a small family glyph, with a transparent exterior.

Usage

```
reflow_icon_base64(scheme_name)
```

Arguments

`scheme_name`	Character. A v5 family key (e.g., `"clinical"`) or one of the earlier scheme names, which resolve to their v5 family icon.

Value

A character string containing the base64 data URI (e.g., `"..."`).

Examples

```
uri <- reflow_icon_base64("clinical")
nchar(uri)
```

reflow_imaging_plan *Assemble a sequential imaging dependency graph*

Description

Stages must be in dependency order. Forward references and cycles fail. File inputs must use `reflow_imaging_input()` so content changes invalidate a resume. Plain character arguments are configuration, not tracked file inputs.

Usage

```
reflow_imaging_plan(..., packages = character())
```

Arguments

`...`	Stage specifications.
`packages`	Additional installed packages to fingerprint, for example optional backends used by a stage. Declare these explicitly.

Value

A versioned plan; no stage is executed.

Examples

```
plan <- reflow_imaging_plan(
  reflow_imaging_stage("values", "base", "identity",
                       list(x = c(2, 4, 4, 5))),
  reflow_imaging_stage("mean", "base", "mean",
                       list(x = reflow_imaging_ref("values")))
)
vapply(plan$stages, `[[`, character(1), "id")
```

---

reflow_imaging_ref *Reference an upstream result or an input file*

Description

Reference an upstream result or an input file

Usage

```
reflow_imaging_ref(stage, select = character())

reflow_imaging_input(path, read = FALSE)
```

Arguments

stage	Identifier of an earlier stage.
select	Character vector of nested list components to extract.
path	Existing input file or directory. All contents are hashed.
read	Read an RDS input as an object; otherwise pass its absolute path.

Value

An explicit dependency reference.

Examples

```
# Pass the value of an earlier stage, or one component of it
reflow_imaging_ref("fit")
reflow_imaging_ref("fit", select = "coefficients")

# Track an input file: its contents are hashed into the run definition
path <- tempfile(fileext = ".rds")
saveRDS(c(2, 4, 4, 5), path)
input <- reflow_imaging_input(path, read = TRUE)
input$read
unlink(path)
```

`reflow_imaging_run` *Execute or resume a recorded imaging plan*

Description

Runs are sequential and local. Every call result is saved to a new immutable attempt directory. Receipts contain configuration, input, package, runtime, resolved-argument and output hashes. Resume rejects changed definitions or inputs and corrupt results, rather than silently reusing stale output. A failed call can be retried with the identical plan; completed stages are verified before reuse. Random stages require an explicit stage seed; the caller's random state is restored after each call.

Usage

```
reflow_imaging_run(plan, directory)

reflow_imaging_resume(plan, directory)
```

Arguments

`plan`	A `reflow_imaging_plan()`.
`directory`	New run directory, or existing run directory for resume.

Details

While a stage function is being called, an external calling handler can use `invokeRestart("reflow_abort_stage", condition)` to reject a condition. The restart raises a `simpleError` with that condition's message and call inside the executor's failure handlers, so the stage and run record `FAILED` before the error reaches the caller. No downstream stage runs. This also supports warnings delivered directly with `base::signalCondition()`. The restart is available only during the stage call, not argument resolution or result persistence. Warning classification and muffling remain the caller's policy; warning behaviour is unchanged when the restart is not invoked.

The executor never initializes Git, publishes, installs packages or downloads data. Called functions are trusted and can have side effects; this API does not sandbox them. File-producing, distributed and resource-scheduled stages are outside this initial object-returning contract. Package fingerprints cover installed code, native libraries and dependency metadata, not external engines. Common OpenMP/BLAS/MKL/Accelerate/BLIS/RcppParallel thread environment settings are recorded, including unset values, and changes invalidate resume. These are declared environment settings, not measured effective thread counts; backend-specific in-process thread setters still require explicit configuration.

