Package: fmridataset (via r-universe)

October 16, 2026

Type Package

Title Spatially Typed Annotated fMRI Datasets

Version 0.11.0.9000

Description Provides spatially typed, annotated observation-by-feature containers for functional magnetic resonance imaging (fMRI) data. The package aligns lazy numerical assays with observation metadata, feature metadata, experimental entities, and explicit neuroimaging spaces while retaining compatibility with NIfTI, HDF5, BIDS, and in-memory sources.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.3.0)

Imports delarr (>= 0.2.0), digest, fs, Matrix, methods, neuroim2, rlang, stats, stringi (>= 1.8.0), tibble, utils, uuid

Suggests albersdown, bidser (>= 0.5.0), blosc, fmrihrf, fmrilatent, fmrigds, fmristore (>= 0.1.0.9000), hdf5r, jsonlite, knitr, multidesign, neuroatlas, neurosurf, rmarkdown, testthat (>= 3.0.0), withr, zarr (>= 0.4.2)

VignetteBuilder knitr

Remotes bbuchsbaum/delarr@4e0cdf6c7560d6c230d04bf0ca36c6c70179782b, bbuchsbaum/albersdown@54a7432e551413fd15253544d361295d3d65f2b5, bbuchsbaum/fmrigds@67cc585c53ecb74db3b9a405b0fcb67b26c4eb3c, bbuchsbaum/fmrilatent@ceaad61acea6fd6326d6978a0aad23090ec28b18, bbuchsbaum/fmristore@e7fbbfca181ebd118acee150774a4f58640e039d, bbuchsbaum/multidesign@2ed3f30cdce1d96541bb84b34e0826cc0b8a298a, bbuchsbaum/neuroatlas@9e469f21d6c3248910d9e2d57ce18e945b8bd16c, bbuchsbaum/neurosurf@1c4dc595876a619545670122d1dd4244e19ae3e0

Additional_repositories https://bbuchsbaum.r-universe.dev

URL https://github.com/bbuchsbaum/fmridataset, https://bbuchsbaum.github.io/fmridataset/

BugReports https://github.com/bbuchsbaum/fmridataset/issues

Config/Needs/website albersdown

Config/testthat/edition 3

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

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

Date/Publication 2026-10-11 00:35:14 UTC

RemoteUrl https://github.com/bbuchsbaum/fmridataset

RemoteRef HEAD

RemoteSha 58c9f4c389e3885c652d844c6d82ecf20a4357da

Contents

aligned_assay_set . . . 5
append_source_shards . . . 5
apply_feature_validity . . . 6
array-source . . . 7
as_delarr . . . 9
as_fmri_frame . . . 10
auxiliary-table . . . 10
auxiliary_table . . . 11
axis_block . . . 12
axis_frame . . . 13
basis-operators . . . 14
basis_space . . . 15
basis_space_from_decoder . . . 16
basis_space_from_fmrilatent . . . 17
bind_observations . . . 18
block_apply . . . 19
block_manifest . . . 20
canonical-encoding . . . 20
canonicalization_contract . . . 21
collect_assay . . . 22
collect_spatial_maps . . . 23
collection-accessors . . . 24
collection-spaces . . . 24
collection_digest . . . 25
compose_frame_links . . . 26
composite-parts . . . 26
composite_space . . . 27
content_hash . . . 28
content_hash_contract . . . 30
counting_source . . . 30
entity-accessors . . . 31
entity-frame-accessors . . . 32
entity_feature_validity . . . 33

entity_frame . . . 34
entity_registry . . . 34
entity_registry_digest . . . 35
event-accessors . . . 36
event_table . . . 36
execute_block_plan . . . 37
execute_spatial . . . 38
execution_path . . . 39
explain . . . 40
fault_source . . . 41
fds_frame_bindings . . . 41
fds_frame_manifest . . . 42
fds_manifest_digest . . . 43
fds_schema . . . 44
fds_study_bindings . . . 44
fds_study_manifest . . . 45
fds_study_manifest_digest . . . 46
fds_study_representations . . . 46
feature-map-accessors . . . 47
feature-space . . . 48
feature-validity-accessors . . . 49
feature_axis . . . 50
feature_map . . . 51
feature_map_from_target . . . 52
feature_mapped_source . . . 53
filter_entities . . . 53
filter_obs . . . 54
fmri_collection . . . 55
fmri_frame . . . 56
fmri_study . . . 57
frame-accessors . . . 58
frame-schema-validation . . . 59
frame_from_fds_manifest . . . 60
frame_link . . . 61
frame_schema . . . 62
hierarchy-accessors . . . 63
hierarchy_digest . . . 64
hierarchy_index . . . 65
id-policy . . . 66
identity_descriptor . . . 66
index_space . . . 67
key_relation . . . 68
locate_source_rows . . . 69
map_features . . . 69
mask-bank-accessors . . . 70
mask_bank . . . 71
memory_source . . . 72
nifti_array_source . . . 73

nifti_source_space . . . 74
observation_validity . . . 75
parcel-operators . . . 76
parcel_space . . . 76
parcel_space_from_atlas . . . 78
plan_blocks . . . 79
provenance-graph . . . 80
provenance_record . . . 81
read_bids_bold . . . 81
relation-accessors . . . 83
relation_registry . . . 84
relation_registry_digest . . . 84
row_sharded_source . . . 85
select_features . . . 86
shard_manifest . . . 86
source_descriptor . . . 87
source_error . . . 88
source_view . . . 89
space . . . 89
sparse_relation . . . 90
spatial_map . . . 91
study-accessors . . . 92
study-registries . . . 92
study_digest . . . 93
study_from_fds_manifest . . . 94
surface_space . . . 95
surface_space_from_neurosurf . . . 96
temporal-schema . . . 97
unaligned_record . . . 99
validate_event_table . . . 100
validate_fmri_collection . . . 100
validate_fmri_study . . . 101
validate_relation_registry . . . 102
validate_unaligned_record . . . 102
validity_coverage . . . 103
validity_masked_source . . . 104
volume_space . . . 105
write_frame . . . 105
zarr_array_source . . . 106

Index **108**

aligned_assay_set *Construct a strictly aligned assay set*

Description

Construct a strictly aligned assay set

Usage

```
aligned_assay_set(assays, observations, features)
```

Arguments

assays	Named matrices or ArraySource descriptors.
observations	Observation axis.
features	Feature axis.

Value

An aligned_assay_set.

Examples

```
sp <- volume_space(dim = c(2, 2, 2), affine = diag(4))
obs <- axis_frame(data.frame(.obs_id = sprintf("vol-%d", 1:4)), axis = "observation")
feat <- feature_axis(feature_data(sp), space = sp)
bold <- matrix(rnorm(4 * n_features(sp)), nrow = 4)
aligned_assay_set(list(bold = bold), obs, feat)
```

append_source_shards *Append immutable source shards*

Description

Append immutable source shards

Usage

```
append_source_shards(x, sources, shard_ids = NULL, shard_data = NULL)
```

Arguments

x	An existing row_sharded_source.
sources	New compatible child sources.
shard_ids	Stable IDs for the new shards.
shard_data	Optional metadata for the new shards. Its columns must match existing shard metadata.

Value

A new `row_sharded_source`; `x` and its child descriptors are not modified.

Examples

```
src <- row_sharded_source(list(memory_source(matrix(1:4, nrow = 2))))
grown <- append_source_shards(src, list(memory_source(matrix(5:8, nrow = 2))))
source_shape(grown)
```

---

apply_feature_validity

Apply one validity relation lazily to frame assays

Description

Apply one validity relation lazily to frame assays

Usage

```
apply_feature_validity(x, name = NULL, assays = NULL)
```

Arguments

`x`	An `fmri_frame` or view.
`name`	Validity relation name.
`assays`	Assay names to mask. Defaults to all assays.

Value

A new frame with lazy `NA` masking and derivation provenance.

Examples

```
space <- index_space(4, ids = paste0("f", 1:4), namespace = "validity-ex")
subjects <- entity_frame(
  data.frame(subject_id = c("sub-1", "sub-2")),
  key = "subject_id"
)
validity <- entity_feature_validity(
  entity = "subject", entity_ids = c("sub-1", "sub-2"),
  masks = rbind(
    c(TRUE, TRUE, FALSE, TRUE),
    c(TRUE, FALSE, FALSE, TRUE)
  ),
  space = space
)
frame <- fmri_frame(
  assays = list(signal = matrix(1:8, nrow = 2)),
```

```
  observations = data.frame(.obs_id = c("o1", "o2"), subject_id = c("sub-1", "sub-2")),
  space = space,
  entities = list(subject = subjects),
  relations = list(
    observation_subject = key_relation("subject_id"),
    subject_feature_validity = validity
  )
)
masked_frame <- apply_feature_validity(frame)
collect_assay(masked_frame)
```

`array-source` *Serializable numerical array sources*

Description

The estimate distinguishes source storage from the R object returned by a read. Numeric source dtypes, including float16 and float32, are realized as R doubles. The conservative peak estimate adds selection, conversion, and compressed-input buffers to the retained output. It covers numerical payloads; fixed R object headers and selector metadata are outside the estimate.

Usage

```
as_array_source(x, ...)

source_shape(x, ...)

source_dtype(x, ...)

source_chunks(x, ...)

source_capabilities(x, ...)

source_fingerprint(x, ...)

source_open(x, ...)

source_read(x, observations = NULL, features = NULL, ...)

source_read_native(x, observations = NULL, ...)

source_close(x, ...)

source_realization_cost(x, observations = NULL, features = NULL)
```

Arguments

x	An array source or object coercible to one.
...	Additional method arguments.
observations	Optional observation selector: NULL for every observation, integer positions in request order, or a logical mask. Selectors follow the package's normalization law: positions must be whole numbers, may reorder, may be negative but not mixed with positive, drop zero, must be in bounds, and may not repeat an element; masks must match the axis length with no NA; an empty selection is legal. The law is checked at the generic before a method is dispatched.
features	Optional feature selector, under the same law.

Details

Fingerprint policy. source_fingerprint() returns the revision fingerprint of a descriptor: one SHA-256 string over the serializable descriptor and the physical revision evidence a backend can observe without reading values (file sizes and modification times, store metadata, or a per-object identity token for in-memory sources), combined with selectors and shard composition for wrappers. It never hashes array values, it is computed once at construction and cached in the descriptor, and it survives serialization. Equal fingerprints mean "the same descriptor of the same revision"; they never mean equal values, and different fingerprints never mean different values. Two independently constructed memory_source() objects with equal payloads have different fingerprints by design.

Value identity is a separate, explicit, O(n) operation: content_hash(). File-backed sources compare their captured revision evidence with the physical state before every open and read and raise fmridataset_error_source_stale when it differs; genuine backend failures raise fmridataset_error_backend_io. The policy is recorded in inst/architecture/ADR-009-source-fingerprints-and-content-hashes.md.

Value

A source_realization_cost list containing storage and realized dtypes, storage and output bytes, temporary buffer components, and the estimated peak bytes.

Examples

```
src <- memory_source(matrix(seq_len(6), nrow = 2))
source_shape(src)
source_dtype(src)
handle <- source_open(src)
source_read(handle, observations = 1)
source_close(handle)
```

`as_delarr` *Convert an array source to a lazy delarr array*

Description

`as_delarr()` wraps a serializable array source as a `delarr` provider so that bounded, chunk-aware execution can be delegated to `delarr` without materializing the assay. The realization budget bounds each provider pull, not the whole source, so a lazy array may describe more data than the budget allows; every pull is checked before it reads.

Usage

```
as_delarr(x, memory_budget = Inf, ...)

## Default S3 method:
as_delarr(x, memory_budget = Inf, ...)
```

Arguments

`x`	An array source, or another object with an `as_delarr()` method.
`memory_budget`	Maximum realized bytes permitted for a single pull. `Inf` disables the check.
`...`	Additional arguments passed to methods.

