Package: fscontext (via r-universe)

July 7, 2026

Title File System Contextualisation and Record Set Reconstruction

Version 0.2.0

Language en-GB

Description Provides a provenance-aware framework for contextual reconstruction from file systems and related digital resource collections. The package creates reproducible snapshots of file-level metadata, paths, repository context, and optional content signatures. It supports contextual grouping, structural abstraction, temporal analysis, semantic stabilization, duplicate and reuse detection, and lightweight workflow reconstruction from file system observations. The framework deliberately separates observational evidence, contextual abstraction, semantic interpretation, and analytical reconstruction, enabling reproducible workflows that can be inspected by reviewers. It is designed to support future alignment with archival and contextual knowledge representation models, including the World Wide Web Consortium Provenance Ontology (PROV-O): Lebo et al. (2013) <https://www.w3.org/TR/prov-o/> and Records in Contexts developed by the International Council on Archives Expert Group on Archival Description (EGAD) <https://www.ica.org/ica-network/expert-groups/egad/records-in-contexts-ric/>.

License GPL (>= 3)

URL https://fscontext.dataobservatory.eu/

BugReports https://github.com/dataobservatory-eu/fscontext/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

Imports digest, fs, progress, dplyr, utils, rlang, purrr, dataset, stats, tools, glue, tibble, tidyr, magrittr, stringr, labelled, jsonlite

Suggests knitr, rmarkdown, spelling, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Depends R (>= 4.1.0)

LazyData true

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libx11-dev zlib1g-dev

Repository https://dataobservatory-eu.r-universe.dev

Date/Publication 2026-06-29 16:00:41 UTC

RemoteUrl https://github.com/dataobservatory-eu/fscontext

RemoteRef HEAD

RemoteSha a4d9846718cc7f3ca1dbdc29bdf28f8056ea25e9

Contents

`add_snapshot_context` *Add contextual identifiers to snapshot observations*

Description

Enriches filesystem observations with deterministic contextual identifiers used for longitudinal, cross-storage, and forensic reconstruction workflows.

Usage

```
add_snapshot_context(df)
```

Arguments

`df`	A snapshot `data.frame` created by `scan_storage()` or `read_snapshot()`.

Details

The function preserves the original observational rows and adds contextual identifiers derived from:

- storage context;
- filesystem location;
- observation time.

The package deliberately separates filesystem observation from later analytical interpretation and documentary aggregation.

The function therefore:

- enriches filesystem observations with contextual identifiers;
- supports repeated observation tracking across time;
- supports comparison across storage systems;
- does not construct Record Sets or higher-level documentary aggregations.

The added identifiers support:

- reconstruction of distributed work environments;
- provenance-aware analytical workflows;
- longitudinal filesystem analysis;

- cross-storage comparison of observations.

Added variables:

- `storage_full_path`: globally contextualised filesystem locator (`storage_id::full_path`);
- `storage_path_id`: deterministic storage-scoped filesystem identifier (`storage_id::rel_path`);
- `observation_id`: deterministic identifier of a specific filesystem observation, combining storage context, relative path, and observation time.

Value

A `data.frame` enriched with additional contextual identifier variables.

Examples

```
data("fscontextdemo_snapshot_02")

snapshots <- add_snapshot_context(fscontextdemo_snapshot_02)

head(
  snapshots[
    ,
    c(
      "storage_full_path",
      "storage_path_id",
      "observation_id"
    )
  ]
)
```

`as_character` *Semantic character coercion*

Description

Convert objects into semantic character representations.

Usage

```
as_character(x, ...)
```

Arguments

`x`	An object.
`...`	Additional arguments.

Details

`as_character()` creates operational semantic workspaces suitable for:

- contextual refinement;
- semantic stabilization;
- provenance-aware harmonisation;
- contextual reconstruction workflows.

Unlike base `as.character()`, semantic operationalisation preserves observational provenance and semantic vocabulary.

Value

A semantic operationalisation of `x`.

See Also

`prelabel()`, `refine()`, `dataset::as_character()`

`as_value_key` *Standardize contextual semantic mappings*

Description

Convert contextual semantic mapping carriers into canonical key-value representations suitable for:

Usage

```
as_value_key(x)
```

Arguments

`x`	A semantic mapping carrier: • named character vector; • named list; • two-column data frame or tibble.

Details

- contextual reconstruction;
- Record Set projection;
- workflow annotation;
- semantic stabilisation;

- lightweight semantic harmonisation.

@details

`as_value_key()` standardizes:

- named vectors;
- named lists;
- two-column tibbles or data frames.

Named lists may represent one-to-many contextual mappings, allowing multiple contextual roots to share a semantic grouping.

The resulting object is a named character vector compatible with:

- `prelabel()`;
- contextual semantic overlays;
- refinement workflows;
- lightweight contextual harmonisation.

Value

A named character vector representing canonical contextual semantic mappings.

See Also

`dataset::as_value_key()`

Examples

```
record_sets <- list(
  conceptualisation = c(
    "D:/_package/alpha",
    "D:/_markdown/alpha-methodology"
  ),
  betaR = c(
    "D:/_packages/beta",
    "D:/_packages/prebeta"
  )
)

as_value_key(record_sets)

invert_value_key(record_sets)
```

`classify_operational_file_type` *Classify operational file types*

Description

`classify_operational_file_type()` assigns observed files to broad operational categories such as code, data, documents, generated website files, and other artefacts.

The current implementation provides an `"r_development"` profile derived from software development, reproducible research, and analytical repository workflows.

The broader objective of operational classification is to introduce an intermediate semantic layer between filesystem observations and contextual reconstruction.

Usage

```
classify_operational_file_type(
  x,
  extension = "extension",
  profile = "r_development"
)
```

Arguments

`x`	A `data.frame` or tibble containing observed files.
`extension`	Character scalar identifying the column that contains file extensions. Defaults to `"extension"`.
`profile`	Character scalar defining the classification profile. Currently only `"r_development"` is implemented.

Details

Classify observed resources into operational categories that support contextual reconstruction and Record Set derivation.

In many digital collections, archives, and research environments, files participate in operational roles that cannot be inferred from directory structure alone. Examples include source materials, preservation masters, derivative artefacts, metadata records, rights documentation, analytical outputs, and publication-ready resources.

Operational classification provides a lightweight mechanism for assigning observed resources to such workflow-oriented categories before higher-level contextualisation, Record Set construction, or semantic stabilisation takes place.

Classification is currently based on extension patterns and workflow-specific heuristics. The function does not inspect file contents, infer authoritative media types, determine archival significance, or perform provenance reasoning.

The resulting classifications should therefore be interpreted as operational hypotheses that support exploration, reconstruction, and contextualisation workflows rather than authoritative documentary assertions.

Future classification profiles may support archival, audiovisual, heritage, research-data, and Records in Contexts workflows, where operational roles provide an important bridge between low-level filesystem observations and higher-level documentary interpretation.

Value

A character vector of operational file type labels.

Possible values for the `"r_development"` profile include:

code R source files.

markdown Markdown, R Markdown, or Quarto files.

workspace R workspace or serialized R objects.

data Tabular spreadsheet or delimited data files.

artifact Image artefacts.

document PDF or word-processing documents.

website_generated Generated website assets.

other Files not matched by the selected profile.

Examples

```
toy_files <- tibble::tibble(
  extension = c("R", "qmd", "csv", "png", "woff2")
)

classify_operational_file_type(
  toy_files,
  profile = "r_development"
)
```

`compile_rulebook` *Compile a semantic refinement rulebook*

Description

Compile a tidy semantic refinement rulebook into grouped refinement specifications suitable for `refine_by_rulebook()` workflows.

Usage

```
compile_rulebook(rulebook)
```

Arguments

`rulebook`	A data frame or tibble containing semantic refinement rules. The input must contain: • `"refine_id"`: refinement stage identifier; • `"variable"`: matching variable; • `"match"`: matching semantics; • `"pattern"`: matching pattern; • `"refined_assertion"`: refined semantic assertion.

Details

`compile_rulebook()` transforms a long-form rule table into a lightweight operational structure for iterative semantic stabilization.

Each compiled refinement stage contains:

- matching variables;
- matching semantics;
- observational matching patterns;
- refined semantic assertions.

Rulebooks support workflows where semantic interpretations are progressively stabilized through deterministic contextual matching rather than fully predefined ontology structures.

The function intentionally preserves tidy relational semantics in the rulebook representation while creating a lightweight operational structure for iterative semantic refinement.