A directory lock prevents concurrent execution. After an interrupted process, inspect the lock's owner receipt and verify that process has stopped before manually removing `.lock`; stale locks are never automatically stolen.

Value

Invisibly, a named list of stage result objects.

Examples

```
plan <- reflow_imaging_plan(
  reflow_imaging_stage("values", "base", "identity",
                       list(x = c(2, 4, 4, 5))),
  reflow_imaging_stage("mean", "base", "mean",
                       list(x = reflow_imaging_ref("values")))
)
run <- tempfile("imaging-")
results <- reflow_imaging_run(plan, run)
results$mean

# Resume verifies the recorded results instead of calling the stages again
identical(reflow_imaging_resume(plan, run), results)
unlink(run, recursive = TRUE)
```

---

reflow_imaging_stage *Declare an imaging workflow stage*

Description

Stages call installed exported functions; they do not evaluate script text. Scientific methods and validation remain in the called packages. Calls must return serializable values and must not mutate their inputs or write external outputs. This executor is not a sandbox: use trusted package functions only.

Usage

```
reflow_imaging_stage(id, package, fun, args = list(), seed = NULL)
```

Arguments

id	Stage identifier containing letters, digits, underscores; unique within a plan even when letter case is ignored.
package	Installed package name.
fun	Exported function name.
args	Named list of arguments, including explicit references if needed.
seed	Optional integer seed for stochastic calls. Calls consuming the ambient random stream without an explicit stage seed fail.

Value

A stage specification.

Examples

```
stage <- reflow_imaging_stage("summary", "base", "summary",
                              args = list(object = c(2, 4, 4, 5)))
stage$id
# Stochastic calls need an explicit seed
noisy <- reflow_imaging_stage("noise", "stats", "rnorm", list(n = 3), seed = 1L)
noisy$seed
```

`reflow_init` *Initialize a Themed workflowr Project*

Description

Wraps `wflow_start` to create a standard workflowr project, then extends it with a themed colour scheme, structured Rmd analysis pipeline, utility scripts, and a Quarto report template.

Usage

```
reflow_init(
  directory,
  name = NULL,
  author = "Author Name",
  email = NULL,
  affiliation = NULL,
  orcid = NULL,
  website = NULL,
  scheme = c("brand", "spectral", "spatial", "omics", "clinical", "methods",
    "simulation", "evidence", "platform", "basic", "code", "special", "other"),
  depth = c("standard", "minimal", "extended"),
  seed = as.integer(format(Sys.Date(), "%Y%m%d")),
  github_url = NULL,
  existing = FALSE,
  overwrite = FALSE,
  git = TRUE,
  change_wd = rlang::is_interactive(),
  user.name = NULL,
  user.email = NULL,
  open = rlang::is_interactive()
)
```

Arguments

`directory`	Character. Path where the project will be created. Passed directly to `workflowr::wflow_start()`.
`name`	Character. Project name (used in titles and navbar). If `NULL`, derived from the directory basename.
`author`	Character. Full name of the primary author.