Value

A `delarr` lazy array whose pulls route through `source_read()`.

See Also

array-source for the source protocol.

Examples

```
src <- memory_source(matrix(seq_len(6), nrow = 2))
lazy <- as_delarr(src)
dim(lazy)
```

as_fmri_frame *Coerce an object to the canonical frame protocol*

Description

Companion packages implement methods for legacy or domain-specific objects. The generic is owned by `fmridataset` so conversions have one canonical destination and do not introduce competing frame classes.

Usage

```
as_fmri_frame(x, ...)
```

Arguments

`x`	An object convertible to an `fmri_frame`.
`...`	Method-specific arguments.

Value

An `fmri_frame`.

Examples

```
src <- memory_source(matrix(seq_len(6), nrow = 2))
obs <- tibble::tibble(.obs_id = c("o1", "o2"))
fr <- fmri_frame(list(beta = src), obs)
identical(as_fmri_frame(fr), fr)
```

auxiliary-table *Validate and inspect a typed auxiliary table*

Description

Validate and inspect a typed auxiliary table

Usage

```
validate_auxiliary_table(x)

table_data(x)

table_key(x)

table_role(x)
```

Arguments

x	An fmri_auxiliary_table.

Value

validate_auxiliary_table() returns x invisibly; other functions return table data, key, or role.

Examples

```
at <- auxiliary_table(
  data.frame(contrast = c("A-B", "B-A"), stat = c(2.1, -2.1)),
  key = "contrast", role = "contrasts"
)
validate_auxiliary_table(at)
table_key(at)
table_role(at)
```

---

auxiliary_table *Construct a typed auxiliary table*

Description

Construct a typed auxiliary table

Usage

```
auxiliary_table(data, key = NULL, role = "auxiliary", metadata = list())
```

Arguments

data	Scalar tabular data.
key	Optional stable unique-key column.
role	Stable table role such as "files", "contrasts", or "transforms".
metadata	Unaligned table-level metadata.

Value

An fmri_auxiliary_table.

Examples

```
at <- auxiliary_table(
  data.frame(contrast = c("A-B", "B-A"), stat = c(2.1, -2.1)),
  key = "contrast", role = "contrasts"
)
table_data(at)
```

`axis_block` *Construct an axis-aligned multivariate block*

Description

A block is two-dimensional: rows are the elements of the owning axis and columns are named components. Arrays with more than two dimensions are rejected because a trailing axis without typed metadata would be an anonymous semantic dimension; represent higher-order structure as named components, as several blocks, or as an assay.

Usage

```
axis_block(
  data,
  components = NULL,
  role = "continuous",
  units = NULL,
  metadata = list()
)

axis_block_data(x)

block_components(x)

block_component_ids(x)
```

Arguments

`data`	A matrix, two-dimensional lazy array, or serializable array source. Its first dimension is aligned with the owning axis and its second dimension indexes `components`.
`components`	Component metadata. The `.component_id` column is generated when absent.
`role`	Semantic role such as `"continuous"`, `"confound"`, or `"embedding"`.
`units`	Optional units applying to the block as a whole.
`metadata`	Additional serializable metadata.
`x`	An `axis_block`.

Value

An `axis_block`.

Examples

```
b <- axis_block(
  matrix(1:6, 3, 2),
  components = data.frame(.component_id = c("x", "y")),
```

```
  role = "continuous"
)
dim(axis_block_data(b))
block_component_ids(b)
```

`axis_frame` *Construct an annotated axis*

Description

Construct an annotated axis

Usage

```
axis_frame(
  data,
  blocks = list(),
  id = NULL,
  axis = c("observation", "feature", "entity", "component"),
  id_col = NULL,
  metadata = list(),
  id_policy = c("require", "deterministic", "ephemeral"),
  id_keys = NULL,
  id_namespace = NULL
)

axis_data(x)

axis_blocks(x)

axis_ids(x)
```

Arguments

`data`	A data frame with one row per axis element.
`blocks`	Named `axis_block` objects aligned on their first dimension.
`id`	Optional stable IDs.
`axis`	Axis role. Observation is the public default.
`id_col`	Name of the ID column.
`metadata`	Additional serializable metadata.
`id_policy`	ID policy. `"require"` accepts only supplied durable IDs; `"deterministic"` derives durable IDs from `id_keys` and `id_namespace`; `"ephemeral"` creates visibly marked session-only IDs that cannot be persisted or used for certified semantic identity.
`id_keys`	Columns that uniquely identify rows under deterministic policy.
`id_namespace`	Stable namespace under deterministic policy.
`x`	An `axis_frame`.

Value

An axis_frame.

Examples

```
x <- axis_frame(data.frame(value = 1:3), id_policy = "ephemeral")
axis_ids(x)
axis_data(x)
```

---

basis-operators *Inspect basis-space operators*

Description

Inspect basis-space operators

Usage

```
basis_analysis(x)

basis_synthesis(x)

basis_projection_info(x)
```

Arguments

x	A basis_space.

Value

basis_analysis() returns the parent-to-component analysis operator; basis_synthesis() returns the optional component-to-parent synthesis operator; basis_projection_info() returns validation metadata. These names deliberately avoid colliding with fmrilatent::basis_decoder(), which constructs model-level decoders.

Examples

```
parent <- volume_space(c(2, 1, 1), support = 1:2)
x <- basis_space(parent, c("c1", "c2"), diag(2), diag(2))
basis_analysis(x)
basis_synthesis(x)
basis_projection_info(x)
```

basis_space *Construct a linear basis feature space*

Description

A basis_space is a representational feature axis linked to one parent spatial space. Its encoder maps parent features to component coefficients; its optional decoder maps coefficients back to the parent. Fitting and model-specific offsets remain the responsibility of packages such as fmrilatent.

Usage

```
basis_space(
  parent,
  component_ids,
  encoder,
  decoder = NULL,
  data = NULL,
  basis_type = "linear_basis",
  provenance = list(),
  tolerance = 1e-08,
  metadata = list()
)
```

Arguments

parent	Parent feature_space represented by the basis.
component_ids	Stable component identifiers.
encoder	Component-by-parent analysis operator.
decoder	Optional parent-by-component synthesis operator.
data	One metadata row per component.
basis_type	Stable basis-family label.
provenance	Serializable derivation metadata.
tolerance	Maximum absolute error permitted when validating that encoder %*% decoder is the component-space identity.
metadata	Additional serializable metadata.

Value

A basis_space.

Examples

```
parent <- volume_space(c(2, 1, 1), support = 1:2)
x <- basis_space(parent, c("c1", "c2"), diag(2), diag(2))
n_features(x)
```

basis_space_from_decoder *Construct a basis space from a synthesis dictionary*

Description

Computes the exact unregularized least-squares encoder `(D' D)^{-1} D'` for a full-column-rank decoder `D`.

Usage

```
basis_space_from_decoder(
  parent,
  component_ids,
  decoder,
  data = NULL,
  basis_type = "linear_basis",
  provenance = list(),
  tolerance = 1e-08,
  metadata = list(),
  encoder = c("least_squares", "none")
)
```

Arguments

`parent`, `component_ids`, `decoder`, `data`, `basis_type`, `provenance`, `tolerance`, `metadata`	Passed to `basis_space()`.
`encoder`	Either `"least_squares"` to construct and validate the exact unregularized left inverse, or `"none"` for a synthesis-only basis.

Value

A `basis_space`.

Examples

```
parent <- volume_space(c(2, 1, 1), support = 1:2)
x <- basis_space_from_decoder(parent, c("c1", "c2"), diag(2))
basis_synthesis(x)
```

`basis_space_from_fmrilatent` *Adapt fmrilatent spatial loadings to a basis feature space*

Description

`fmrilatent` remains the owner of latent fitting, temporal scores, handles, and offsets. This adapter extracts only its spatial synthesis dictionary and constructs the corresponding least-squares feature-space algebra.

Usage

```
basis_space_from_fmrilatent(
  x,
  parent,
  component_ids = NULL,
  data = NULL,
  provenance = list(),
  tolerance = 1e-08
)
```

Arguments

`x`	An explicit `fmrilatent` object with `loadings()`.
`parent`	Parent feature space aligned to the loading rows.
`component_ids`	Optional stable component IDs.
`data`	Optional component metadata.
`provenance`	Additional serializable provenance.
`tolerance`	Left-inverse validation tolerance.

Value

A `basis_space`.

Examples

```
# Loading fmrilatent itself takes several seconds, so this full
# integration example is wrapped in \donttest{}.
if (requireNamespace("fmrilatent", quietly = TRUE)) {
  parent <- volume_space(c(2, 2, 1), support = 1:4, template = "toy-native")
  decoder <- Matrix::Matrix(
    matrix(c(1, 0, 0, 1, 1, 1, 2, -1), nrow = 4, byrow = TRUE),
    sparse = TRUE
  )
  scores <- Matrix::Matrix(matrix(c(1, 0, 0, 1, 2, -1), nrow = 3, byrow = TRUE))
  mask <- neuroim2::LogicalNeuroVol(
    array(TRUE, dim = c(2, 2, 1)), neuroim2::NeuroSpace(c(2, 2, 1))
```

```
  )
  latent <- fmrilatent::LatentNeuroVec(
    basis = scores, loadings = decoder,
    space = neuroim2::NeuroSpace(c(2, 2, 1, 3)), mask = mask,
    offset = rep(10, 4), meta = list(family = "toy_pca")
  )
  x <- basis_space_from_fmrilatent(latent, parent = parent)
  basis_projection_info(x)
}
```

`bind_observations` *Bind frames along observations*

Description

Bind frames along observations

Usage

```
bind_observations(
  ...,
  metadata_policy = c("identical", "merge"),
  active_assay = NULL
)
```

Arguments

`...`	Frames with identical feature IDs, spaces, and assay semantics.
`metadata_policy`	Frame-metadata reconciliation. `"identical"` requires exact equality; `"merge"` recursively combines non-conflicting unaligned records.
`active_assay`	Optional active assay for the result. Required when operands have different active assays.

Value

A lazily row-bound `fmri_frame`.

Examples

```
sp <- volume_space(dim = c(2, 2, 2), affine = diag(4))
make <- function(prefix) {
  fmri_frame(
    assays = list(bold = matrix(rnorm(4 * n_features(sp)), nrow = 4)),
    observations = data.frame(.obs_id = sprintf("%s-%d", prefix, 1:4)),
    space = sp
  )
```

```
}
bound <- bind_observations(make("a"), make("b"))
dim(bound)
```

`block_apply` *Apply a function to bounded feature blocks*

Description

Apply a function to bounded feature blocks

Usage

```
block_apply(x, FUN, block_size = 4096L, assay = active_assay(x), ...)
```

Arguments

`x`	An `fmri_frame` or view.
`FUN`	Function receiving an observation-by-feature matrix and feature IDs.
`block_size`	Number of features per block.
`assay`	Assay name.
`...`	Additional arguments passed to `FUN`.

Value

A list of block results.

Examples

```
sp <- volume_space(dim = c(2, 2, 2), affine = diag(4))
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(4 * n_features(sp)), nrow = 4)),
  observations = data.frame(.obs_id = sprintf("vol-%d", 1:4)),
  space = sp
)
block_apply(frame, function(mat, ids) colMeans(mat), block_size = 4L)
```

block_manifest *Inspect a frame block plan*

Description

Inspect a frame block plan

Usage

```
block_manifest(plan)
```

Arguments

plan	A frame_block_plan.

Value

A data frame containing logical block bounds and byte estimates.