Rulebooks are designed to support workflows where semantic stabilization emerges gradually through deterministic matching operations rather than through fully predefined ontological structures.

Value

A list of compiled semantic refinement specifications suitable for `refine_by_rulebook()`.

Each list element contains:

- `"refine_id"`: refinement stage identifier;
- `"rules"`: wide matching rule table;
- `"by"`: matching variables;
- `"match"`: matching semantics;
- `"assertion"`: refined semantic assertion.

See Also

`refine()`, `refine_by_rulebook()`

Examples

```
rulebook <- data.frame(
  refine_id = c(
    "refine_1",
    "refine_1",
    "refine_2"
  ),
  variable = c(
    "extension",
    "filename",
    "extension"
  ),
  match = c(
    "exact",
    "starts_with",
    "exact"
  ),
  pattern = c(
    "png",
    "film",
    "csv"
  ),
  refined_assertion = c(
    "visualisation",
    "visualisation",
    "tabular_data"
  ),
  stringsAsFactors = FALSE
)

compile_rulebook(rulebook)
```

---

construct_structural_paths

Construct recursive contextual structural paths

Description

Construct normalized context-relative structural paths from filesystem snapshot observations.

Usage

```
construct_structural_paths(snapshot, contexts)
```

Arguments

snapshot	A filesystem snapshot tibble containing at least: • full_path

- `rel_path`

`contexts` A contextual reconstruction object containing contextual roots.

Details

The function creates an intermediate structural abstraction layer between:

- observational filesystem paths;
- contextual semantic interpretation.

Observations are:

- matched to contextual roots;
- normalized into context-relative structural paths;
- recursively expanded into hierarchical structural components.

This enables:

- contextual filesystem diagnostics;
- recursive structural matching;
- contextual coverage analysis;
- semantic refinement workflows.

For example:

`"R/import/helpers.R"` expands into:

- `"R"`
- `"R/import"`

`"tests/testthat/test-import.R"` expands into:

- `"tests"`
- `"tests/testthat"`

The function explicitly separates:

`rel_path` Storage-relative observational filesystem path.

`structural_path` Context-relative normalized structural path.

`explored_path` Recursively expanded structural abstraction layer.

Value

A tibble containing recursively expanded contextual structural paths.

The returned tibble includes:

context Contextual ecosystem identifier.

root Matched contextual root.

rel_path Original storage-relative observational path.

structural_path Context-relative normalized structural path.

explored_path Recursively expanded structural abstraction.

Examples

```
data("fscontextdemo_snapshot_02")

mini_context <- list(
  alpha = "D:/_packages/fscontextdemo"
)

mini_snapshot <- fscontextdemo_snapshot_02[
  c(1, 3, 5, 10),
]

structural_paths <- construct_structural_paths(
  snapshot = mini_snapshot,
  contexts = mini_context
)

structural_paths[, c("context", "structural_path", "explored_path")]
```

`context_roots` *Extract normalized contextual roots*

Description

Returns a normalized character vector of contextual roots from either:

Usage

```
context_roots(x)
```

Arguments

`x`	Character vector of contextual roots or a named list of contextual definitions.

Details

- a character vector of contextual roots;
- a named list of contextual definitions.

The function:

- flattens contextual roots;
- normalizes filesystem separators;
- removes trailing separators;
- validates uniqueness.

This is the public interface for obtaining contextual aggregation boundaries from context definitions.

Value

A normalized character vector of unique contextual roots.

Examples

```
mini_context <- list(
  packages = c(
    "D:/packages/examplepkg",
    "C:/packages/examplepkg"
  ),
  research = "D:/research/projectA"
)

context_roots(mini_context)
```

`coverage_roots` *Evaluate contextual root coverage*

Description

Evaluates whether observational aggregation units are included in a set of contextual roots.

Usage

```
coverage_roots(provenance, roots)
```

Arguments

`provenance`	Output from `observe_universe()`.
`roots`	Character vector of contextual roots or a context object.

Details

The function operates on observational universes created by `observe_universe()` and performs scale-aware matching between:

- observational aggregation units;
- contextual roots.

Matching is performed using:

- normalized filesystem paths;
- aggregation depth.

Contextual roots and observational units are therefore only matched within the same aggregation depth.

The function does not compute coverage summaries or percentages. It only classifies observational units as included or excluded relative to contextual roots.

Value

A tibble containing observational aggregation units with contextual inclusion status.

`coverage_rules_path` *Evaluate contextual rule coverage for structural paths*

Description

coverage_rules_path() expands observed filesystem paths into recursive structural components and evaluates whether those components are covered by contextual path rules.

The function provides a diagnostic view of contextual coverage. It identifies which observed filesystem structures are recognised by a rulebook and which remain unmatched.

This makes it useful for:

- contextual reconstruction;
- semantic stabilisation workflows;
- rulebook development and refinement;
- coverage diagnostics;
- filesystem archaeology;
- exploratory analysis of operational structures.

Usage

```
coverage_rules_path(snapshot, contexts)
```

Arguments

`snapshot`	A filesystem snapshot data.frame containing at least: • full_path; • rel_path.
`contexts`	A contextual reconstruction object containing contextual roots and path rules.

Details

Evaluates how filesystem observations align with a contextual path-rule model.

Structural matching is recursive.

For example:

- "R/import/helpers.R" expands to:
- "R"
- "R/import"
- "tests/testthat/test-import.R" expands to:
- "tests"

- "tests/testthat"

Expanded structural paths are evaluated against contextual path rules associated with the matching contextual root.

The function operates on filesystem observations and explicit contextual rules. It does not infer:

- Activities;
- Events;
- Record Sets;
- provenance relationships.

Instead, it evaluates whether observed filesystem structures are covered by an existing contextual model.

Unmatched structures may indicate:

- missing contextual rules;
- evolving workflows;
- operational noise;
- areas requiring further semantic stabilisation.

Value

A tibble describing structural path-rule coverage.

The returned tibble includes:

context Contextual environment associated with the matched rule.

root Contextual root used for evaluation.

explored_path Recursively expanded structural path.

matched_rule Path rule evaluated against the structural path.

activity Rule outcome associated with the matched rule.

matched Logical indicator showing whether the structural path is covered by the rule.

See Also

`construct_structural_paths()`, `refine_by_rulebook()`, `compile_rulebook()`

Examples

```
small_snapshot <- tibble::tibble(
  full_path = c(
    "D:/packages/fscontext/R/import/helpers.R",
    "D:/packages/fscontext/tests/testthat/test-import.R",
    "D:/packages/fscontext/data-raw/input.csv"
  ),
  rel_path = c(
    "R/import/helpers.R",
    "tests/testthat/test-import.R",
```

```
    "data-raw/input.csv"
  )
)

small_context <- list(
  contexts = list(
    fscontext = list(
      roots = "D:/packages/fscontext",
      rules = list(
        path = c(
          "R" = "software_development",
          "tests/testthat" = "unit_testing",
          "data-raw" = "etl"
        )
      )
    )
  )
)

coverage_rules_path(
  snapshot = small_snapshot,
  contexts = small_context
)
```

`derive_record_set` *Derive contextual Record Set membership*

Description

Assign observational units to one or more contexts using declared filesystem boundaries.

Usage

```
derive_record_set(
  x,
  contextual_groups,
  observed_unit_var = "observed_unit",
  include_subfolders = TRUE
)
```

Arguments

`x`	A data frame containing observational units, typically created with `observe_universe()`.
`contextual_groups`	A data frame defining contextual boundaries. Must contain: **context** Context identifier.

root Filesystem root used to derive membership.

`observed_unit_var`
: Name of the column containing observational units. Defaults to "observed_unit".

`include_subfolders`
: Logical. Currently retained for future compatibility. Membership is derived recursively.

Details

derive_record_set() creates a contextual membership layer over an observational universe. Membership is derived by matching observational units against one or more declared context roots.

The function is intended as an intermediate step between filesystem observation and Record Set construction.

In a software repository, contexts may correspond to projects, packages, or reporting workflows. In archival environments, contexts may correspond to collections, fonds, or other documentary aggregations.

Membership is currently derived using recursive path-prefix matching.

Value

A tibble containing observational units assigned to one or more contexts.

Additional variables include:

context Context identifier.

context_root Root used for membership derivation.

construction_method Membership derivation method.

derived_by Function that created the assignment.

derived_at Timestamp of derivation.