email	Character. Author's email address (optional).
affiliation	Character. Author's institutional affiliation (optional).
orcid	Character. Author's ORCID iD (optional, format: "0000-0000-0000-0000").
website	Character. URL to author's or project website (optional).
scheme	Character. CTTIR v5 family whose colours, icon and fonts style the site: one of "brand" (default), "spectral", "spatial", "omics", "clinical", "methods", "simulation", "evidence" or "platform". The earlier names "basic", "code", "special" and "other" remain accepted as aliases of brand, methods, evidence and platform. See reflow_schemes.
depth	Character. Rmd pipeline depth. One of "minimal", "standard", "extended". Defaults to "standard".
seed	Integer. Random seed for workflowr reproducibility. Defaults to as.integer(format(Sys.Date(), "%Y%m%d")).
github_url	Character. GitHub repository URL (optional). Adds a GitHub link to the navbar.
existing	Logical. Is the directory an existing project or folder? Passed to workflowr::wflow_start(existing = ...), which adds the missing workflowr files and leaves present ones alone. Default FALSE. Files that reflowR writes are replaced only with overwrite = TRUE. Files already in the folder are not committed (see git).
overwrite	Logical. Replace files that reflowR writes if they already exist in directory: README.md, _workflowr.yml, analysis/_site.yml, analysis/index.Rmd, analysis/about.Rmd and the theme, icon, script, pipeline and report files. Default FALSE: the function then stops before writing anything and lists the files. Other workflowr files are never replaced. Re-theming at a different depth leaves the pipeline files of the earlier depth that the new one does not write in analysis/; remove them yourself if they are no longer needed.
git	Logical. Initialize a git repository (or use the one in an existing folder) and commit the files that this call writes? Other files in directory, and changes to them, are not committed. If the repository already has staged changes, no commit is made and a warning says so. Default TRUE.
change_wd	Logical. Change working directory to the new project? Default TRUE in interactive sessions.
user.name	Character. Git user name (optional). Passed to workflowr::wflow_start().
user.email	Character. Git user email (optional). Passed to workflowr::wflow_start().
open	Logical. Open the project in RStudio after creation? Default TRUE in interactive sessions.

Details

The function proceeds in two phases:

Phase 1 – workflowr base: Calls workflowr::wflow_start() with git = FALSE to create the standard workflowr scaffold without committing yet.

Phase 2 – reflowR theming: Replaces _site.yml, index.Rmd, about.Rmd, _workflowr.yml and README.md with themed versions; adds the analysis pipeline Rmd files, CSS/SCSS, icon, setup

script, utility script, Quarto report template, and additional subdirectories. If `git = TRUE`, initializes git and commits the files written by both phases.

If any of the files that Phase 2 writes already exists in `directory` before the call, for example the README or `_site.yml` of an existing workflowr project or the files of an earlier call, the function stops before writing anything and lists them, so a project is never changed silently or left partly updated. Set `overwrite = TRUE` to replace them with the reflowR versions. It also stops before writing anything if one of these files, or the folder it goes in, is read-only.

Value

Invisibly returns the path to the created project directory.

See Also

`wflow_start` for the underlying project creation, `wflow_build` to render Rmd files, `wflow_publish` to commit + build + commit, `reflow_schemes` and `reflow_presets` for available options.

Examples

```
# A clinical project with the three-step pipeline, in a temporary folder
project <- file.path(tempdir(), "my_project")
reflow_init(
  directory = project,
  author    = "Your Name",
  email     = "you@example.com",
  scheme    = "clinical",
  depth     = "minimal",
  git       = FALSE,
  change_wd = FALSE,
  open      = FALSE
)
list.files(file.path(project, "analysis"))

# In a real project, continue with the standard workflowr commands, e.g.
# workflowr::wflow_build() and workflowr::wflow_publish().
unlink(project, recursive = TRUE)
```

---

reflow_init_strict *Write a Strict Workflow Project*

Description

Writes a new project from canonical templates, without installing dependencies, committing, rendering or publishing. Existing paths and symbolic links are rejected. Raw data files and their directory are made read-only.

Usage

```
reflow_init_strict(
  directory,
  profile,
  name,
  author,
  dependency_lock,
  git = FALSE,
  composition = NULL
)
```

Arguments

directory	New project directory.
profile	A declaration from reflow_profile().
name	Plain-text project name.
author	Plain-text project author.
dependency_lock	Parsed R renv lockfile containing R and Packages fields.
git	Whether to initialize an empty Git repository. Default FALSE. No commit is made and Git identity configuration is not changed.
composition	For strict-cttir-v1 only, a plain list with exactly schema_version = 1L, study, recipe, and runtime. Study and recipe are bounded JSON objects whose scientific semantics are validated by cttiR. Runtime has package = "cttiR", version, writer = "study_stage", and files, a complete named list of relative installed-package file SHA-256 digests. The package version must match the dependency lock. Default NULL.