Examples

```
frame <- fmri_frame(
  assays = list(signal = memory_source(matrix(seq_len(20), 5, 4))),
  observations = data.frame(.obs_id = sprintf("obs-%d", 1:5)),
  active_assay = "signal"
)
plan <- plan_blocks(frame, memory_budget = 10 * 1024^2)
block_manifest(plan)
```

canonical-encoding *Encode and hash canonical R values*

Description

Canonicalization version 1 normalizes semantic R values and writes a tagged, length-prefixed package binary format. It is an explicitly R-only format. Named lists and attributes use lexicographic field order; unnamed list order is preserved. Strings are UTF-8 NFC, sparse matrix storage layouts are normalized, and NaN payloads use one R value while remaining distinct from NA_real_. Negative zero remains distinct from zero.

Usage

```
canonical_bytes(x)

canonical_sha256(x)
```

Arguments

x	A serializable R value.

Value

`canonical_bytes()` returns a raw vector. `canonical_sha256()` returns its lowercase SHA-256 hexadecimal digest.

Examples

```
canonical_bytes(list(b = 2, a = 1))
canonical_sha256(list(b = 2, a = 1))
```

---

canonicalization_contract

Identity and canonicalization contract

Description

Canonicalization version 1 hashes normalized R objects with SHA-256 over a package-owned tagged binary encoding. It is stable for supported package operations and round trips, but is deliberately R-only rather than a cross-language wire encoding.

Usage

```
canonicalization_contract()
```

Value

A serializable canonicalization descriptor.

Examples

```
contract <- canonicalization_contract()
contract$algorithm
```

`collect_assay` *Collect one frame assay under an explicit memory budget*

Description

Collect one frame assay under an explicit memory budget

Usage

```
collect_assay(
  x,
  assay = active_assay(x),
  memory_budget = getOption("fmridataset.collect_budget", 2 * 1024^3),
  force = FALSE
)
```

Arguments

`x`	An `fmri_frame` or view.
`assay`	One assay name; defaults to the active assay.
`memory_budget`	Maximum estimated peak bytes, including the retained output and source conversion or decompression buffers.
`force`	Allow collection above the estimated peak budget.

Value

A dense matrix.

Examples

```
sp <- volume_space(dim = c(2, 2, 2), affine = diag(4))
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(4 * n_features(sp)), nrow = 4)),
  observations = data.frame(.obs_id = sprintf("vol-%d", 1:4)),
  space = sp
)
collect_assay(frame)
```

`collect_spatial_maps` *Collect spatial maps through native or reconstructed reads*

Description

Collect spatial maps through native or reconstructed reads

Usage

```
collect_spatial_maps(
  x,
  observations = NULL,
  assay = active_assay(x),
  path = c("auto", "native", "reconstruct"),
  memory_budget = getOption("fmridataset.spatial_budget", 512 * 1024^2)
)
```

Arguments

`x`	An `fmri_frame` or view.
`observations`	Observation IDs or integer positions. The requested order is preserved. Duplicated selectors are rejected, as they are on every other frame axis selection.
`assay`	Assay name.
`path`	One of `"auto"`, `"native"`, or `"reconstruct"`.
`memory_budget`	Maximum estimated peak bytes for all returned native maps plus the current packed read, conversion, and reconstruction buffers.

Value

A named list with one native spatial object per observation.

Examples

```
sp <- volume_space(dim = c(2L, 2L, 1L), affine = diag(4), support = 1:4)
frame <- fmri_frame(
  assays = list(signal = memory_source(matrix(seq_len(12), 3, 4))),
  observations = data.frame(.obs_id = sprintf("obs-%d", 1:3)),
  space = sp,
  active_assay = "signal"
)
maps <- collect_spatial_maps(frame, observations = c(1L, 2L))
names(maps)
```

collection-accessors *Access frames in an fMRI collection*

Description

Access frames in an fMRI collection

Usage

```
collection_frames(x)

collection_frame(x, id)

collection_ids(x)
```

Arguments

x	An fmri_collection.
id	One stable frame ID.

Value

The named frame list, one frame, or the stable frame IDs.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
collection <- fmri_collection(list(sub01 = frame, sub02 = frame))
collection_ids(collection)
collection_frame(collection, "sub01")
```

collection-spaces *Summarize collection feature spaces*

Description

Summarize collection feature spaces

Usage

```
collection_space_data(x)

collection_common_space(x)
```

Arguments

x	An fmri_collection.

Value

collection_space_data() returns one metadata row per frame; collection_common_space() returns whether every feature space is exactly compatible with the first.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
collection <- fmri_collection(list(sub01 = frame, sub02 = frame))
collection_space_data(collection)
collection_common_space(collection)
```

---

collection_digest *Compute a deterministic collection digest*

Description

Compute a deterministic collection digest

Usage

```
collection_digest(x)
```

Arguments

x	An fmri_collection.

Value

A SHA-256 digest computed without reading numerical arrays.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
collection <- fmri_collection(list(sub01 = frame, sub02 = frame))
collection_digest(collection)
```

compose_frame_links *Compose source-to-target frame links*

Description

The target of `first` must be the source of `second`, and their addressed axes must agree. Explicit ID maps are joined through the intermediate IDs. Feature operators are composed as `second %*% first` without changing link direction.

Usage

```
compose_frame_links(first, second, type = NULL, metadata = list())
```

Arguments

`first`	A canonical source-to-intermediate `frame_link`.
`second`	A canonical intermediate-to-target `frame_link`.
`type`	Result link type. It may be omitted when both inputs have the same type.
`metadata`	Unaligned result-link metadata.

Value

A canonical source-to-target `frame_link`.

Examples

```
first <- frame_link("bold", "beta", type = "derivation")
second <- frame_link("beta", "summary", type = "derivation")
composed <- compose_frame_links(first, second)
composed$source
composed$target
```

composite-parts *Inspect composite feature-space parts*

Description

Inspect composite feature-space parts

Usage

```
composite_parts(x)

composite_part_names(x)

composite_part(x, name)
```

Arguments

x	A composite_space.
name	One child part name.

Value

composite_parts() returns the ordered named child spaces; composite_part_names() returns their names; and composite_part() returns one child space.

Examples

```
parts <- list(
  left = index_space(2, ids = c("l1", "l2")),
  right = index_space(2, ids = c("r1", "r2"))
)
x <- composite_space(parts)
composite_part_names(x)
composite_part(x, "left")
```

---

composite_space *Construct an ordered composite feature space*

Description

A composite_space forms one feature axis from heterogeneous child spaces, such as left cortex, right cortex, and subcortical volume. It owns only the ordered routing between that axis and its named parts; each child remains the authority for spatial identity, vectorization, and reconstruction.

Usage

```
composite_space(
  parts,
  composite_type = "composite",
  metadata = list(),
  route = NULL
)
```

Arguments

parts	A named list of non-empty feature_space objects.
composite_type	A stable semantic label, such as "grayordinate_like".
metadata	Additional serializable metadata.
route	Optional internal routing table with part and part_index columns. By default, all child features are concatenated in part order.

Value

A composite_space.

Examples

```
parts <- list(
  left = index_space(2, ids = c("l1", "l2")),
  right = index_space(2, ids = c("r1", "r2"))
)
x <- composite_space(parts, composite_type = "bilateral")
feature_ids(x)
```

---

content_hash *Compute a numerical content hash of an array source*

Description

content_hash() is the one operation in the package that identifies an array by its values. It is the content domain of the identity contract (see identity_descriptor()) and it is always explicit: nothing in the package computes it for you. Ordinary construction, inspection, fingerprinting, planning, and persistence never call it.

Usage

```
content_hash(x, ...)

## Default S3 method:
content_hash(x, ...)

## S3 method for class 'array_source'
content_hash(
  x,
  ...,
  block_bytes = getOption("fmridataset.target_block_bytes", 4 * 1024^2)
)

## S3 method for class 'memory_source'
content_hash(
  x,
  ...,
  block_bytes = getOption("fmridataset.target_block_bytes", 4 * 1024^2)
)

## S3 method for class 'source_view'
content_hash(
  x,
  ...,
```

```
  block_bytes = getOption("fmridataset.target_block_bytes", 4 * 1024^2)
)

## S3 method for class 'row_sharded_source'
content_hash(
  x,
  ...,
  block_bytes = getOption("fmridataset.target_block_bytes", 4 * 1024^2)
)
```

Arguments

x	An array source, a handle, or a matrix coercible to a memory source.
...	Passed to methods.
block_bytes	Upper bound on the realized bytes of one read. The default reuses the block-planning option fmridataset.target_block_bytes.

Details

The digest is O(n) in the number of values. The default method streams: it opens one handle, reads bounded row blocks through the source protocol aligned to the source chunk grid, and hashes each block into a running state, so no source is materialized whole. block_bytes bounds the realized size of one read and affects only I/O; every block size produces the same digest. Blocks cover whole rows when a row fits the bound and consecutive feature ranges of one row otherwise.

Two sources hash equal exactly when they have the same shape, realize to the same R mode (double, logical, or complex), and hold equal values. Storage dtype, chunking, wrappers, views, and shard composition do not enter the digest, so an in-memory copy and a row-bound composition of the same values agree. NaN payloads and NA are normalized to one representation each and remain distinct; negative zero stays distinct from zero.

The digest describes the values at the moment they were read. It is a receipt, not a live property: a later in-memory mutation of a memory source's payload is not detected by source_fingerprint() and shows only when a fresh content_hash() is compared with the earlier one.

Value

One lowercase SHA-256 hexadecimal string.

See Also

source_fingerprint() for the cheap revision identity of a descriptor, identity_descriptor() for typed identity domains, and inst/architecture/ADR-009-source-fingerprints-and-content-hashes.md for the policy.

Examples

```
a <- memory_source(matrix(1:6, 2, 3))
b <- memory_source(matrix(as.double(1:6), 2, 3), chunks = c(1, 1))
identical(source_fingerprint(a), source_fingerprint(b))
identical(content_hash(a), content_hash(b))
```

content_hash_contract *Content-hash contract*

Description

Content hashing version 1 is a chained SHA-256 over fixed-size leaves of the realized values in row-major order. The digest depends on the shape, the realized R mode, and every value, and on nothing else: not on the storage dtype, the chunk grid, the read block size, or the composition of sources that produced the values.

Usage

```
content_hash_contract()
```

Value

A serializable content-hash contract descriptor.

Examples

```
content_hash_contract()
```

counting_source *Developer tool: instrument an array source*

Description

`counting_source()` records numerical reads without placing a mutable environment inside the source descriptor. It is exported for backend and downstream conformance suites, not as an application data source.

Usage

```
counting_source(source)

source_counts(x)

reset_source_counts(x)
```

Arguments

source	An array source.
x	A counting source.

Details

This is developer-only test instrumentation. The counter registry is process-local, is not persisted with the descriptor, and must not be used as provenance or as an execution receipt.

Value

A serializable instrumented source.

Examples

```
src <- counting_source(memory_source(matrix(seq_len(6), nrow = 2)))
source_read(src, observations = 1)
source_counts(src)$reads
```

---

entity-accessors *Access entities from a frame or registry*

Description

Access entities from a frame or registry

Usage

```
entities(x, ...)

entity(x, name, ...)

entity_names(x)
```

Arguments

x	An fmri_frame, view, or entity_registry.
...	Additional method arguments.
name	One registered entity name.

Value

entities() returns the registry; entity() returns one entity_frame; entity_names() returns registry names.

Examples

```
subjects <- entity_frame(
  data = tibble::tibble(subject_id = c("sub-1", "sub-2")),
  key = "subject_id"
)
registry <- entity_registry(subject = subjects)
entities(registry)
entity(registry, "subject")
entity_names(registry)
```

`entity-frame-accessors` *Entity-frame accessors*

Description

Entity-frame accessors

Usage

```
entity_key(x)

entity_ids(x)

entity_data(x)

entity_blocks(x)
```

Arguments

`x`	An `entity_frame`.

Value

The stable key name, entity IDs, scalar data, or aligned blocks.

Examples