Examples

```
toy_universe <- tibble::tibble(
  observed_unit = c(
    "D:/projects/eviota",
    "D:/projects/eviota/tests",
    "D:/other"
  ),
  inst_id = c("a", "b", "c")
)

contextual_groups <- tibble::tibble(
  context = "eviota",
  root = "D:/projects/eviota"
)

derive_record_set(
  toy_universe,
  contextual_groups
```

```
)
```

`derive_structural_groups` *Derive structural aggregation metadata from relative paths*

Description

Derives lightweight structural aggregation metadata from observed relative filesystem paths.

The function identifies recurring structural patterns in directory hierarchies and creates candidate aggregations that can support exploratory analysis, navigation, contextual reconstruction, and later semantic interpretation.

The resulting groupings are derived solely from path structure. They are analytical projections rather than authoritative Record Sets, provenance assertions, or documentary relationships.

Usage

```
derive_structural_groups(rel_path, profile = "folder-depth-2")
```

Arguments

`rel_path`	Character vector of relative filesystem paths.
`profile`	Character scalar specifying the structural aggregation strategy. Available profiles are: **"folder-depth-1"** Group by the first directory level. **"folder-depth-2"** Group by the first two directory levels (default). **"folder-depth-3"** Group by the first three directory levels. **"folder-depth-4"** Group by the first four directory levels. **"wacz"** Use the first path component as the structural group and the second component as the structural subdivision, matching the standard organisation of WACZ archives.

Details

Structural aggregation metadata provides a lightweight abstraction of observed directory organisation. It can increase the informativeness of filesystem observations by exposing recurring organisational patterns without asserting semantic meaning.

Within the fscontext workflow:

- filesystem observations provide evidence about observed resources;
- relative paths provide structural organisation;
- structural aggregations expose candidate groups that may later support contextual reconstruction, Record Set construction, semantic stabilisation, or other downstream analyses.

Future versions may introduce additional aggregation profiles based on repository structure, provenance, temporal patterns, or other observational evidence.

Value

A `data.frame` with two columns:

structural_group Candidate structural aggregation derived from the selected path profile.

component Immediate structural subdivision within the aggregation, when present.

Examples

```
rel_path <- c(
  "_packages/demo/R/file.R",
  "_packages/demo/tests/testthat/test-file.R",
  "_packages/demo/data/input.csv"
)

derive_structural_groups(rel_path)

derive_structural_groups(
  rel_path,
  profile = "folder-depth-1"
)

derive_structural_groups(
  c(
    "archive/data.warc.gz",
    "indexes/index.cdx",
    "pages/pages.jsonl"
  ),
  profile = "wacz"
)
```

---

detect_generated_artifacts

Detect operationally generated or low-priority artifacts

Description

Detects generated, transient, synchronized, or operationally low-priority resources commonly encountered in filesystem-based reconstruction and preservation workflows.

The function is designed for provenance-aware analytical pipelines where generated artifacts may otherwise:

- inflate duplication metrics;
- obscure meaningful reconstruction signals;
- introduce synchronization noise;
- or reduce review efficiency.

Typical examples include:

- generated website assets;
- transient editor files;
- synchronization metadata;
- cached rendering artifacts;
- font and frontend dependencies;
- local workspace history files.

The function intentionally performs lightweight operational classification rather than authoritative preservation appraisal.

It is designed to work together with:

- `read_snapshot()`
- `snapshot_to_reconstruction_context()`
- `classify_operational_file_type()`

as part of layered provenance-aware reconstruction workflows.

Usage

```
detect_generated_artifacts(
  x,
  filename = "filename",
  extension = "extension",
  ignored_names = c(".Rhistory", ".RData", ".gitignore", ".DS_Store", "dir.c9r",
    "masterkey.cryptomator", "vault.cryptomator"),
  ignored_extensions = c("css", "js", "map", "woff", "woff2", "ttf", "c9r")
)
```

Arguments

`x`	A `data.frame` or tibble containing observed resources.
`filename`	Character scalar identifying the column containing filenames. Defaults to `"filename"`.
`extension`	Character scalar identifying the column containing file extensions. Defaults to `"extension"`.
`ignored_names`	Character vector of filenames commonly treated as generated, synchronized, transient, or operational noise.
`ignored_extensions`	Character vector of file extensions commonly associated with generated or low-priority artifacts.

Details

The function intentionally uses lightweight operational heuristics.

It does not:

- inspect file contents;

- infer preservation value;
- determine archival significance;
- perform semantic interpretation;
- replace curatorial review.

Classification is based primarily on:

- filename heuristics;
- extension heuristics;
- operational workflow conventions.

Future versions may support:

- workflow-specific profiles;
- preservation-oriented review vocabularies;
- institution-specific ignore registries;
- synchronized workspace heuristics.

Value

A logical vector indicating whether each resource is likely to represent a generated or operationally low-priority artifact.

Examples

```
toy_files <- tibble::tibble(
  filename = c(
    ".Rhistory",
    "app.css",
    "analysis.R",
    "font.woff2"
  ),
  extension = c(
    "",
    "css",
    "R",
    "woff2"
  )
)

detect_generated_artifacts(
  toy_files
)
```

`exclude_operational_noise` *Exclude operational noise from analytical workflows*

Description

Excludes operationally low-value system and workflow artifacts from analytical reconstruction workflows while preserving the original observational evidence.

The function is designed for provenance-aware analytical pipelines where certain operational artifacts may:

- inflate duplication metrics;
- distort reuse analysis;
- obscure meaningful reconstruction patterns;
- or reduce review efficiency.

Unlike destructive filtering, the function is intended to operate on contextual or analytical reconstruction layers after observational evidence has already been preserved.

This distinction is important for:

- forensic reproducibility;
- provenance-aware reconstruction;
- archival transparency;
- and Heritage Digital Twin workflows.

The function supports lightweight operational noise profiles.

Current built-in profiles include:

- `"generic"`
- `"rstudio"`

The `"generic"` profile targets common system and synchronization artifacts.

The `"rstudio"` profile targets operational artifacts commonly produced during R and RStudio workflows.

Future versions may support:

- workflow-specific profiles;
- institution-specific registries;
- YAML-based noise vocabularies;
- preservation-oriented filtering policies.

The function is designed to work together with:

- `read_snapshot()`
- `snapshot_to_reconstruction_context()`

as part of layered provenance-aware reconstruction workflows.

Usage

```
exclude_operational_noise(
  x,
  filename = "filename",
  extension = "extension",
  profiles = c("generic", "rstudio")
)
```

Arguments

`x`	A `data.frame` or tibble containing contextual or analytical reconstruction entities.
`filename`	Character scalar identifying the filename column. Defaults to `"filename"`.
`extension`	Character scalar identifying the extension column. Defaults to `"extension"`.
`profiles`	Character vector defining operational noise profiles to apply. Current profiles include: • `"generic"` • `"rstudio"`

Details

The function intentionally excludes only operationally low-priority resources.

It does not:

- delete observational evidence;
- modify the original snapshot data;
- infer preservation value;
- determine archival significance;
- replace curatorial review.

Resources excluded from analytical workflows may still remain important for:

- forensic preservation;
- synchronization reconstruction;
- reproducibility auditing;
- or operational environment analysis.

Value

A filtered `data.frame` excluding operational noise resources.

Examples

```
toy_files <- tibble::tibble(
  filename = c(
    ".DS_Store",
    ".Rhistory",
    "analysis.R",
    "report.qmd"
  ),
  extension = c(
    "",
    "",
    "R",
    "qmd"
  )
)

exclude_operational_noise(
  toy_files,
  profiles = c(
    "generic",
    "rstudio"
  )
)
```

`fscontextdemo_snapshot_01`

Example filesystem snapshot 01

Description

A small filesystem snapshot included in `fscontextdemo` for demonstrating provenance-aware filesystem reconstruction workflows with `fscontext`.

Usage

```
fscontextdemo_snapshot_01
```

Format

A data frame containing filesystem observations and contextual metadata.

Details

The snapshot contains observational filesystem metadata, timestamp information, file signatures, and Git repository context collected from the `fscontextdemo` package itself.

The dataset is intentionally small but structurally realistic, making it suitable for:

- vignettes

- tests
- provenance reconstruction examples
- contextual event projection
- Git-aware filesystem analysis

The dataset contains 78 file observations.

Source

Generated with `fscontext::snapshot_storage()` from the `fscontextdemo` package source tree.