Value

Invisibly, the scaffold with its absolute project directory.

Examples

```
pins <- c(digest = "0.6.37", jsonlite = "1.8.9", knitr = "1.49",
          rmarkdown = "2.29", workflowr = "1.7.1")
lock <- list(
  R = list(Version = "4.4.2"),
  Packages = lapply(stats::setNames(names(pins), names(pins)), function(p) {
    list(Package = p, Version = pins[[p]], Source = "Repository")
  })
)
profile <- reflow_profile("clinical", "secondary", "observational")
project <- tempfile("strict-", tmpdir = normalizePath(tempdir(), winslash = "/", mustWork = TRUE))
scaffold <- reflow_init_strict(project, profile, "Example study", "Your Name", lock)
list.files(project)

# Raw data are read-only; make them writable again before deleting
```

```
Sys.chmod(file.path(project, "data", "raw"), "0755")
unlink(project, recursive = TRUE)
```

`reflow_preset` *Get a Depth Preset Definition*

Description

Returns the full preset list for a given preset name.

Usage

```
reflow_preset(name)
```

Arguments

`name`	Character. One of `"minimal"`, `"standard"`, `"extended"`.

Value

A named list containing the preset name, label, file names, and descriptions.

See Also

`reflow_presets()` for listing all available presets.

Examples

```
p <- reflow_preset("standard")
p$files
p$descriptions
```

`reflow_presets` *List Available Depth Presets*

Description

Prints a formatted table of all available reflowR depth presets and returns their names invisibly.

Usage

```
reflow_presets()
```

Value

Invisibly returns a character vector of preset names.

Examples

```
reflow_presets()
```

`reflow_preview` *Preview a Colour Scheme*

Description

Opens an HTML preview in the RStudio viewer (or browser) showing the family icon on a flat navbar with its accent rule, the light and dark surfaces, the identity colours and the Okabe-Ito palette.

Usage

```
reflow_preview(scheme = "brand")
```

Arguments

`scheme`	Character. A v5 family key (e.g., `"clinical"`) or an earlier scheme name, see `reflow_scheme()`.

Value

Invisibly returns the scheme list.

Examples

```
if (interactive()) {
  reflow_preview("clinical")
  reflow_preview("brand")
}
```

`reflow_profile` *Declare a Strict Research Workflow Profile*

Description

Creates a versioned declaration for a research-group project. Domain, data origin and design are independent dimensions. A profile records applicable stages; it does not select statistical methods or approve data access.

Usage

```
reflow_profile(
  domain,
  data_origin,
  design,
  seed = 1L,
  stages = NULL,
  profile_version = "strict-v1",
  data_access = "unknown",
  jurisdiction = "unknown",
  legal_basis = "unknown"
)
```

Arguments

domain	One of clinical, basic, methods, software, or other.
data_origin	One of primary or secondary.
design	One of experimental, observational, synthesis, simulation, software, or other.
seed	One nonnegative integer for reproducible analysis sessions.
stages	Optional named list of stage overrides. Each value has exactly applicable (logical) and reason (plain text) fields. An inapplicable stage requires a nonempty reason. Unspecified stages remain applicable.
profile_version	strict-v1 (default), or the explicit cttiR composition strict-cttir-v1.
data_access	One of unknown, public, controlled, restricted, or synthetic. This records a declaration, not access authorization.
jurisdiction	Plain-text jurisdiction or unknown.
legal_basis	Plain-text access/use basis or unknown.

Value

A declarative list of class reflow_strict_profile.

Examples

```
p <- reflow_profile("clinical", "secondary", "observational")
p$scheme
p$badge
```

`reflow_scaffold` *Plan a Strict Workflow Scaffold*

Description

Returns canonical project files without writing or running them. The supplied dependency lock is parsed data from an existing renv lockfile; no dependencies are installed or restored. Preserve the returned declaration outside the generated project as the trusted input to validation.