```
embedding <- axis_block(matrix(as.double(1:8), 2, 4), role = "embedding")
x <- entity_frame(
  data = tibble::tibble(stimulus_id = c("stim-1", "stim-2")),
  key = "stimulus_id",
  blocks = list(semantic = embedding)
)
entity_key(x)
entity_ids(x)
entity_data(x)
entity_blocks(x)
```

`entity_feature_validity` *Describe compressed entity-by-feature validity*

Description

Describe compressed entity-by-feature validity

Usage

```
entity_feature_validity(entity, entity_ids, masks, space, metadata = list())
```

Arguments

`entity`	Entity registry name.
`entity_ids`	Stable entity IDs aligned to mask rows.
`masks`	Logical entity-by-feature matrix or a `mask_bank` whose original row assignments have the same length as `entity_ids`.
`space`	Exact feature space addressed by validity columns.
`metadata`	Serializable relation metadata.

Value

An `entity_feature_validity` relation descriptor.

Examples

```
space <- index_space(4, ids = paste0("f", 1:4), namespace = "validity-ex")
validity <- entity_feature_validity(
  entity = "subject", entity_ids = c("sub-1", "sub-2"),
  masks = rbind(
    c(TRUE, TRUE, FALSE, TRUE),
    c(TRUE, FALSE, FALSE, TRUE)
  ),
  space = space
)
validity_entity(validity)
```

entity_frame *Construct a keyed entity frame*

Description

An entity frame stores one row per subject, session, run, stimulus, item, or other study entity. Scalar annotations live in `data`; multivariate values live in named, first-axis-aligned `axis_block` objects.

Usage

```
entity_frame(data, key, blocks = list(), entity_type = NULL, metadata = list())
```

Arguments

`data`	Scalar entity annotations with one row per entity.
`key`	Name of the stable primary-key column.
`blocks`	Named entity-aligned `axis_block` objects.
`entity_type`	Optional semantic type such as `"subject"` or `"stimulus"`.
`metadata`	Additional serializable metadata.

Value

An `entity_frame`, also implementing the `axis_frame` contract.

Examples

```
x <- entity_frame(
  data = tibble::tibble(
    stimulus_id = c("stim-1", "stim-2", "stim-3"),
    category = c("face", "scene", "object")
  ),
  key = "stimulus_id"
)
entity_ids(x)
```

entity_registry *Construct and validate an entity registry*

Description

Construct and validate an entity registry

Usage

```
entity_registry(entities = list(), ...)

validate_entity_registry(x)
```

Arguments

entities	A named list of entity_frame objects.
...	Alternatively, named entity_frame objects.
x	An entity registry.

Value

A named entity_registry.

Examples

```
subjects <- entity_frame(
  data = tibble::tibble(subject_id = c("sub-1", "sub-2")),
  key = "subject_id"
)
registry <- entity_registry(subject = subjects)
entity_names(registry)
```

---

entity_registry_digest

Compute a stable entity-registry digest

Description

Compute a stable entity-registry digest

Usage

```
entity_registry_digest(x)
```

Arguments

x	A frame, view, or entity registry.

Value

A hexadecimal digest over the normalized registry.

Examples

```
subjects <- entity_frame(
  data = tibble::tibble(subject_id = c("sub-1", "sub-2")),
  key = "subject_id"
)
registry <- entity_registry(subject = subjects)
entity_registry_digest(registry)
```

event-accessors *Event-table accessors*

Description

Event-table accessors

Usage

```
event_data(x)

event_key(x)
```

Arguments

x	An fmri_event_table.

Value

Event scalar data or the stable key name.

Examples

```
et <- event_table(
  data.frame(event_id = c("e1", "e2"), onset = c(0, 10)),
  key = "event_id"
)
event_data(et)
event_key(et)
```

event_table *Construct a keyed event table*

Description

Event rows retain their natural cardinality and are not expanded to acquired volumes. Entity-key columns are validated against a study when attached.

Usage

```
event_table(data, key = "event_id", metadata = list())
```

Arguments

data	Scalar event annotations.
key	Stable event-key column.
metadata	Serializable event-table metadata.

Value

An fmri_event_table.

Examples

```
et <- event_table(
  data.frame(event_id = c("e1", "e2"), onset = c(0, 10), duration = c(2, 2)),
  key = "event_id"
)
event_data(et)
event_key(et)
```

---

execute_block_plan *Execute a bounded frame block plan*

Description

Execute a bounded frame block plan

Usage

```
execute_block_plan(x, plan, FUN, ..., assay = plan$assay)
```

Arguments

x	The same frame or view used to construct plan.
plan	A frame_block_plan.
FUN	Function receiving values, observation_ids, feature_ids, and the one-row block manifest entry.
...	Additional arguments passed to FUN.
assay	Assay name; defaults to the planned assay.

Value

A list containing one result per planned block.

Examples

```
frame <- fmri_frame(
  assays = list(signal = memory_source(matrix(seq_len(20), 5, 4))),
  observations = data.frame(.obs_id = sprintf("obs-%d", 1:5)),
  active_assay = "signal"
)
plan <- plan_blocks(frame, memory_budget = 10 * 1024^2)
execute_block_plan(frame, plan, function(values, observation_ids, feature_ids, block) {
  sum(values)
})
```

`execute_spatial` *Stream an operation over spatial maps*

Description

Unlike `collect_spatial_maps()`, `execute_spatial()` holds only one input map at a time. The callback result is retained, so callers remain responsible for keeping returned values appropriately small.

Usage

```
execute_spatial(
  x,
  observations = NULL,
  FUN,
  ...,
  assay = active_assay(x),
  path = c("auto", "native", "reconstruct"),
  memory_budget = getOption("fmridataset.spatial_budget", 512 * 1024^2)
)
```

Arguments

`x`	An `fmri_frame` or view.
`observations`	Observation IDs or integer positions.
`FUN`	Function receiving `map` and `observation_id`.
`...`	Additional arguments passed to `FUN`.
`assay`	Assay name.
`path`	One of `"auto"`, `"native"`, or `"reconstruct"`.
`memory_budget`	Maximum estimated peak bytes for one input spatial map plus its packed read, conversion, and reconstruction buffers.

Value

A list of callback results in requested observation order.

Examples

```
sp <- volume_space(dim = c(2L, 2L, 1L), affine = diag(4), support = 1:4)
frame <- fmri_frame(
  assays = list(signal = memory_source(matrix(seq_len(12), 3, 4))),
  observations = data.frame(.obs_id = sprintf("obs-%d", 1:3)),
  space = sp,
  active_assay = "signal"
)
execute_spatial(frame, c(1L, 2L), function(map, observation_id) observation_id)
```

`execution_path` *Select a matrix or spatial execution path*

Description

Matrix operations always use bounded observation-by-feature blocks. Spatial operations use a source's native-image capability only when the frame view retains the complete feature domain; otherwise they reconstruct maps from packed assay values through the frame's feature space.

Usage

```
execution_path(
  x,
  operation = c("matrix", "spatial"),
  assay = active_assay(x),
  path = c("auto", "native", "reconstruct")
)
```

Arguments

`x`	An `fmri_frame` or view.
`operation`	Either `"matrix"` or `"spatial"`.
`assay`	Assay name.
`path`	For spatial operations, one of `"auto"`, `"native"`, or `"reconstruct"`.

Value

One of `"matrix"`, `"native"`, or `"reconstruct"`.

Examples

```
sp <- volume_space(dim = c(2L, 2L, 1L), affine = diag(4), support = 1:4)
frame <- fmri_frame(
  assays = list(signal = memory_source(matrix(seq_len(12), 3, 4))),
  observations = data.frame(.obs_id = sprintf("obs-%d", 1:3)),
  space = sp,
  active_assay = "signal"
)
execution_path(frame, operation = "matrix")
execution_path(frame, operation = "spatial")
```

`explain` *Explain a frame without reading numerical values*

Description

`explain()` returns a bounded, serializable summary of the visible frame. Axis IDs are sampled by default so inspecting a large study does not create another large object. Set `ids = "complete"` only when every visible ID is required.

Usage

```
explain(x, ids = c("sample", "none", "complete"), sample_size = 3L)
```

Arguments

`x`	An `fmri_frame` or view.
`ids`	One of `"sample"`, `"none"`, or `"complete"`.
`sample_size`	Number of IDs sampled from each end of each axis.

Details

The schema digest describes column, block, assay, relation, entity, table, and space contracts. The semantic digest covers the complete source-free FDS manifest. Physical source fingerprints are reported separately.

Value

A bounded serializable execution summary. No assay or aligned-block values are read. `ids_durable` reports whether the observation axis and the feature space carry durable IDs; when it is `FALSE` the semantic digest is `NULL` because ephemeral IDs cannot enter an FDS manifest.

Examples

```
sp <- volume_space(dim = c(2, 2, 2), affine = diag(4))
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(4 * n_features(sp)), nrow = 4)),
  observations = data.frame(.obs_id = sprintf("vol-%d", 1:4)),
  space = sp
)
summary <- explain(frame)
summary$shape
```

fault_source *Developer tool: inject deterministic source failures*

Description

This source is exported solely for deterministic backend, codec, cleanup, and recovery conformance tests.

Usage

```
fault_source(
  source,
  stage = c("read", "open", "native_read", "close"),
  message = NULL
)
```

Arguments

source	An array source.
stage	One of "open", "read", "native_read", or "close".
message	Failure message.

Details

This is developer-only test instrumentation. Never persist a fault_source as study data or use one in an analysis plan.

Value

A serializable fault-injecting source.

Examples

```
src <- fault_source(memory_source(matrix(seq_len(6), nrow = 2)), stage = "read")
tryCatch(source_read(src), error = function(e) conditionMessage(e))
```

fds_frame_bindings *Extract physical array bindings from a frame*

Description

Storage codecs use this helper to pair each source-free FDS array declaration with its current physical or in-memory array_source.

Usage

```
fds_frame_bindings(x)
```

Arguments

x	An fmri_frame.

Value

A named list of array_source descriptors keyed exactly like the manifest arrays registry.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
bindings <- fds_frame_bindings(frame)
names(bindings)
```

---

fds_frame_manifest *Construct and validate an FDS v1 frame manifest*

Description

The manifest owns semantic alignment but deliberately excludes physical source descriptors. Storage packages bind assay names to physical array locations separately and reconstruct frames with frame_from_fds_manifest().

Usage

```
fds_frame_manifest(x)

validate_fds_manifest(manifest)
```

Arguments

x	An fmri_frame.
manifest	An FDS manifest.

Value

A serializable backend-neutral manifest.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
manifest <- fds_frame_manifest(frame)
manifest$shape
validate_fds_manifest(manifest)
```

`fds_manifest_digest` *Compute a canonical FDS manifest digest*

Description

Compute a canonical FDS manifest digest

Usage

```
fds_manifest_digest(manifest)
```

Arguments

`manifest`	A valid FDS manifest.

Value

A stable hexadecimal digest over semantic manifest content.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
manifest <- fds_frame_manifest(frame)
fds_manifest_digest(manifest)
```

fds_schema *FDS logical schema identity*

Description

FDS version 1 is the backend-neutral semantic contract for persisted `fmri_frame` objects. Physical codecs may add locations, chunks, compression, and checksums outside this manifest, but cannot change its field meanings.

Usage

```
fds_schema()

fds_schema_version()
```

Value

`fds_schema()` returns the immutable schema identity; `fds_schema_version()` returns its integer major version.

Examples

```
fds_schema()
fds_schema_version()
```

fds_study_bindings *Extract shared study-level physical bindings*

Description

Extract shared study-level physical bindings

Usage

```
fds_study_bindings(x)
```

Arguments

`x`	An `fmri_study`.