`fscontextdemo_snapshot_02`

Second example filesystem snapshot

Description

A second observational snapshot of the `fscontextdemo` package created after additional artefacts and semantic enrichment workflows were added.

Usage

```
fscontextdemo_snapshot_02
```

Format

A data frame containing filesystem observations and snapshot-level provenance metadata.

Details

The dataset supports:

- longitudinal filesystem comparison
- event projection
- provenance reconstruction
- workflow archaeology

Source

Generated with `fscontext::snapshot_storage()` from the evolving `fscontextdemo` package source tree.

`invert_contextual_grouping` *Invert contextual grouping mappings*

Description

`invert_contextual_grouping()` transforms lightweight contextual grouping definitions into a two-column table of group membership.

This is useful when contextual roots, resources, or candidate Record Set members are first declared as a named list but later need to be used in joins, rulebooks, or reconstruction workflows.

Usage

```
invert_contextual_grouping(x)
```

Arguments

`x`	A named list. Each list name identifies a contextual group, and each list element contains one or more members of that group.

Details

Convert a named list of contextual groupings into a long-form relational table.

The function performs a lightweight structural transformation.

It does not validate ontology semantics, enforce uniqueness, construct graph objects, or infer hierarchical relations.

The result is suitable for relational operations such as joins, filtering, contextual reconstruction, and candidate Record Set membership workflows.

Value

A tibble with two columns:

group Contextual grouping identifier.

member Member associated with the contextual group.

See Also

`as_value_key()`, `invert_value_key()`

Examples

```
record_sets <- list(
  conceptualisation = c(
    "D:/_packages/alpha",
    "D:/_markdown/alpha-methodology"
  ),
  beta = c(
    "D:/_packages/beta",
    "D:/_packages/prebeta"
  )
)

invert_contextual_grouping(record_sets)
```

`invert_value_key` *Invert contextual semantic mappings*

Description

Convert contextual semantic mappings into a two-column relational representation suitable for:

Usage

```
invert_value_key(x)
```

Arguments

x	A semantic mapping carrier: • named character vector; • named list; • two-column data frame or tibble.

Details

- joins;
- contextual grouping workflows;
- Record Set inspection;
- relational projection;
- roundtrip conversion with `as_value_key()`.

Value

A two-column tibble containing contextual semantic mappings.

See Also

as_value_key() dataset::invert_value_key()

Examples

```
record_sets <- list(
  conceptualisation = c(
    "D:/_package/alpha",
    "D:/_markdown/alpha-methodology"
  )
)

invert_value_key(record_sets)
```

---

is.prelabelled *Test if a vector is prelabelled*

Description

Determine whether an object inherits from the "prelabelled" class.

Usage

```
is.prelabelled(x)
```

Arguments

x	An object.

Details

Useful for lightweight semantic stabilization and contextual reconstruction workflows where observational evidence and semantic assertions remain explicitly separated.

Value

Logical scalar.

See Also

prelabel(), dataset::is.prelabelled()

`observe_universe` *Construct a longitudinal observational universe*

Description

Aggregates repeated filesystem observations into a lightweight longitudinal observational universe.

Usage

```
observe_universe(
  snapshot_dir,
  max_aggregation_depth = 2,
  by_storage = TRUE,
  by_person = FALSE,
  exclude_patterns = c("\\.gitignore$", "\\.Rbuildignore$", "\\.github$",
    "\\.quarto$", "\\.Rcheck$", "\\.RDataTmp", "\\.Trash-1000",
    "\\.cryptomator$", "\\.editorconfig$", "\\.gitattributes$",
    "\\.webmanifest$")
)
```

Arguments

`snapshot_dir`	Directory containing snapshot `.rds` files.
`max_aggregation_depth`	Integer giving the maximum filesystem path depth used to derive observational aggregation units.
`by_storage`	Logical. If `TRUE`, aggregation preserves `storage_id`.
`by_person`	Logical. If `TRUE`, aggregation preserves `person_id`.
`exclude_patterns`	Character vector of regular expressions used to exclude operational artefacts from observational aggregation units. Defaults exclude common: • hidden metadata folders; • temporary artefacts; • generated build artefacts; • repository management files. Exclusions are applied after aggregation-unit derivation and before longitudinal summarisation.

Details

The function operates on snapshot `.rds` files created by `scan_storage()` and summarises repeated observations of operational filesystem aggregation units across time.

The resulting table is intentionally observational and pre-interpretive:

- no intellectual Record Sets are inferred;
- no semantic reconciliation is performed;
- no provenance assertions beyond observation are made.

Instead, the function provides a lightweight observational universe suitable for:

- reconstruction workflows;
- audit preparation;
- preservation planning;
- storage coverage analysis;
- identifying candidate contextual Record Sets;
- longitudinal filesystem observation.

Observational aggregation units are operationally approximated from observed file paths using configurable path truncation rules.

Aggregation units derived at different aggregation depths are not directly comparable.

Single files are never treated as aggregation units.

Aggregation may optionally preserve:

- storage boundaries (`storage_id`);
- person boundaries (`person_id`).

Value

A tibble containing longitudinal observational summaries of filesystem aggregation units.

Variables include:

observed_unit Operational filesystem aggregation unit derived from path truncation.

aggregation_depth Actual observed filesystem depth of the aggregation unit.

max_aggregation_depth Maximum filesystem path depth used during aggregation.

n_observations Number of snapshot observations in which the aggregation unit appeared.

avg_files_unit Average number of files observed per snapshot for the aggregation unit.

avg_size_unit Average observed size in bytes per snapshot for the aggregation unit.

avg_size_mb_unit Average observed size in megabytes per snapshot for the aggregation unit.

avg_size_gb_unit Average observed size in gigabytes per snapshot for the aggregation unit.

total_files_unit Total files observed for the aggregation unit across all snapshots.

total_size_unit Total bytes observed for the aggregation unit across all snapshots.

Examples

```
data("fscontextdemo_snapshot_01")
data("fscontextdemo_snapshot_02")

tmp <- tempfile()
dir.create(tmp)

saveRDS(
  fscontextdemo_snapshot_01,
  file.path(tmp, "snapshot_01.rds")
)

saveRDS(
  fscontextdemo_snapshot_02,
  file.path(tmp, "snapshot_02.rds")
)

observation_universe <- observe_universe(
  snapshot_dir = tmp,
  max_aggregation_depth = 2
)

head(observation_universe)
```

---

observe_wacz *Observe a WACZ web archive*

Description

Creates an observational data frame from a WACZ web archive.

The function extracts structural metadata from the archive, combines page-level information with WARC index metadata, and returns one observational row for each archived web page.

The resulting object represents observations only. It intentionally avoids making semantic assertions about Records, Record Parts, Instantiations, or other archival entities. Such interpretation can be added later with `wacz_to_recordset_df()` or downstream semantic enrichment workflows.

Usage

```
observe_wacz(wacz)
```

Arguments

`wacz`	Path to a `.wacz` archive.

Details

The function performs the following steps:

- extracts the WACZ archive into a temporary directory;
- reads the archive `datapackage.json`;
- parses page metadata from `pages/pages.jsonl`;
- parses WARC index metadata from `indexes/index.cdx`;
- collapses multiple archived versions of the same resource;
- joins page observations with archive metadata.

The resulting observations preserve the evidence contained in the archive without interpreting its archival semantics.

Value

A tibble containing observations extracted from the archive.

The returned object carries two attributes:

- `datapackage`, containing the parsed `datapackage.json` metadata supplied by the WACZ archive;
- `wacz`, containing the normalized path to the source archive.

Typical variables include:

- page identifiers;
- resource locators (URLs);
- page titles;
- timestamps;
- extracted text;
- text signatures;
- MIME types;
- WARC digests;
- archive offsets;
- version counts.

References

The WACZ format specification: https://specs.webrecorder.net/wacz/1.1.1/

See Also

`wacz_to_recordset_df()`

Examples

```
wacz <- system.file("testdata", "fscontext_020.wacz", package = "fscontext")

observe_wacz(wacz)
```

`prelabel` *Create a prelabelled vector*

Description

Attach provisional semantic assertions to an observational vector.

Usage

```
prelabel(x, labels, unmatched = "keep", missing_label = "<NA>")
```

Arguments

`x`	A vector.
`labels`	Candidate semantic mappings describing provisional semantic assertions. `labels` is internally normalised with `as_value_key()` and may therefore be supplied as: • a named vector; • a named list; • a two-column data frame or tibble.
`unmatched`	Behaviour for unmatched observational values. One of: • `"keep"` (default): preserve unmatched values as observational semantic assertions; • `"na"`: operationalise unmatched values as `NA` during semantic coercion.
`missing_label`	Semantic assertion used internally for missing observational values.