Usage

```
reflow_scaffold(profile, name, author, dependency_lock, composition = NULL)
```

Arguments

`profile`	A declaration from `reflow_profile()`.
`name`	Plain-text project name.
`author`	Plain-text project author.
`dependency_lock`	Parsed R renv lockfile containing R and Packages fields.
`composition`	For `strict-cttir-v1` only, a plain list with exactly `schema_version = 1L`, `study`, `recipe`, and `runtime`. Study and recipe are bounded JSON objects whose scientific semantics are validated by cttiR. Runtime has `package = "cttiR"`, `version`, `writer = "study_stage"`, and `files`, a complete named list of relative installed-package file SHA-256 digests. The package version must match the dependency lock. Default `NULL`.

Value

A list containing the validated profile, trusted declaration, named relative file contents, directories, and content fingerprints.

Examples

```
# A parsed renv lockfile; in practice use jsonlite::read_json("renv.lock")
pins <- c(digest = "0.6.37", jsonlite = "1.8.9", knitr = "1.49",
          rmarkdown = "2.29", workflowr = "1.7.1")
lock <- list(
  R = list(Version = "4.4.2"),
  Packages = lapply(stats::setNames(names(pins), names(pins)), function(p) {
    list(Package = p, Version = pins[[p]], Source = "Repository")
  })
)
profile <- reflow_profile("clinical", "secondary", "observational")
scaffold <- reflow_scaffold(profile, "Example study", "Your Name", lock)
head(names(scaffold$files))
```

`reflow_scheme` *Get a Colour Scheme Definition*

Description

Returns the site scheme for one CTTIR v5 topic family. Every colour, surface and font is derived from `inst/cttir/design_tokens.csv`, the token table shared with hexmakR and the themakR documentation template.

Usage

```
reflow_scheme(name)
```

Arguments

`name`	Character. One of the v5 families `"spectral"`, `"spatial"`, `"omics"`, `"clinical"`, `"methods"`, `"simulation"`, `"evidence"`, `"platform"` or `"brand"`. The earlier names remain accepted as aliases: `"basic"` (brand), `"code"` (methods), `"special"` (evidence) and `"other"` (platform). An alias prints a one-time message per session.

Value

A named list. `family`, `label`, `accent` and `navbar_icon` identify the family. Surfaces are given per colour mode as `page_*`, `text_*`, `heading_*`, `link_*`, `code_surface_*`, `border_*` and `muted_*` with suffix `_light` or `_dark`; `face_light` and `face_dark` are the tinted tile faces, `syntax_light` and `syntax_dark` the code token colours, `palette` the Okabe-Ito topic colours starting with the family accent, and `font_body`, `font_code`, `font_brand` and `brand_weight` the typography. The earlier fields (`primary`, `primary_light`, `secondary`, `light_grey`, `grey`, `dark`, `white`, `gradient_from`, `gradient_to` and the named hues) remain available and are derived from the same tokens.

See Also

`reflow_schemes()` for listing all available schemes.

Examples

```
s <- reflow_scheme("clinical")
s$accent
s$palette
```

reflow_schemes *List Available Colour Schemes*

Description

Prints the CTTIR v5 topic families with their labels and accents, followed by the earlier scheme names that remain accepted as aliases, and returns the family keys invisibly.

Usage

```
reflow_schemes()
```

Value

Invisibly returns a character vector of the nine v5 family keys.

Examples

```
reflow_schemes()
```

reflow_validate_strict *Validate a Strict Project Against an External Declaration*

Description

Reads files without executing project code. The declaration and reviewed hashes must come from a trusted location outside the generated project. A project file cannot approve itself. Static validity, execution readiness and recorded completion are separate results; none constitutes scientific or legal approval.