Value

A named list of shared entity-block payloads. Representation arrays remain owned by their individual frame bindings.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
study <- fmri_study(list(main = frame))
fds_study_bindings(study)
```

`fds_study_manifest` *Construct and validate an FDS v2 study manifest*

Description

Study manifests retain shared entities, typed links, relational tables, and the semantic manifests of every frame or collection member. Numerical sources remain separate bindings so physical storage packages do not own or reinterpret study semantics.

Usage

```
fds_study_manifest(x)

validate_fds_study_manifest(manifest)
```

Arguments

`x`	An `fmri_study`.
`manifest`	An FDS study manifest.

Value

`fds_study_manifest()` returns a serializable source-free manifest; `validate_fds_study_manifest()` returns `manifest` invisibly.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
study <- fmri_study(list(main = frame))
manifest <- fds_study_manifest(study)
names(manifest$representations)
```

fds_study_manifest_digest

Compute a canonical FDS study-manifest digest

Description

Compute a canonical FDS study-manifest digest

Usage

```
fds_study_manifest_digest(manifest)
```

Arguments

manifest	A valid FDS study manifest.

Value

A stable hexadecimal digest over source-free study semantics.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
study <- fmri_study(list(main = frame))
manifest <- fds_study_manifest(study)
fds_study_manifest_digest(manifest)
```

fds_study_representations

Extract canonical study representations for persistence

Description

Extract canonical study representations for persistence

Usage

```
fds_study_representations(x)
```

Arguments

x	An fmri_study.

Value

Named frames and collections matching `fds_study_manifest(x)`.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
study <- fmri_study(list(main = frame))
names(fds_study_representations(study))
```

---

feature-map-accessors *Validate and inspect feature maps*

Description

Validate and inspect feature maps

Usage

```
validate_feature_map(x)

feature_map_source_space(x)

feature_map_target_space(x)

feature_map_operator(x)

feature_map_digest(x)
```

Arguments

x	A feature_map.

Value

`validate_feature_map()` returns `x` invisibly. The accessors return the source space, target space, linear operator, or deterministic digest.

Examples

```
src <- index_space(4, ids = paste0("v", 1:4), namespace = "map-source")
tgt <- index_space(2, ids = paste0("p", 1:2), namespace = "map-target")
op <- matrix(c(0.5, 0.5, 0, 0, 0, 0, 0.5, 0.5), nrow = 2, byrow = TRUE)
m <- feature_map(src, tgt, op, map_type = "toy_aggregation")
feature_map_source_space(m)
feature_map_target_space(m)
feature_map_digest(m)
```

---

feature-space *Feature-space contract*

Description

Every feature-space class (such as volume_space(), surface_space(), parcel_space(), basis_space(), composite_space(), and index_space()) implements this shared generic contract for feature count, stable identity, restriction, vectorization, and reconstruction.

Usage

```
n_features(x, ...)

feature_ids(x, ...)

native_shape(x, ...)

feature_data(x, ...)

space_digest(x, ...)

restrict_space(x, index, ...)

vectorize_space(x, spatial_object, ...)

reconstruct_space(x, vector, ...)

adjacency(x, ...)

same_space(x, y, ...)

assert_same_space(x, y, ...)

## S3 method for class 'surface_space'
reconstruct_space(x, vector, format = c("surface_map", "neurosurf"), ...)
```

Arguments

x	A feature-space object.
...	Additional arguments for methods.
index	Features to keep, in order: positive or negative positions, a logical mask over the features, or character feature IDs. Every built-in space resolves these by the same selection law and rejects out-of-bounds, duplicated, or unknown selectors with structured errors.
spatial_object	A native spatial object to vectorize. For a volume_space, a NeuroVol must match the space's dimensions and affine and an array must match its dimensions; a dimensionless vector of length prod(dim) is read as the full volume in column-major voxel order.
vector	A feature vector to reconstruct. An unnamed vector is positional; a named vector must name every feature ID exactly once and is reordered to the space's feature order.
y	A second feature-space object.
format	Surface reconstruction format. The backend-neutral default is "surface_map"; "neurosurf" returns a neurosurf::NeuroSurface when embedded unilateral geometry is available.

Value

n_features() returns a single integer; feature_ids() returns a character vector of stable feature identifiers; native_shape() returns the class-specific native dimensions; feature_data() returns a tibble of per-feature metadata; space_digest() returns a content digest string; restrict_space() and reconstruct_space() return an object of the same feature-space class; vectorize_space() returns a numeric vector; adjacency() returns a sparse adjacency matrix or NULL; same_space() returns a space_comparison list whose same field is TRUE only when class, digest, and ordered feature IDs all agree; assert_same_space() returns that report invisibly and signals an fmridataset_error_space_mismatch error otherwise.

Examples

```
x <- index_space(3, ids = c("a", "b", "c"))
n_features(x)
feature_ids(x)
y <- restrict_space(x, 1:2)
same_space(x, x)$same
```

---

feature-validity-accessors

Validate and inspect entity-feature validity

Description

Validate and inspect entity-feature validity

Usage

```
validate_entity_feature_validity(x)

validity_entity(x, name = NULL)

validity_entity_ids(x, name = NULL)

validity_mask_bank(x, name = NULL)

validity_space(x, name = NULL)

validity_matrix(x, name = NULL)
```

Arguments

x	An entity_feature_validity, frame, or view.
name	Relation name when x is a frame or view.

Value

The validated descriptor, entity name/IDs, mask bank, feature space, or expanded entity-by-feature logical matrix.

Examples

```
space <- index_space(4, ids = paste0("f", 1:4), namespace = "validity-ex")
validity <- entity_feature_validity(
  entity = "subject", entity_ids = c("sub-1", "sub-2"),
  masks = rbind(
    c(TRUE, TRUE, FALSE, TRUE),
    c(TRUE, FALSE, FALSE, TRUE)
  ),
  space = space
)
validity_entity_ids(validity)
validity_matrix(validity)
```

---

feature_axis *Construct a spatial feature axis*

Description

Construct a spatial feature axis

Usage

```
feature_axis(data, space = NULL, blocks = list(), metadata = list(), ...)
```

Arguments

data	Feature metadata or an fmri_frame when used as an accessor.
space	A FeatureSpace.
blocks	Feature-aligned blocks.
metadata	Additional metadata.
...	Additional arguments for methods.

Value

A feature axis_frame carrying its space.

Examples

```
sp <- index_space(3, ids = sprintf("f%d", 1:3), namespace = "ex")
fx <- feature_axis(data.frame(.feature_id = sprintf("f%d", 1:3)), space = sp)
axis_ids(fx)
```

---

feature_map *Describe an explicit transformation between feature spaces*

Description

A feature map owns a target-by-source linear operator and the complete spatial identity of both axes. Equal dimensions are never treated as evidence of spatial compatibility. Statistical execution plans and covariance models remain the responsibility of packages such as fmrigds.

Usage

```
feature_map(
  from,
  to,
  operator,
  map_type = "linear",
  traits = list(linear = TRUE),
  provenance = list(),
  metadata = list()
)
```

Arguments

from	Source feature_space.
to	Target feature_space.
operator	Target-by-source matrix, sparse Matrix, or serializable two-dimensional array_source.
map_type	Stable map-family label.
traits	Named serializable semantic traits.
provenance	Serializable derivation metadata for the map itself.
metadata	Additional serializable metadata.

Value

A serializable `feature_map` descriptor.

Examples

```
src <- index_space(4, ids = paste0("v", 1:4), namespace = "map-source")
tgt <- index_space(2, ids = paste0("p", 1:2), namespace = "map-target")
op <- matrix(c(0.5, 0.5, 0, 0, 0, 0, 0.5, 0.5), nrow = 2, byrow = TRUE)
m <- feature_map(src, tgt, op, map_type = "toy_aggregation")
feature_map_operator(m)
```

`feature_map_from_target` *Derive the canonical map owned by a parent-linked target space*

Description

Parcel spaces contribute their aggregation operator and basis spaces their analysis operator. Other transformations require an explicit `feature_map()`.

Usage

```
feature_map_from_target(target)
```

Arguments

`target`	A parent-linked `parcel_space` or `basis_space`.

Value

A `feature_map` from `parent_space(target)` to `target`.

Examples

```
parent <- volume_space(c(2, 2, 1), support = 1:4, template = "toy")
membership <- Matrix::sparseMatrix(
  i = 1:4, j = c(1L, 1L, 2L, 2L), x = 1, dims = c(4L, 2L)
)
target <- parcel_space(parent, c("left", "right"), membership, atlas = "toy")
m <- feature_map_from_target(target)
feature_map_target_space(m)
```

feature_mapped_source *Construct a lazy source transformed through a feature map*

Description

Construct a lazy source transformed through a feature map

Usage

```
feature_mapped_source(source, map, rule = c("linear", "independent_variance"))
```

Arguments

source	Observation-by-source-feature array_source.
map	A compatible feature_map.
rule	Transformation rule. "linear" maps ordinary values; "independent_variance" maps diagonal variances with squared weights.

Value

A serializable feature_mapped_source.

Examples

```
src <- index_space(4, ids = paste0("v", 1:4), namespace = "map-source")
tgt <- index_space(2, ids = paste0("p", 1:2), namespace = "map-target")
op <- matrix(c(0.5, 0.5, 0, 0, 0, 0, 0.5, 0.5), nrow = 2, byrow = TRUE)
m <- feature_map(src, tgt, op, map_type = "toy_aggregation")
fs <- feature_mapped_source(memory_source(matrix(1:12, nrow = 3)), m)
dim(source_read(fs))
```

filter_entities *Filter every study representation through one shared entity selection*

Description

Dropping entity records cascades: child entities whose key relation references a dropped record are dropped too, observations reaching a dropped record are removed, and sparse edges, validity rows, link maps, and table rows are restricted to the surviving IDs. Table rows without a key value are kept.

Usage

```
filter_entities(x, entity, predicate)
```

Arguments

x	An fmri_study.
entity	Bare or quoted shared entity name.
predicate	A scalar-metadata predicate evaluated on that entity table.

Value

A self-contained fmri_study whose numerical sources remain lazy.

Examples

```
subject_entity <- entity_frame(
  data.frame(subject_id = c("s1", "s2"), age = c(20, 60)),
  key = "subject_id"
)
obs <- data.frame(
  .obs_id = paste0("o", 1:4),
  subject_id = c("s1", "s1", "s2", "s2")
)
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(16), nrow = 4)),
  observations = obs,
  space = index_space(4, ids = paste0("f", 1:4)),
  entities = list(subject = subject_entity),
  relations = list(
    observation_subject = key_relation("subject_id", target = "subject")
  )
)
study <- fmri_study(list(main = frame), entities = list(subject = subject_entity))
older <- filter_entities(study, subject, age > 30)
observation_ids(study_frame(older, "main"))
```

---

filter_obs *Filter frame observations using scalar metadata*

Description

Filter frame observations using scalar metadata

Usage

```
filter_obs(x, predicate, resolve = TRUE)
```

Arguments

x	An fmri_frame or view.
predicate	A metadata expression returning one logical value per observation.
resolve	Whether the predicate may use namespaced entity metadata.

Value

An fmri_view.

Examples

```
sp <- volume_space(dim = c(2, 2, 2), affine = diag(4))
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(4 * n_features(sp)), nrow = 4)),
  observations = data.frame(
    .obs_id = sprintf("vol-%d", 1:4),
    run_id = rep(c("run-1", "run-2"), each = 2)
  ),
  space = sp
)
filter_obs(frame, run_id == "run-1")
```

---

fmri_collection *Construct a collection of semantically equivalent fMRI frames*

Description

A collection keeps frames separate when they share an observational and assay contract but cannot share a feature axis, as with participant-native volume or surface spaces. Equal feature dimensions or IDs are not required; feature-space type and annotation semantics are validated explicitly.

Usage

```
fmri_collection(frames, metadata = list(), provenance = NULL)
```

Arguments

frames	A non-empty named list of fmri_frame objects or lazy views.
metadata	Unaligned collection-level metadata.
provenance	NULL or a validated provenance_graph.