Details

`prelabel()` creates lightweight semantic mappings that support iterative semantic refinement workflows before values mature into formally defined variables created with `labelled::labelled()` or `dataset::defined()`.

Unlike strictly defined semantic vectors, `prelabelled` vectors tolerate:

- incomplete semantic mappings;
- unresolved observational values;
- contextual ambiguity;
- incremental semantic stabilisation.

This design is particularly useful in provenance-aware, contextual reconstruction, and archival workflows where semantic interpretations emerge gradually through iterative refinement.

The class supports workflows inspired by RiC-O and PROV-O, where observational evidence and semantic interpretation remain explicitly separated.

`prelabelled` vectors intentionally separate:

- observational evidence;
- semantic operationalisation;
- contextual semantic refinement.

Original observational values remain unchanged while semantic assertions may evolve through iterative refinement workflows.

Semantic operationalisation is available through:

- `as.character()` for lightweight semantic coercion;
- `as_character()` for provenance-preserving semantic workspaces.

Value

A vector with:

- class `"prelabelled"`;
- attached provisional semantic vocabulary stored in the `"prelabel"` attribute.

References

Lebo, T., Sahoo, S., McGuinness, D. et al. (2013). PROV-O: The PROV Ontology. https://www.w3.org/TR/prov-o/

International Council on Archives Expert Group on Archival Description (2023). Records in Contexts (RiC). https://www.ica.org/ica-network/expert-groups/egad/records-in-contexts-ric/

See Also

`dataset::prelabel()`, `dataset::defined()`, `dataset::as_value_key()`

Examples

```
x <- c("R", "png", "csv", "unknown")

extension_map <- c(
  R = "functional_programming",
  png = "visualisation",
  csv = "tabular_data"
)

x <- prelabel(x, labels = extension_map)

x

as.character(x)

semantic_workspace <- as_character(x)

attributes(semantic_workspace)
```

`quick_signature` *Compute a fast operational signature for a file*

Description

Generates a lightweight content signature by hashing sampled byte regions from a file. The signature provides a fast operational approximation for detecting identical or differing file instances without computing a full cryptographic hash.

Usage

```
quick_signature(path, n = 1024)
```

Arguments

`path`	Character. Path to the file.
`n`	Integer. Number of bytes sampled from selected regions (default: 1024).

Details

The function is designed for large-scale observational workflows where complete file hashing would be unnecessarily expensive.

The signature is constructed by hashing sampled byte regions:

- small files: full file content
- medium files: beginning and end
- large files: beginning, middle and end

The resulting signature is intended as a fast observational aid:

- identical signatures suggest identical file content;
- differing signatures indicate differing file content;
- collisions are possible but unlikely in operational use.

Missing or inaccessible files return `NA_character_`.

The signature does not establish authoritative identity or provenance. It provides lightweight observational evidence that may support later contextual reconstruction, duplicate detection, version analysis, or Record Set construction.

Unlike `quick_signature_text()`, this function operates on the binary representation of a file rather than its textual content.

Value

Character. A lightweight operational signature.

See Also

`quick_signature_text()`, `scan_storage()`, `summarise_duplicates()`

`quick_signature_text` *Compute a fast operational signature for text*

Description

Generates a lightweight content signature by hashing sampled character regions from one or more text strings. The signature provides a fast approximation for detecting identical or differing textual content without comparing complete strings.

Usage

```
quick_signature_text(x, n = 1024)
```

Arguments

`x`	Character vector.
`n`	Integer. Number of characters sampled from selected regions (default: 1024).

Details

The function is intended for observational workflows where textual representations have already been extracted from digital resources, such as HTML pages, OCR output, PDFs, or office documents.

The signature is constructed by hashing sampled character regions:

- short texts: complete text;
- medium texts: beginning and end;
- long texts: beginning, middle and end.

The resulting signature is intended as a fast observational aid:

- identical signatures suggest identical textual content;
- differing signatures indicate differing textual content;
- collisions are possible but unlikely in operational use.

Missing values return `NA_character_`. Empty strings return `"empty"`.

Unlike `quick_signature()`, this function operates on extracted text rather than binary file content. Consequently, different file formats (for example DOCX, PDF and HTML) containing the same textual content may produce identical text signatures while retaining different file signatures.

The function provides lightweight observational evidence that may support duplicate detection, content reconciliation, semantic stabilisation, or later contextual reconstruction.

Value

Character vector of operational signatures that summarises the observed textual representation of a resource.

See Also

quick_signature(), observe_wacz()

---

read_snapshot *Read and combine observational filesystem snapshots*

Description

read_snapshot() reconstructs an observational layer from one or more snapshots previously created with scan_storage().

The function preserves filesystem observations as originally recorded, while appending snapshot-level provenance and contextual identifiers that support longitudinal and cross-storage analytical workflows.

The resulting table is intended to represent observed filesystem Instantiations rather than authoritative documentary entities.

Usage

```
read_snapshot(snapshot_files, include_repo_metadata = FALSE)
```

Arguments

snapshot_files	Character vector of snapshot .rds files.
include_repo_metadata	Logical. If TRUE, repository metadata stored in snapshot attributes are materialized into the returned observational table. The following repository-level variables may be added: • git_remote • git_branch • git_repo_id This may increase memory usage because repository metadata are repeated across all observations belonging to the same repository.

Details

Read one or more serialized observational filesystem snapshots and combine them into a unified observational table.

The function performs:

- observational aggregation across snapshots
- snapshot-level provenance preservation
- contextual identifier enrichment
- optional materialization of repository-level Git metadata

The function intentionally does not:

- deduplicate observations
- infer stable file identity
- infer Record Resources or Record Sets
- resolve documentary semantics
- interpret provenance relationships

Multiple observations of the same filesystem approximation may occur:

- across observation times
- across storage contexts
- across partially overlapping snapshots
- across synchronized or copied working environments

In RiC-aligned operational terms:

- each row represents one observed filesystem Instantiation
- repeated observations may later support inference of more stable Record Resources
- higher-level documentary interpretation is deferred to later analytical or curatorial stages

Snapshot-level provenance metadata are appended as columns to support:

- provenance-aware analytics
- reconstruction workflows
- cross-storage comparison
- longitudinal temporal analysis

Repository metadata are normally stored as snapshot attributes in order to avoid repeating identical repository information across all observations. When `include_repo_metadata = TRUE`, repository-level metadata are materialized into the returned table to support repository-aware analytical workflows.

Value

A `data.frame` containing combined filesystem observations.

The returned table contains all variables created by `scan_storage()` together with additional provenance and contextual identifiers:

- `snapshot_file`: normalized path of the source snapshot artefact
- `snapshot_created_at`: observation timestamp recorded in snapshot metadata
- `snapshot_schema_version`: schema version recorded in snapshot metadata
- `storage_full_path`: globally contextualized filesystem locator (`storage_id::full_path`)
- `storage_path_id`: storage-scoped logical filesystem identifier (`storage_id::rel_path`)
- `observation_id`: identifier of a specific filesystem observation event, combining storage context, logical path, and observation time

recordset_df *Create a semantically annotated Record Set*

Description

Create a recordset_df, a lightweight extension of dataset::dataset_df for representing archival Record Sets.

A recordset_df stores dataset-level metadata together with optional Record and Record Part identifiers using lightweight conventions inspired by the Records in Contexts (RiC) model. It supports provenance-aware archival, curatorial and semantic enrichment workflows while remaining compatible with ordinary data frames.

See the **"Working with Record Sets"** vignette for a complete workflow starting from filesystem observations.

Usage

```
recordset_df(
  x,
  title = NULL,
  creator = utils::person("Jane", "Doe"),
  description = NULL,
  record_set_identifier = NULL,
  record_identifier = NULL,
  record_part_identifier = NULL,
  record_subject = "Record Set",
  ...
)
```

Arguments

x	A data.frame or dataset_df.
title	Character scalar giving the title of the Record Set.
creator	A utils::person() object describing the creator of the Record Set metadata.
description	Optional description of the Record Set.
record_set_identifier	Optional identifier of the Record Set.
record_identifier	Name of the column containing Record identifiers. The selected column is annotated as rico:Identifier and labelled "Record Identifier".
record_part_identifier	Name of the column containing Record Part identifiers. The selected column is annotated as rico:Identifier and labelled "Record Part Identifier".
record_subject	Subject term describing the Record Set. Defaults to "Record Set".
...	Reserved for future extensions.