Usage

```
reflow_validate_strict(
  directory,
  declaration,
  reviewed_files = NULL,
  execution_receipts = NULL
)
```

Arguments

directory	Existing strict project directory.
declaration	The declaration returned by reflow_scaffold(), retained externally. Contains profile, name, author, dependency_lock and, only for strict-cttir-v1, the exact composition.
reviewed_files	Optional named character vector mapping declared stage Rmd/R paths to externally reviewed SHA-256 hashes. Control files are immutable. The composed cttiR profile refuses all reviewed-file overrides.
execution_receipts	Optional named character vector mapping stage IDs to externally retained SHA-256 hashes of their execution receipts.

Value

A list with valid, pristine, execution_ready, completed, per-stage results and diagnostic checks. Structural errors return valid=FALSE.

Examples

```
pins <- c(digest = "0.6.37", jsonlite = "1.8.9", knitr = "1.49",
          rmarkdown = "2.29", workflowr = "1.7.1")
lock <- list(
  R = list(Version = "4.4.2"),
  Packages = lapply(stats::setNames(names(pins), names(pins)), function(p) {
    list(Package = p, Version = pins[[p]], Source = "Repository")
  })
)
profile <- reflow_profile("clinical", "secondary", "observational")
project <- tempfile("strict-", tmpdir = normalizePath(tempdir(), winslash = "/", mustWork = TRUE))
scaffold <- reflow_init_strict(project, profile, "Example study", "Your Name", lock)

# Keep the declaration outside the project and validate against it
result <- reflow_validate_strict(project, scaffold$declaration)
result[c("valid", "pristine", "execution_ready", "completed")]

Sys.chmod(file.path(project, "data", "raw"), "0755")
unlink(project, recursive = TRUE)
```

Index

base::signalCondition(), *25*

reflow_artifact_plan, 3
reflow_artifact_plan(), *4*
reflow_artifact_plan_resume
(reflow_artifact_plan_run), 4
reflow_artifact_plan_resume_bounded
(reflow_artifact_plan_run_bounded), 5
reflow_artifact_plan_run, 4, *6*
reflow_artifact_plan_run_bounded, 5
reflow_artifact_ref, 7
reflow_artifact_ref(), *15*, *17*
reflow_artifact_resume
(reflow_artifact_run), 8
reflow_artifact_resume(), *4*
reflow_artifact_resume_bounded
(reflow_artifact_run_bounded), 9
reflow_artifact_run, 8
reflow_artifact_run_bounded, *6*, 9
reflow_artifact_select, 12
reflow_artifact_selection_verify, 13, *16*
reflow_artifact_spec, 14
reflow_artifact_spec(), *9*, *16*
reflow_artifact_stage, 15
reflow_artifact_stage(), *3*, *17, 18*
reflow_artifact_target, 16
reflow_call_stage, 17
reflow_call_stage(), *18*
reflow_call_writer, 18
reflow_call_writer(), *17*
reflow_figure_plot, 19
reflow_figure_recipe, 20
reflow_figure_recipe(), *19*
reflow_icon_base64, 23
reflow_imaging_input
(reflow_imaging_ref), 24
reflow_imaging_input(), *3*, *14*, *17*, *23*
reflow_imaging_plan, 23
reflow_imaging_plan(), *25*
reflow_imaging_ref, 24
reflow_imaging_resume
(reflow_imaging_run), 25
reflow_imaging_run, 25
reflow_imaging_stage, 26
reflow_init, 27
reflow_init_strict, 29
reflow_preset, 31
reflow_presets, *29*, 31
reflow_presets(), *31*
reflow_preview, 32
reflow_profile, 32
reflow_profile(), *30*, *34*
reflow_scaffold, 34
reflow_scaffold(), *37*
reflow_scheme, 35
reflow_scheme(), *32*
reflow_schemes, *28, 29*, 36
reflow_schemes(), *35*
reflow_validate_strict, 36

wflow_build, *29*
wflow_publish, *29*
wflow_start, *27*, *29*