Value

An fmri_collection.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
collection <- fmri_collection(list(sub01 = frame, sub02 = frame))
collection_ids(collection)
```

`fmri_frame` *Construct a spatially typed annotated matrix*

Description

Construct a spatially typed annotated matrix

Usage

```
fmri_frame(
  assays,
  observations,
  features = NULL,
  space = NULL,
  entities = list(),
  relations = list(),
  tables = list(),
  active_assay = NULL,
  metadata = list(),
  provenance = NULL
)
```

Arguments

`assays`	Named matrices or serializable array sources.
`observations`	Observation metadata or an observation `axis_frame`.
`features`	Feature metadata or a spatial feature axis.
`space`	Feature space used when `features` is not already spatial.
`entities`	A named `entity_registry` or entries normalizable by `entity_registry()`.
`relations`	Named relation registry.
`tables`	Named typed tables created by `event_table()` or `auxiliary_table()`.
`active_assay`	Active assay name.
`metadata`	Unaligned frame-level record. Aligned values belong on an axis, entity, block, assay, relation, typed table, or linked frame.
`provenance`	`NULL` or a validated `provenance_graph`.

Value

An `fmri_frame`.

Examples

```
sp <- volume_space(dim = c(2, 2, 2), affine = diag(4))
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(4 * n_features(sp)), nrow = 4)),
  observations = data.frame(.obs_id = sprintf("vol-%d", 1:4)),
  space = sp
)
frame
```

---

fmri_study *Construct a linked fMRI study*

Description

Construct a linked fMRI study

Usage

```
fmri_study(
  frames,
  entities = list(),
  links = list(),
  tables = list(),
  metadata = list(),
  provenance = NULL
)
```

Arguments

frames	Named fmri_frame or fmri_collection representations.
entities	Shared authoritative entity registry.
links	Named frame_link descriptors.
tables	Named typed relational tables.
metadata	Unaligned study-level metadata.
provenance	NULL or a validated provenance_graph.

Value

An fmri_study.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
study <- fmri_study(list(main = frame))
study_ids(study)
```

`frame-accessors` *Frame accessors*

Description

Generic accessors for the components of an `fmri_frame` or `fmri_view`: its assays, active assay, observation and feature axes and IDs, block registries, and dimensions.

Usage

```
assays(x, ...)

assay(x, name = active_assay(x), ...)

active_assay(x, ...)

observation_axis(x, ...)

observations(x, resolve = FALSE, ...)

features(x, ...)

observation_ids(x, ...)

obs_blocks(x, resolve = FALSE, ...)

feature_blocks(x, ...)

## S3 method for class 'fmri_frame'
dim(x)

## S3 method for class 'fmri_view'
dim(x)
```

Arguments

`x`	An `fmri_frame` or `fmri_view`.

...	Additional method arguments.
name	One assay name; defaults to the active assay. Positions, vectors, and missing values are refused.
resolve	Whether to append reachable, namespaced entity annotations or lazily lifted entity blocks.

Value

assays() returns a named aligned_assay_set list and assay() one aligned_assay from it. active_assay() returns a single assay name. observation_axis() returns an axis_frame; observations() and features() return a data frame of the corresponding metadata. observation_ids() and feature_ids() return character vectors of stable IDs. obs_blocks() and feature_blocks() return named lists of axis_blocks. dim() returns a length-2 integer vector, and nrow() and ncol() return single integers.

Examples

```
sp <- volume_space(dim = c(2, 2, 2), affine = diag(4))
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(4 * n_features(sp)), nrow = 4)),
  observations = data.frame(.obs_id = sprintf("vol-%d", 1:4)),
  space = sp
)
names(assays(frame))
active_assay(frame)
dim(frame)
```

---

frame-schema-validation

Validate and compare canonical frame schemas

Description

Validate and compare canonical frame schemas

Usage

```
validate_frame_schema(schema)

compare_frame_schema(x, reference, mode = c("same", "collection", "bind"))

validate_against_schema(x, reference, mode = c("same", "collection", "bind"))

frame_schema_digest(x)
```

Arguments

schema	A canonical frame schema.
x	A frame, view, or schema to validate.
reference	A frame, view, or schema used as the reference contract.
mode	Comparison mode. same compares the complete schema; collection permits different observation counts and feature identities; bind permits different observation counts but requires one feature identity.

Details

compare_frame_schema() treats named records, such as column descriptors and the assay, entity, relation, and table registries, as unordered: column order and assay registry order are not semantic. This matches frame_schema_digest(), whose canonical encoding sorts record fields by name. Unnamed sequences such as feature and component IDs remain order-sensitive. validate_frame_schema() checks the shape of every section and rejects missing or unknown top-level fields.

Value

Validators return their input invisibly. compare_frame_schema() returns a structured frame_schema_compatibility report. frame_schema_digest() returns a single hex digest string.

Examples

```
sp <- volume_space(dim = c(2, 2, 2), affine = diag(4))
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(4 * n_features(sp)), nrow = 4)),
  observations = data.frame(.obs_id = sprintf("vol-%d", 1:4)),
  space = sp
)
schema <- frame_schema(frame)
validate_frame_schema(schema)
compare_frame_schema(frame, frame)$compatible
frame_schema_digest(frame)
```

---

frame_from_fds_manifest

Reconstruct a frame from an FDS manifest and physical sources

---

Description

Reconstruct a frame from an FDS manifest and physical sources

Usage

```
frame_from_fds_manifest(manifest, bindings)
```

Arguments

manifest	A valid FDS v1 frame manifest.
bindings	Named physical array payloads or array_source descriptors, one per manifest array declaration.

Value

An fmri_frame whose semantic state comes from manifest and whose lazy arrays come from bindings.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
manifest <- fds_frame_manifest(frame)
bindings <- fds_frame_bindings(frame)
rebuilt <- frame_from_fds_manifest(manifest, bindings)
identical(feature_ids(rebuilt), feature_ids(frame))
```

---

frame_link *Describe a typed link between study representations*

Description

Describe a typed link between study representations

Usage

```
frame_link(
  source,
  target,
  type = c("derivation", "mapping", "correspondence", "alignment"),
  map = NULL,
  source_axis = c("observation", "feature"),
  target_axis = c("observation", "feature"),
  metadata = list(),
  operator = NULL
)
```

Arguments

source	Source representation name.
target	Target representation name.
type	Link type: derivation, feature mapping, correspondence, or alignment.
map	Optional scalar table with .source_id and .target_id columns.
source_axis	Axis addressed by .source_id.
target_axis	Axis addressed by .target_id.
metadata	Serializable link metadata.
operator	Optional typed feature operator. This is valid only for a feature-to-feature mapping or alignment and remains a first-class field.

Value

A frame_link descriptor.

Examples

```
link <- frame_link("bold", "surface", type = "derivation")
link$source
link$target
```

---

frame_schema *Compute the canonical metadata-only frame schema*

Description

The schema describes aligned assay annotations, scalar column types and factor levels, multivariate block trailing axes and components, spatial identity, entities, relations, auxiliary tables, and active-assay policy. It inspects descriptors and metadata only and never reads numerical values.

Usage

```
frame_schema(x)
```

Arguments

x	An fmri_frame, synchronized view, or canonical frame schema.

Value

A serializable fmri_frame_schema.

Examples

```
sp <- volume_space(dim = c(2, 2, 2), affine = diag(4))
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(4 * n_features(sp)), nrow = 4)),
  observations = data.frame(.obs_id = sprintf("vol-%d", 1:4)),
  space = sp
)
schema <- frame_schema(frame)
schema$active_assay
```

---

hierarchy-accessors *Access derived hierarchy index data*

Description

Access derived hierarchy index data

Usage

```
hierarchy_ids(x)

hierarchy_groups(x)

hierarchy_levels(x)

hierarchy_relations(x)

hierarchy_complete(x)
```

Arguments

x	An fmri_hierarchy_index.

Value

hierarchy_ids() returns stable entity IDs; hierarchy_groups() returns stable integer grouping codes; hierarchy_levels() and hierarchy_relations() return named character vectors; hierarchy_complete() returns a logical vector.

Examples

```
subject <- entity_frame(
  data = tibble::tibble(subject_id = c("sub-1", "sub-2")),
  key = "subject_id"
)
observations <- tibble::tibble(
  .obs_id = paste0("obs-", 1:4),
  subject_id = c("sub-1", "sub-1", "sub-2", "sub-2")
```

```
)
frame <- fmri_frame(
  assays = list(beta = memory_source(matrix(as.double(1:12), 4, 3))),
  observations = observations,
  entities = list(subject = subject),
  relations = list(
    observation_subject = key_relation("subject_id", target = "subject")
  )
)
idx <- hierarchy_index(frame, levels = "subject")
hierarchy_ids(idx)
hierarchy_groups(idx)
hierarchy_levels(idx)
hierarchy_relations(idx)
hierarchy_complete(idx)
```

`hierarchy_digest` *Compute the deterministic digest of a hierarchy index*

Description

Compute the deterministic digest of a hierarchy index

Usage

```
hierarchy_digest(x)
```

Arguments

`x`	An `fmri_hierarchy_index`.

Value

A SHA-256 digest.

Examples

```
subject <- entity_frame(
  data = tibble::tibble(subject_id = c("sub-1", "sub-2")),
  key = "subject_id"
)
observations <- tibble::tibble(
  .obs_id = paste0("obs-", 1:4),
  subject_id = c("sub-1", "sub-1", "sub-2", "sub-2")
)
frame <- fmri_frame(
  assays = list(beta = memory_source(matrix(as.double(1:12), 4, 3))),
  observations = observations,
  entities = list(subject = subject),
  relations = list(
```

```
    observation_subject = key_relation("subject_id", target = "subject")
  )
)
idx <- hierarchy_index(frame, levels = "subject")
hierarchy_digest(idx)
```

`hierarchy_index` *Derive stable observation hierarchy indices*

Description

A hierarchy index is an immutable, assay-free cache derived from validated key relations. `levels` are supplied root-to-leaf. Each adjacent edge must form a strict chain from observations to the deepest entity and then through its parents. Crossed relations are therefore never mistaken for containment.

Usage

```
hierarchy_index(x, levels, relations = NULL)
```

Arguments

`x`	An `fmri_frame` or `fmri_view`.
`levels`	Unique entity names in root-to-leaf order.
`relations`	Optional named character vector mapping every level to the key-relation name used for its incoming edge. This is required when an edge is ambiguous.

Details

Integer group codes use entity-registry order, so they remain stable when a frame is filtered or reordered.

Value

An `fmri_hierarchy_index`.

Examples

```
subject <- entity_frame(
  data = tibble::tibble(subject_id = c("sub-1", "sub-2")),
  key = "subject_id"
)
observations <- tibble::tibble(
  .obs_id = paste0("obs-", 1:4),
  subject_id = c("sub-1", "sub-1", "sub-2", "sub-2")
)
frame <- fmri_frame(
  assays = list(beta = memory_source(matrix(as.double(1:12), 4, 3))),
```

```
  observations = observations,
  entities = list(subject = subject),
  relations = list(
    observation_subject = key_relation("subject_id", target = "subject")
  )
)
idx <- hierarchy_index(frame, levels = "subject")
hierarchy_ids(idx)
```

`id-policy` *Inspect axis ID durability*

Description

Inspect axis ID durability

Usage

```
axis_id_policy(x)

ids_are_durable(x)
```

Arguments

`x`	An axis, feature space, frame, or view.

Value

`axis_id_policy()` returns the versioned ID-policy descriptor; `ids_are_durable()` returns one logical value.

Examples