Value

A `recordset_df`, which inherits from `dataset_df`, `tbl_df`, `tbl` and `data.frame`.

References

International Council on Archives Expert Group on Archival Description (2023). Records in Contexts (RiC). https://www.ica.org/ica-network/expert-groups/egad/records-in-contexts-ric/

See Also

`dataset::dataset_df()`, `observe_wacz()`, `wacz_to_recordset_df()`

Examples

```
x <- data.frame(
  resource_locator = c(
    "https://example.org/1", "https://example.org/2"
  ),
  filename = c("a.html", "b.html"),
  stringsAsFactors = FALSE
)

rs <- recordset_df(
  x,
  title = "Demo Record Set",
  creator = utils::person("Joe", "Doe", role = "aut"),
  record_identifier = "resource_locator",
  record_part_identifier = "filename"
)

rs
```

---

refine *Refine semantic assertions through contextual matching*

Description

`refine()` incrementally stabilizes semantic assertions through deterministic contextual matching rules while preserving row cardinality and the original observational universe.

Usage

```
refine(x, target = NULL, rules, by, assertion, comment = NULL, match = "exact")
```

Arguments

x	A data frame or tibble.
target	Name of the target column to refine.
rules	A rule table or compiled rulebook.
by	Optional grouping variables used during refinement.
assertion	Optional assertion text recorded in provenance.
comment	Optional comment attached to the refinement step.
match	Matching strategy. Defaults to "first".

Details

The function is designed for lightweight semantic refinement workflows where semantic interpretations mature gradually through ordinary tidyverse operations.

Matching observations are identified through configurable matching semantics applied to one or more observational variables.

Supported matching semantics include:

- "exact" relational equality;
- "starts_with" hierarchical prefix matching;
- "ends_with" suffix matching;
- "contains" substring detection.

Matching positions in the target vector are replaced by refined semantic assertions.

Unmatched values remain unchanged.

refine() intentionally never:

- removes rows;
- reshapes tables;
- modifies unrelated observations.

This makes refinement stages auditable, reversible, and compatible with iterative semantic stabilization workflows.

refine() operates on semantic operationalisations produced through workflows such as:

- prelabel();
- as.character();
- as_character();
- previous refinement stages.

Rather than enforcing formally complete ontology semantics, the function provides a lightweight operational mechanism for progressively stabilizing semantic interpretations inside ordinary analytical workflows.

Multiple refinement stages may later mature into:

- controlled vocabularies;
- `labelled::labelled()` vectors;
- `dataset::defined()` vectors;
- semantically enriched datasets
- compatible with iterative semantic workflows.

`refine()` operates on semantic operationalisations produced through workflows such as:

- `prelabel()`;
- `as.character()`;
- `as_character()`;
- earlier refinement stages.

The function does not attempt to construct formally complete semantic graphs or enforce ontology-level consistency.

Instead, it provides a lightweight operational mechanism for progressively stabilizing semantic interpretations inside ordinary tidyverse workflows.

This approach is particularly useful when working with:

- partially harmonised datasets;
- inconsistent coding systems;
- ambiguous metadata;
- hierarchical filesystem structures;
- exploratory semantic reconstruction workflows.

Multiple refinement stages may later mature into:

- controlled vocabularies;
- formally defined semantic vectors;
- semantically enriched datasets;
- or graph-based semantic representations.

Value

A character vector.

See Also

`refine_by_rulebook()`, `compile_rulebook()`

Examples

```
files <- tibble::tibble(
  filename = c("filmA.png", "filmB.png", "film.xlsx", "fill.png"),
  extension = c("png", "png", "xlsx", "png")
)

out <- refine(
  x = files,
  target =
    rep("unresolved", nrow(files)),
  rules = tibble::tibble(
    filename = "film",
    extension = "png"
  ),
  by = c("filename", "extension"),
  match = c("starts_with", "exact"),
  assertion = "film_visualisation"
)

out
```

`refine_by_rulebook` *Iteratively refine semantic assertions using a rulebook*

Description

Apply a compiled semantic rulebook sequentially to a target semantic assertion vector.

Usage

```
refine_by_rulebook(
  x,
  target,
  rulebook,
  prefix = NULL,
  keep_intermediate = TRUE,
  final_name = "final"
)
```

Arguments

`x`	A data frame.
`target`	Bare or quoted name of the initial semantic assertion column.
`rulebook`	A compiled rulebook object created with `compile_rulebook()`.
`prefix`	Prefix used for generated refinement columns. Defaults to the target column name.

keep_intermediate
: Logical. If TRUE, keeps all intermediate refinement columns. If FALSE, returns only the final refinement column.

final_name
: Name of the final refinement column.

Details

refine_by_rulebook() wraps an iterative refinement workflow where each semantic refinement stage operates on the semantic state produced by the previous stage.

The function is designed for operational semantic stabilization workflows where semantic assertions progressively mature through deterministic contextual refinement.

Each refinement stage creates a new refinement column, preserving semantic progression for inspection, auditing, and review.

Rulebooks are typically created with compile_rulebook() and operate on semantic operationalisations produced with:

- prelabel();
- as.character();
- as_character();
- refine().

Conceptually, the function behaves similarly to an iterative purrr::reduce() workflow applied to semantic refinement stages.

Value

A tibble with progressively refined semantic assertions.

See Also

refine(), compile_rulebook()

---

save_scan *Persist an observational snapshot to disk*

Description

Saves a snapshot dataset (typically produced by scan_storage()) as a uniquely identified .rds file.

Usage

```
save_scan(df, storage_id, path, label = NULL)
```

Arguments

df	Data.frame. Scan result with created with scan_storage() that has a created_at attribute.
storage_id	Character. Identifier of the storage.
path	Character. Directory where the file will be saved.
label	Character. Optional human-readable label describing the scanned scope (e.g. "d_eviota"). Default is NULL (no label).

Details

The filename encodes:

- storage context
- observation time
- optional scope label
- a deterministic hash

This ensures:

- chronological ordering
- uniqueness
- reproducibility of stored observations

Value

Invisibly returns the full file path of the saved .rds file.

Examples

```
tmp_dir <- tempfile()
dir.create(tmp_dir)

dir.create(file.path(tmp_dir, "R"))
dir.create(file.path(tmp_dir, "data"))

file.create(file.path(tmp_dir, "R", "a.R"))
file.create(file.path(tmp_dir, "R", "b.R"))
file.create(file.path(tmp_dir, "data", "c.csv"))

scan <- scan_storage(
  root = tmp_dir,
  storage_id = "test-storage"
)

save_scan(
  df = scan,
  storage_id = "test-storage",
  path = tmp_dir
)
```

scan_storage *Observe a filesystem and construct a reproducible snapshot*

Description

Recursively scans a root folder and returns a data.frame where each row represents one filesystem observation recorded at a specific time.

Usage

```
scan_storage(
  root,
  storage_id = "local-storage",
  person_id = "local-user",
  scan_time = Sys.time(),
  compute_signature = TRUE,
  max_signature_size = 200 * 1024 * 1024
)
```

Arguments

root	Character. Path to the root folder to observe.
storage_id	Character. Identifier of the storage context.
person_id	Character. Identifier of the observer or operator.
scan_time	POSIXct. Timestamp of the observation. Defaults to Sys.time() if not provided.
compute_signature	Logical. Whether to compute lightweight content signatures.
max_signature_size	Numeric. Maximum file size (bytes) for signature computation.

Details

The function implements a read-only filesystem observation model:

- it records accessible filesystem state;
- it does not interpret file contents;
- it does not assume canonical, complete, or authoritative state.

Each observation records:

- a relative filesystem locator (rel_path);
- a storage context (storage_id);
- an observation timestamp (scan_time).

Additional metadata may include:

- filesystem properties (size, timestamps, permissions);
- optional content signatures (`quick_sig`);
- repository and version-control context (`repo_root`, `repo_rel_path`, `git_tracked`).

The package deliberately records filesystem observations first and postpones documentary interpretation, Record Set construction, and RiC-aligned semantic assertions to later analytical stages.

This creates a reproducible observational snapshot suitable for:

- forensic analysis of development environments;
- reconstruction of activity patterns;
- audit and compliance workflows;
- alignment with version-controlled repositories.