```
x <- axis_frame(data.frame(value = 1:3), id_policy = "ephemeral")
axis_id_policy(x)
ids_are_durable(x)
```

`identity_descriptor` *Inspect typed identity domains*

Description

Semantic and schema identities are independent of physical locations. Source fingerprints identify descriptors and selectors, not array contents. Content digests are optional receipts computed explicitly by `content_hash()` or supplied by a backend; this function never reads data to infer one, so the content domain always requires `content_digest`.

Usage

```
identity_descriptor(
  x,
 domain = c("auto", "semantic", "schema", "space", "source", "provenance", "content"),
  content_digest = NULL
)
```

Arguments

x	A frame, frame schema, feature space, array source, FDS manifest, provenance graph, collection, or study.
domain	Identity domain. Usually inferred from x.
content_digest	Externally computed content digest, such as the value of content_hash(). Required for domain = "content" and rejected with an fmridataset_error_identity error for any other domain.

Value

A serializable typed identity descriptor.

Examples

```
src <- memory_source(matrix(seq_len(6), nrow = 2))
identity_descriptor(src)$domain
```

---

index_space *Construct a generic indexed feature space*

Description

Construct a generic indexed feature space

Usage

```
index_space(
  n,
  ids = NULL,
  namespace = NULL,
  data = NULL,
  id_policy = c("require", "deterministic", "ephemeral")
)
```

Arguments

n	Number of features.
ids	Optional stable feature IDs.
namespace	Stable namespace used for deterministic IDs.
data	Optional feature metadata.
id_policy	ID policy used when ids is absent. The default requires explicit IDs; deterministic IDs additionally require namespace.

Value

An index_space.

Examples

```
index_space(3, ids = c("a", "b", "c"))
index_space(2, namespace = "roi", id_policy = "deterministic")
```

---

key_relation *Describe a symbolic foreign-key relation*

Description

A key relation does not duplicate a mapping. It declares that one scalar column on an observation, feature, or entity table references the stable IDs of one entity frame.

Usage

```
key_relation(
  key,
  target = NULL,
  source = "observation",
  allow_missing = FALSE,
  metadata = list()
)
```

Arguments

key	Foreign-key column on the source domain.
target	Target entity name. When NULL, frame validation infers the unique entity whose primary key has the same name as key.
source	Source domain: "observation", "feature", an entity name, or an explicit "entity:<name>" domain.
allow_missing	Whether missing foreign-key values are permitted.
metadata	Additional serializable metadata.

Value

A serializable `key_relation` descriptor.

Examples

```
rel <- key_relation("stimulus_id", target = "stimulus", source = "observation")
rel$key
```

`locate_source_rows` *Resolve logical observation rows to shards*

Description

Resolve logical observation rows to shards

Usage

```
locate_source_rows(x, observations = NULL)
```

Arguments

`x`	A `row_sharded_source`.
`observations`	Logical observation positions in requested order.

Value

A data frame mapping each request position to a shard and local row.

Examples

```
shards <- list(memory_source(matrix(1:4, nrow = 2)), memory_source(matrix(5:8, nrow = 2)))
locate_source_rows(row_sharded_source(shards), observations = c(1, 3))
```

`map_features` *Lazily transform a frame into a new feature domain*

Description

Lazily transform a frame into a new feature domain

Usage

```
map_features(x, target = NULL, map = NULL, assay_rules = "linear")
```

Arguments

x	An fmri_frame or view.
target	Optional parent-linked target space from which a canonical map can be derived.
map	Optional explicit feature_map.
assay_rules	Named rules for every assay: "linear" or "independent_variance". Unnamed scalar rules are recycled.

Value

A new linked-domain fmri_frame whose assays remain lazy.

Examples

```
parent <- volume_space(c(2, 2, 1), support = 1:4, template = "toy")
frame <- fmri_frame(
  assays = list(signal = matrix(1:12, nrow = 3)),
  observations = data.frame(.obs_id = paste0("o", 1:3)),
  space = parent
)
parcels <- parcel_space(
  parent,
  parcel_ids = c("left", "right"),
  membership = Matrix::sparseMatrix(
    i = 1:4, j = c(1L, 1L, 2L, 2L), x = 1, dims = c(4L, 2L)
  ),
  atlas = "toy-atlas"
)
parcel_frame <- map_features(frame, target = parcels)
dim(collect_assay(parcel_frame))
```

---

mask-bank-accessors *Validate and inspect a mask bank*

Description

Validate and inspect a mask bank

Usage

```
validate_mask_bank(x)

n_masks(x)

mask_values(x, mask = NULL)

mask_bank_digest(x)
```

Arguments

x	A mask_bank or validity descriptor.
mask	Optional mask ID or integer position.

Value

The validated bank, number of masks, unpacked logical masks, or deterministic digest.

Examples

```
space <- index_space(6, ids = paste0("f", 1:6), namespace = "validity-ex")
masks <- rbind(
  c(TRUE, TRUE, FALSE, TRUE, FALSE, TRUE),
  c(TRUE, FALSE, FALSE, TRUE, TRUE, TRUE)
)
bank <- mask_bank(masks, space)
mask_values(bank)
mask_bank_digest(bank)
```

---

mask_bank *Construct a deduplicated, bit-packed bank of feature masks*

Description

Construct a deduplicated, bit-packed bank of feature masks

Usage

```
mask_bank(masks, space, metadata = list())
```

Arguments

masks	Logical mask-by-feature matrix. Duplicate rows are stored once.
space	Exact feature space addressed by mask columns.
metadata	Serializable metadata.

Value

A serializable mask_bank.

Examples

```
space <- index_space(6, ids = paste0("f", 1:6), namespace = "validity-ex")
masks <- rbind(
  c(TRUE, TRUE, FALSE, TRUE, FALSE, TRUE),
  c(TRUE, FALSE, FALSE, TRUE, TRUE, TRUE)
)
bank <- mask_bank(masks, space)
n_masks(bank)
```

`memory_source` *Construct an in-memory array source*

Description

A memory source holds its values in the descriptor, so its revision fingerprint cannot be derived from a file state. It is instead derived from the shape, dtype, chunk grid, and a per-object identity token assigned once at construction, plus the optional caller-supplied `revision`. Construction therefore never hashes the payload, whatever its size. It reads values only when `dtype` narrows the data's natural type (for example `"logical"` or an integer width over doubles), to verify once that every value is representable; incompatible data raise `fmridataset_error_source_contract`. Reads return the R storage mode the dtype realizes to: doubles for every numeric dtype, logical for `"logical"`, complex for complex dtypes.

Usage

```
memory_source(
  data,
  dtype = NULL,
  chunks = NULL,
  revision = NULL,
  identity = c("object", "content")
)
```

Arguments

`data`	A two-dimensional matrix or array.
`dtype`	Logical storage dtype. Numeric R matrices default to `"float64"`.
`chunks`	Optional logical chunk shape.
`revision`	Optional single string naming the revision of `data`, such as a version label or an upstream checksum. It is never interpreted, but it replaces the per-object identity token, so two sources built independently under the same revision share a fingerprint: the caller is asserting they are the same source in the same revision.
`identity`	`"object"` assigns a fresh identity token (unless `revision` is supplied); `"content"` derives it from `content_hash()` of `data`.

Details

Two independently constructed memory sources with equal values have different fingerprints by design, and fingerprint equality is never value equality. A serialization round trip preserves the token, so a descriptor and its copies agree. Values compared across objects require `content_hash()`. Set `identity = "content"` to opt into a content-derived token: the payload is hashed once at construction and equal-valued sources then share a fingerprint at O(n) cost.

Value

A serializable `memory_source`.

See Also

source_fingerprint() and content_hash() for the fingerprint policy.

Examples

```
src <- memory_source(matrix(seq_len(6), nrow = 2))
source_shape(src)
source_dtype(src)
```

---

nifti_array_source *Construct a pushdown-aware NIfTI array source*

Description

The descriptor reads headers and one mask at construction, but no fMRI volumes. Numerical reads split requested global observations by file, pass local volume indices into neuroim2::read_vec(), and restrict the mask to requested packed features before materialization. Native reads return full-volume NeuroVec objects in requested observation order.

Usage

```
nifti_array_source(paths, mask, chunks = NULL)
```

Arguments

paths	One or more NIfTI files with a common spatial grid.
mask	A NIfTI mask path or a compatible volume_space.
chunks	Optional logical observation-by-feature chunk hint.

Details

The fingerprint covers the descriptor and the size and modification time of every file captured at construction, never the voxel values. Every open, read, and native read re-observes those files first and raises fmridataset_error_source_stale (with source, expected, actual, and changed fields) if any differ; genuine read failures remain fmridataset_error_backend_io. See content_hash() to identify values.

Value

A serializable nifti_array_source.

Examples

```
# A small NIfTI fixture shipped with neuroim2 stands in for real data.
path <- system.file("extdata", "global_mask_v4.nii", package = "neuroim2")
if (nzchar(path)) {
  src <- nifti_array_source(path, path)
  source_shape(src)
  source_dtype(src)
}
```

`nifti_source_space` *Recover the spatial domain of a NIfTI source*

Description

Recover the spatial domain of a NIfTI source

Usage

```
nifti_source_space(x, template = NULL)
```

Arguments

`x`	A `nifti_array_source`.
`template`	Optional template or native-space label.

Value

A compatible `volume_space`.

Examples

```
path <- system.file("extdata", "global_mask_v4.nii", package = "neuroim2")
if (nzchar(path)) {
  src <- nifti_array_source(path, path)
  spatial <- nifti_source_space(src, template = "fixture")
  n_features(spatial)
}
```

`observation_validity` *Resolve validity onto frame observations*

Description

Resolve validity onto frame observations

Usage

```
observation_validity(x, name = NULL)
```

Arguments

`x`	An `fmri_frame` or view.
`name`	Validity relation name.

Value

Observation-by-feature logical validity matrix.

Examples

```
space <- index_space(4, ids = paste0("f", 1:4), namespace = "validity-ex")
subjects <- entity_frame(
  data.frame(subject_id = c("sub-1", "sub-2")),
  key = "subject_id"
)
validity <- entity_feature_validity(
  entity = "subject", entity_ids = c("sub-1", "sub-2"),
  masks = rbind(
    c(TRUE, TRUE, FALSE, TRUE),
    c(TRUE, FALSE, FALSE, TRUE)
  ),
  space = space
)
frame <- fmri_frame(
  assays = list(signal = matrix(1:8, nrow = 2)),
  observations = data.frame(.obs_id = c("o1", "o2"), subject_id = c("sub-1", "sub-2")),
  space = space,
  entities = list(subject = subjects),
  relations = list(
    observation_subject = key_relation("subject_id"),
    subject_feature_validity = validity
  )
)
observation_validity(frame)
```

parcel-operators *Inspect parent-linked feature spaces and parcel-space operators*

Description

Inspect parent-linked feature spaces and parcel-space operators

Usage

```
parent_space(x)

parcel_membership(x)

parcel_aggregation(x)
```

Arguments

x	A parent-linked feature space such as a parcel_space or basis_space.

Value

parent_space() returns the parent feature space; parcel_membership() and parcel_aggregation() return sparse operators.

Examples

```
parent <- volume_space(c(3, 2, 1), support = 1:6)
membership <- Matrix::sparseMatrix(
  i = 1:6, j = c(1, 1, 1, 2, 2, 2), x = 1, dims = c(6L, 2L)
)
x <- parcel_space(parent, c(10L, 20L), membership, atlas = "toy-atlas")
parent_space(x)
parcel_membership(x)
```

parcel_space *Construct a parent-linked parcel feature space*

Description

A parcel_space owns only the feature-axis algebra required by fmridataset: stable parcel identity, a parent feature space, and explicit parent-to-parcel operators. Atlas discovery, labels, and provenance remain owned by neuroatlas and can be imported with parcel_space_from_atlas().

Usage