The returned dataset at minimum contains:

- `rel_path`: relative filesystem locator within the observed root;
- `storage_path_id`: deterministic storage-scoped identifier derived from `storage_id::rel_path`;
- `filename`: basename of the observed file;
- `mtime`: last modification timestamp;
- `extension`: file extension.

Additional variables may be present depending on scan configuration.

The function is:

- read-only and non-destructive;
- deterministic for a given filesystem state;
- robust to inaccessible files, which are silently skipped.

The result represents observed filesystem state rather than complete historical provenance.

Value

A `data.frame` where each row represents one filesystem observation.

See Also

`snapshot_storage()`

Examples

```
root <- system.file(
  "testdata/minimal_R_folder",
  package = "fscontext"
)

snapshot <- scan_storage(root)

subset(
```

```
  snapshot,
  rel_path %in% c(
    "DESCRIPTION",
    "NAMESPACE",
    "R/hello_world.R",
    "vignettes/demo.Rmd"
  )
)[, c("rel_path", "extension")]
```

`snapshot_storage` *Create and persist an observational snapshot of a filesystem*

Description

Executes `scan_storage()` and stores the resulting observational dataset as a timestamped `.rds` snapshot.

Usage

```
snapshot_storage(
  root,
  storage_id = "local-storage",
  person_id = "user",
  scan_time = Sys.time(),
  label = NULL,
  path = tempdir(),
  compute_signature = TRUE,
  max_signature_size = 200 * 1024 * 1024
)
```

Arguments

`root`	Character. Path to the root folder to scan.
`storage_id`	Character. Identifier of the storage.
`person_id`	Character. Identifier of the person.
`scan_time`	POSIXct. Timestamp of the scan.
`label`	Character. Optional human-readable label describing the scanned scope (e.g. "d_eviota").
`path`	Character. Directory where snapshots are stored.
`compute_signature`	Logical. Whether to compute fast file signatures.
`max_signature_size`	Numeric. Maximum file size in bytes for signatures.

Details

Each snapshot captures the **state of file instantiations at a specific point in time**, preserving:

- file-level metadata
- structural context
- globally contextualised file paths (`storage_id::local_path`)
- optional content signatures
- repository associations

Snapshots are intended as:

- durable audit artefacts
- inputs for longitudinal analysis
- reproducible evidence of observed environments

Since schema version 0.1.3, snapshots store `full_path` values as globally contextualised paths:

`storage_id::local_filesystem_path`

This prevents collisions between similar local folder structures observed on different machines or storage contexts.

Value

Invisibly returns the path to the stored snapshot.

Invisibly returns the full path to the saved snapshot.

Examples

```
root <- tempfile()
dir.create(root)

dir.create(file.path(root, "R"))
dir.create(file.path(root, "data"))

file.create(file.path(root, "R", "a.R"))
file.create(file.path(root, "R", "b.R"))
file.create(file.path(root, "data", "c.csv"))

snapshot_storage(
  root = root,
  storage_id = "test-storage",
  path = root
)
```

`snapshot_to_reconstruction_context` *Reconstruct a contextual observational Record Set from filesystem snapshots*

Description

Reconstructs a contextual observational corpus from one or more filesystem snapshot fragments.

The function:

1. merges observational filesystem snapshots;
2. filters observations to selected contextual roots;
3. enriches observations with contextual identifiers;
4. derives lightweight structural grouping heuristics;
5. creates a contextual Record Set projection.

The workflow is optimized for:

- forensic reconstruction;
- filesystem archaeology;
- exploratory analytical workflows;
- development environment reconstruction;
- operational reporting.

Unlike `snapshot_to_recordset_df()`, this function intentionally prioritizes analytical reconstruction over preservation-oriented semantic assertions.

Usage

```
snapshot_to_reconstruction_context(
  snapshot_files,
  roots,
  exclude_patterns = "\\.Rcheck"
)
```

Arguments

`snapshot_files`	Character vector of `.rds` snapshot files created with `snapshot_storage()`.
`roots`	Character vector of contextual root paths used for observational selection.
`exclude_patterns`	Character vector of exclusion patterns passed to `subset_snapshot()`.

Details

Snapshot fragments are merged observationally.

Duplicate filesystem observations are intentionally preserved because the same resource may legitimately appear across:

- multiple machines;
- multiple storage contexts;
- repeated scans;
- synchronised working environments.

The resulting object remains observational and analytical.

Structural grouping heuristics are lightweight filesystem-derived operational projections and do not imply authoritative archival Record Set construction.

The function serves as the foundational reconstruction layer for:

- analytical enrichment workflows;
- reconstruction reporting;
- semantic preservation wrappers such as `snapshot_to_recordset_df()`.

Value

A contextual observational reconstruction table enriched with:

- contextual observational identifiers;
- storage-aware path identifiers;
- structural grouping heuristics;
- lightweight contextual Record Set projections.

Core observational variables typically include:

- `storage_id`;
- `person_id`;
- `full_path`;
- `rel_path`;
- `filename`;
- `extension`;
- `mtime`;
- `scan_time`.

Contextual enrichment variables may include:

- `inst_id`;
- `storage_path_id`;
- `observation_id`;

- structural_group;
- component;
- record_set_identifier;
- resource_id;
- locator_path.

See Also

snapshot_to_recordset_df(), subset_snapshot(), add_snapshot_context(), add_structural_groups().

Examples

```
data("fscontextdemo_snapshot_01")

tmp <- tempfile(fileext = ".rds")
saveRDS(fscontextdemo_snapshot_01, tmp)

snapshot_to_reconstruction_context(
  snapshot_files = tmp,
  roots = "D:/_packages/fscontextdemo/R"
)
```

---

snapshot_to_recordset_df

Create a contextual Record Set dataset

Description

Creates a provenance-aware recordset_df() object from observational filesystem snapshots and contextual reconstruction workflows.

The function preserves observed filesystem resources while adding:

- contextual Record Set assertions (human-defined grouping of related archived digital resources);
- dataset-level provenance metadata (information about how, when, and from which observations the dataset was created);
- preservation-oriented semantic context (structured contextual information supporting archival, audit, and long-term reconstruction workflows).

Unlike snapshot_to_reconstruction_context(), which is optimized for analytical and forensic workflows, this function creates a stable contextual preservation object suitable for:

- contextual digital preservation;
- audit reconstruction;
- heritage and archival workflows;
- provenance-aware digital collections;
- human-in-the-loop semantic enrichment.

Usage

```
snapshot_to_recordset_df(
  snapshot_files,
  roots,
  record_set_identifier,
  record_set_title = NULL,
  creator = utils::person("Jane", "Doe", role = "aut"),
  exclude_patterns = c("\\\\.Rcheck")
)
```

Arguments

`snapshot_files`	Character vector of `.rds` snapshot files.
`roots`	Character vector of contextual root paths used for observational selection.
`record_set_identifier`	Character scalar giving the asserted identifier of the resulting Record Set.
`record_set_title`	Optional human-readable title.
`creator`	A `utils::person()` object describing the creator of the semantic Record Set assertion.
`exclude_patterns`	Character vector of exclusion patterns passed to `subset_snapshot()`.

Details

The function intentionally reuses `snapshot_to_reconstruction_context()` to preserve:

- identical observational reconstruction logic;
- stable contextual identifiers;
- reproducible reconstruction workflows.

The resulting object keeps observational rows intact while adding a lightweight semantic preservation layer based on:

- contextual Record Set assertions;
- provenance metadata;
- RiC-aligned contextual semantics.

Value

A semantically enriched `recordset_df` object inheriting from `dataset_df`.

References

International Council on Archives Expert Group on Archival Description (2023). Records in Contexts (RiC). https://www.ica.org/ica-network/expert-groups/egad/records-in-contexts-ric/

`subset_snapshot` *Subset observational filesystem Instantiations*

Description

Selects observations from a snapshot according to structural criteria such as paths, extensions, or exclusion patterns.

Usage

```
subset_snapshot(
  snapshot_path,
  folder_path,
  extensions = NULL,
  exclude_patterns = c("\\.Rcheck")
)
```

Arguments

`snapshot_path`	Character. Path to `.rds` snapshot file.
`folder_path`	Character vector. One or more folder roots.
`extensions`	Optional character vector of file extensions (no dot).
`exclude_patterns`	Optional regex patterns to exclude paths.

Details

The function performs observational selection only and does not derive Record Sets, contextual hierarchies, or analytical groupings.

The function returns a subset of the original snapshot while preserving its observational provenance (e.g. `created_by`, `created_at`).

In addition, it derives a `rel_root_path` column, which represents the path of each file relative to the matched filter root. When multiple `folder_path` values are provided, the deepest matching root is used.

The `rel_root_path` is a context-dependent projection intended for grouping, navigation, and reporting. It is not a stable identifier and should not be used for joins or identity; use `rel_path` for that purpose.

Value

data.frame filtered snapshot with `rel_root_path`

Examples

```
data("fscontextdemo_snapshot_02")

tmp <- tempfile(fileext = ".rds")
saveRDS(fscontextdemo_snapshot_02, tmp)

subset_snapshot(
  snapshot_path = tmp,
  folder_path = "D:/_packages/fscontextdemo/R"
)
```

---

summarise_duplicates *Summarise repeated and divergent filesystem observations*

Description

Aggregates observed filesystem observations by filename and lightweight content signatures (`quick_sig`).

Usage

```
summarise_duplicates(df)

summarize_duplicates(df)
```

Arguments

`df`	A snapshot `data.frame` conforming to the canonical snapshot schema created by `scan_storage()` or `read_snapshot()`. The dataset must contain: • `filename` • `quick_sig` (may contain `NA`)

Details

The function identifies:

- repeated identical observations;
- potentially synchronised copies;
- diverging versions of similarly named resources;
- distributed working duplicates.

The function operates on observational filesystem evidence only.

It does not:

- infer authoritative file identity;
- establish Record Resource equivalence;

- reconstruct provenance lineage;
- determine curatorial relationships.

In RiC-aligned operational terminology:

- rows in the snapshot represent filesystem observations;
- repeated identical `quick_sig` values provide operational evidence that multiple observations may correspond to the same underlying digital resource;
- differing signatures associated with the same filename may indicate divergent versions, forks, or independently evolving resources.

The function therefore supports:

- longitudinal reconstruction;
- distributed workflow analysis;
- duplicate detection;
- exploratory Record Set construction;
- provenance-aware analytical workflows.

Duplicate observations are not inherently anomalous.

In distributed development workflows the same file may legitimately appear:

- across multiple machines;
- across synchronised project folders;
- in backup or staging locations;
- in derived analytical Record Sets.

The function therefore reports observational duplication rather than asserting erroneous copying.

The function treats:

- `filename` as a weak identity signal;
- `quick_sig` as a lightweight content equivalence signal.

Missing signatures (`NA`) are treated as a valid observational group.

This means:

- multiple `NA` signatures are considered identical;
- a mix of `NA` and non-`NA` signatures counts as versioning.

The function operates on observational snapshots and does not resolve identity across time or storage contexts.

Value

A `data.frame` with one row per `filename`.

The returned variables include:

filename File basename used as grouping key.

total_copies Total number of observed filesystem occurrences.

identical_copies Size of the largest identical-signature group.

versioned_copies Number of observations outside the largest identical-signature group.

n_versions Number of distinct observed signatures.

See Also

`quick_signature()`

Examples

```
data("fscontextdemo_snapshot_01")
data("fscontextdemo_snapshot_01")

combined_snapshot <- rbind(
  fscontextdemo_snapshot_01,
  fscontextdemo_snapshot_01
)

summarise_duplicates(combined_snapshot)
```

---

summarise_observed_activity

Summarise observed activity from filesystem observations

Description

`summarise_observed_activity()` summarises observed file-level modification evidence by time period and structural grouping.

The function is intended as a temporal contextualisation step. It does not infer archival Activities, Events, Record Sets, or provenance relations. Instead, it produces candidate activity summaries that may support later human review, semantic stabilisation, or RiC-aligned modelling.

Usage

```
summarise_observed_activity(
  df,
  extensions = c("r", "bak"),
  path_col = "rel_path",
  time_unit = c("week", "month", "day", "year"),
```

```
  max_files = 20
)

summarize_observed_activity(
  df,
  extensions = c("r", "bak"),
  path_col = "rel_path",
  time_unit = c("week", "month", "day", "year"),
  max_files = 20
)
```

Arguments

df	A `data.frame` representing filesystem observations. The data must contain: • `rel_path`; • `filename`; • `mtime`; • `extension`. If present, `git_tracked` is used to count untracked files.
extensions	Character vector of file extensions to include, without leading dots. Matching is case-insensitive.
path_col	Character scalar. Name of the column containing paths used to derive structural groupings. Defaults to `"rel_path"`.
time_unit	Character scalar. One of `"week"`, `"month"`, `"day"`, or `"year"`.
max_files	Integer. Maximum number of file names displayed in each summary row.

Details

Aggregates filesystem observations into reproducible temporal summaries grouped by structural path context.

The function derives:

- `period`, a time bucket derived from file modification times (`mtime`);
- `group_path`, a structural grouping key derived from the selected path column.

It then summarises observations within each `period` x `group_path` combination.

Modification times are treated as observational evidence of change. They are not interpreted as complete editing histories or confirmed archival events.

Structural grouping is deterministic and based on path structure. It is intended for aggregation, review, and reporting, not for file identity. Use `rel_path` for file-level identity.

Typical uses include:

- identifying temporal clusters of filesystem activity;
- reviewing candidate Activities before semantic stabilisation;
- comparing activity across structural contexts;

- supporting archival recontextualisation workflows;
- preparing analytical or audit summaries.

Files under `.Trash` are excluded.

Value

A `data.frame` with one row per `period` and `group_path` combination.

The returned columns include:

period Time bucket identifier.

group_path Structural grouping key derived from `path_col`.

start Earliest observed modification date in the group.

end Latest observed modification date in the group.

file_names Pipe-separated sample of observed file names.

n_files Number of file observations in the group.

n_unique_files Number of distinct paths in the group.

untracked Number of observations not tracked by Git, when `git_tracked` is available; otherwise `NA_integer_`.

Examples

```
data("fscontextdemo_snapshot_02")

summarise_observed_activity(
  fscontextdemo_snapshot_02,
  time_unit = "month"
)
```

---

wacz_to_recordset_df *Create a Record Set dataset from a WACZ observation*

Description

Converts a `wacz_observation` created with `observe_wacz()` into a semantically enriched `dataset_df` representing a Record Set.

The function preserves the original observations while attaching dataset-level metadata and lightweight Records in Contexts (RiC) semantics. Selected identifier columns may be declared as `rico:Identifier` values, allowing downstream workflows to distinguish identifiers intended to refer to Records or Record Parts without requiring a complete RiC-O implementation.

The function intentionally performs only lightweight semantic enrichment. It does not infer Records, Record Parts, Instantiations, or other archival entities, nor does it reconcile identities or build provenance graphs. Such interpretation is expected to occur in later human-guided curation or semantic stabilisation workflows.

Usage

```
wacz_to_recordset_df(
  wacz_observation,
  record_set_id = NULL,
  record_set_title = NULL,
  record_identifier = "resource_locator",
  record_part_identifier = NULL,
  person = utils::person("Jane", "Doe")
)
```

Arguments

wacz_observation	A wacz_observation object created with observe_wacz().
record_set_id	Optional identifier for the resulting Record Set. If NULL, the basename of the WACZ archive (without extension) is used.
record_set_title	Optional human-readable title for the Record Set. If omitted, a title is constructed automatically.
record_identifier	Name of the column whose values identify Records represented in the Record Set. The selected column is annotated as rico:Identifier using dataset::defined(). Set to NULL to skip annotation.
record_part_identifier	Optional name of a column whose values identify Record Parts. The selected column is annotated as rico:Identifier.
person	A utils::person() object describing the creator of the resulting dataset metadata.

Details

This function occupies the boundary between observational data and semantic interpretation.

observe_wacz() records observations extracted from a WACZ archive. wacz_to_recordset_df() adds curatorial assertions describing how particular observed identifiers should be interpreted within a Record Set, while deliberately avoiding stronger ontological commitments such as identity reconciliation or Record construction.

The resulting object is intended for reproducible archival, curatorial, and semantic enrichment workflows.

Value

A dataset_df object enriched with:

- Dublin Core dataset metadata;
- a RiC Record Set subject;
- optional semantic annotations for Record and Record Part identifiers;
- the original datapackage and wacz attributes.

References

International Council on Archives Expert Group on Archival Description (2023). Records in Contexts (RiC). https://www.ica.org/ica-network/expert-groups/egad/records-in-contexts-ric/

See Also

observe_wacz(), dataset::dataset_df(), dataset::defined()

Index