```
parcel_space(
  parent,
  parcel_ids,
  membership,
  data = NULL,
  atlas,
  aggregation = c("mean", "sum"),
  decoder = NULL,
  metadata = list()
)
```

Arguments

parent	Parent feature_space (usually a volume or surface space).
parcel_ids	Stable atlas-native parcel identifiers.
membership	Non-negative parent-feature by parcel membership weights.
data	One metadata row per parcel. The id, label, and hemi conventions match neuroatlas::parcel_data when present.
atlas	Atlas identity list containing at least id, or a scalar ID.
aggregation	Either weighted "mean" or weighted "sum".
decoder	Optional parent-feature by parcel reconstruction operator. The default blends overlapping parcel values by row-normalized membership; restrict_space() rebuilds it from the restricted membership, whereas a supplied decoder is subset to the retained parcels.
metadata	Additional serializable metadata.

Value

A parcel_space.

Examples

```
parent <- volume_space(c(3, 2, 1), support = 1:6)
membership <- Matrix::sparseMatrix(
  i = 1:6, j = c(1, 1, 1, 2, 2, 2), x = 1, dims = c(6L, 2L)
)
x <- parcel_space(parent, c(10L, 20L), membership, atlas = "toy-atlas")
n_features(x)
```

parcel_space_from_atlas

Build a parcel space from a neuroatlas atlas

Description

The adapter delegates atlas identity and label interpretation to neuroatlas. For surface atlases it uses get_roi() so atlas-specific hemisphere-local coding is not duplicated here.

Usage

```
parcel_space_from_atlas(
  atlas,
  parent,
  aggregation = c("mean", "sum"),
  metadata = list()
)
```

Arguments

atlas	A neuroatlas atlas or surfatlas.
parent	The aligned parent volume_space or surface_space.
aggregation	Aggregation method passed to parcel_space().
metadata	Serializable metadata passed to parcel_space().

Value

A parcel_space aligned to parent.

Examples

```
# Loading neuroatlas and neurosurf alone takes several seconds, so this
# example runs under --run-donttest rather than on every check.
if (requireNamespace("neuroatlas", quietly = TRUE) &&
  requireNamespace("neurosurf", quietly = TRUE)) {
  old_rgl <- Sys.getenv("RGL_USE_NULL", unset = NA)
  Sys.setenv(RGL_USE_NULL = "TRUE")
  left <- matrix(c(0, 0, 0, 1, 0, 0, 0, 1, 0), ncol = 3, byrow = TRUE)
  right <- sweep(left, 2, c(0, 0, 1), "+")
  faces <- matrix(c(0, 1, 2), nrow = 1)
  lh <- neurosurf::SurfaceGeometry(left, faces, "lh")
  rh <- neurosurf::SurfaceGeometry(right, faces, "rh")
  atlas <- list(
    name = "toy-surface",
    lh_atlas = neurosurf::NeuroSurface(lh, 1:3, c(1, 1, 0)),
    rh_atlas = neurosurf::NeuroSurface(rh, 1:3, c(2, 2, 0)),
    ids = 1:2, labels = c("A", "B"), orig_labels = c("A", "B"),
    hemi = c("left", "right"), network = NULL, cmap = NULL,
```

```
    surf_type = "pial", surface_space = "toy"
  )
  class(atlas) <- c("toy", "surfatlas", "atlas")
  parent <- surface_space(
    vertex_ids = c(paste0("L-", 1:3), paste0("R-", 1:3)),
    hemisphere = rep(c("left", "right"), each = 3),
    topology = rbind(c(1, 2, 3), c(4, 5, 6)),
    geometry = rbind(left, right),
    template = "toy"
  )
  x <- parcel_space_from_atlas(atlas, parent)
  feature_ids(x)
 if (is.na(old_rgl)) Sys.unsetenv("RGL_USE_NULL") else Sys.setenv(RGL_USE_NULL = old_rgl)
}
```

`plan_blocks` *Plan bounded observation-by-feature blocks*

Description

`plan_blocks()` uses source chunk hints and an explicit byte budget to build a serializable, metadata-only execution plan. `"imagewise"` blocks retain complete feature rows, `"featurewise"` blocks retain complete observation columns, and `"balanced"` blocks scale both axes in the source chunk ratio.

Usage

```
plan_blocks(
  x,
  assay = active_assay(x),
  layout = c("balanced", "imagewise", "featurewise"),
  memory_budget = getOption("fmridataset.block_budget", 512 * 1024^2),
  target_block_bytes = getOption("fmridataset.target_block_bytes", 4 * 1024^2)
)
```

Arguments

`x`	An `fmri_frame` or lazy view.
`assay`	Assay name.
`layout`	One of `"balanced"`, `"imagewise"`, or `"featurewise"`.
`memory_budget`	Hard maximum estimated peak bytes for one input block.
`target_block_bytes`	Preferred block size, capped by `memory_budget`.

Value

A serializable `frame_block_plan`.

Examples

```
frame <- fmri_frame(
  assays = list(signal = memory_source(matrix(seq_len(20), 5, 4))),
  observations = data.frame(.obs_id = sprintf("obs-%d", 1:5)),
  active_assay = "signal"
)
plan <- plan_blocks(frame, memory_budget = 10 * 1024^2)
plan
```

`provenance-graph` *Construct and inspect an immutable provenance graph*

Description

Construct and inspect an immutable provenance graph

Usage

```
provenance_graph(...)

validate_provenance_graph(x)

provenance_records(x)

provenance_tips(x)

provenance_digest(x)

append_provenance(x, records)
```

Arguments

`...`	`provenance_record` objects, or one list of records.
`x`	A `provenance_graph`.
`records`	One or more records appended to `x`.

Value

A validated `provenance_graph`, its records, tips, or digest.

Examples

```
r1 <- provenance_record("load", inputs = list(path = "toy.nii"))
g <- provenance_graph(r1)
r2 <- provenance_record("normalize", parents = r1$id)
g <- append_provenance(g, r2)
provenance_tips(g)
```

provenance_record *Create a content-addressed provenance record*

Description

Create a content-addressed provenance record

Usage

```
provenance_record(
  operation,
  parents = character(),
  inputs = list(),
  parameters = list(),
  outputs = list(),
  software = list(package = "fmridataset"),
  metadata = list()
)
```

Arguments

`operation`	Stable operation name.
`parents`	IDs of direct parent records.
`inputs`, `parameters`, `outputs`, `software`, `metadata`	Serializable record data.

Value

A `provenance_record`.

Examples

```
r <- provenance_record("normalize", inputs = list(method = "zscore"))
r$operation
r$id
```

read_bids_bold *Read one subject's preprocessed BIDS BOLD data as an fmri_frame*

Description

`read_bids_bold()` is the narrow BIDS on-ramp to the canonical frame API. It discovers fMRIPrep `desc-preproc` BOLD runs, creates deterministic volume IDs, resolves one common volume space, and returns a lazy NIfTI-backed frame. BOLD values are not read until an assay or spatial map is explicitly collected.

Usage

```
read_bids_bold(
  path,
  subject,
  task = NULL,
  session = NULL,
  run = NULL,
  space = NULL,
  derivative = "fmriprep",
  mask = "intersection",
  events = TRUE,
  chunks = NULL
)
```

Arguments

path	Path to a BIDS dataset containing fMRIPrep derivatives.
subject	One exact subject label, with or without the sub- prefix.
task, session, run, space	Optional exact BIDS entity selectors.
derivative	Currently only "fmriprep".
mask	"intersection" (default), an explicit NIfTI mask path, or a compatible volume_space.
events	Whether to attach matching BIDS events as a keyed event table.
chunks	Optional observation-by-feature chunk hint passed to nifti_array_source().

Details

Runs are ordered by session, task, and run number. Numeric run labels are compared as integers, so unpadded run-2 precedes run-10.

Each observation carries scan_id, subject_id, session_id, task_id, run_id, volume_index (1-based position within its run), run_time, and TR. run_time is the acquisition onset of the volume relative to the start of its run, (volume_index - 1) * TR. No start_time column is emitted, so as_sampling_frame() uses fmrihrf::sampling_frame()'s default offset of TR/2 and its sample times are run_time + TR/2. This is deliberate: it keeps fmrihrf's modelling default. For onset-aligned sampling, set a zero offset on the schema before converting:

```
schema <- temporal_schema(bold)
schema$start_time <- stats::setNames(rep(0, schema$n_runs), names(schema$run_lengths))
sf <- as_sampling_frame(schema)
```

or build the frame with a start_time = 0 observation column.

Value

A lazy fmri_frame with a signal assay.

Examples

```
## Not run:
bold <- read_bids_bold(
  "/data/my-study",
  subject = "01",
  task = "memory",
  space = "MNI152NLin2009cAsym"
)
first_run <- filter_obs(bold, run_id == "run-1")
map <- spatial_map(bold, observation = 1)

## End(Not run)
```

---

relation-accessors *Access frame relations*

Description

Access frame relations

Usage

```
relations(x, ...)

relation(x, name, ...)

relation_names(x)
```

Arguments

x	An fmri_frame, view, or relation registry.
...	Additional method arguments.
name	One registered relation name.

Value

relations() returns the registry; relation() returns one descriptor; relation_names() returns registry names.

Examples

```
registry <- relation_registry(
  observation_stimulus = key_relation("stimulus_id", target = "stimulus")
)
relations(registry)
relation(registry, "observation_stimulus")
relation_names(registry)
```

relation_registry *Construct a relation registry*

Description

Construct a relation registry

Usage

```
relation_registry(relations = list(), ...)
```

Arguments

relations	A named list of key_relation or sparse_relation descriptors.
...	Alternatively, named relation descriptors.

Value

A named relation_registry.

Examples

```
registry <- relation_registry(
  observation_stimulus = key_relation("stimulus_id", target = "stimulus")
)
relation_names(registry)
```

relation_registry_digest *Compute a stable relation-registry digest*

Description

Compute a stable relation-registry digest

Usage

```
relation_registry_digest(x)
```

Arguments

x	A frame, view, or relation registry.

Value

A hexadecimal digest over the normalized registry.

Examples

```
registry <- relation_registry(
  observation_stimulus = key_relation("stimulus_id", target = "stimulus")
)
relation_registry_digest(registry)
```

`row_sharded_source` *Construct a manifest-backed row-sharded source*

Description

`row_sharded_source()` presents compatible child sources as one logical observation-by-feature array. Stable shard IDs and explicit boundaries make global-to-local row routing inspectable and serializable. Reads are grouped by touched shard, so an arbitrary ordered selector is issued at most once to each selected child.

Usage

```
row_sharded_source(sources, shard_ids = NULL, shard_data = NULL)
```

Arguments

`sources`	A non-empty list of compatible two-dimensional array sources.
`shard_ids`	Stable, unique shard identifiers.
`shard_data`	Optional scalar metadata with one row per shard. Names used by the shard manifest are reserved.

Value

A serializable `row_sharded_source`.

Examples

```
shards <- list(
  memory_source(matrix(1:4, nrow = 2)),
  memory_source(matrix(5:8, nrow = 2))
)
src <- row_sharded_source(shards)
source_shape(src)
```

select_features *Select frame features using feature metadata*

Description

Select frame features using feature metadata

Usage

```
select_features(x, predicate)
```

Arguments

x	An fmri_frame or view.
predicate	A metadata expression returning one logical value per feature.

Value

An fmri_view.

Examples

```
sp <- volume_space(dim = c(2, 2, 2), affine = diag(4))
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(4 * n_features(sp)), nrow = 4)),
  observations = data.frame(.obs_id = sprintf("vol-%d", 1:4)),
  space = sp
)
select_features(frame, i == 1)
```

shard_manifest *Inspect a row-sharded source manifest*

Description

Inspect a row-sharded source manifest

Usage

```
shard_manifest(x)
```

Arguments

x	A row_sharded_source.

Value

A data frame describing stable IDs, logical row ranges, source fingerprints, and user-supplied shard metadata.

Examples

```
shards <- list(memory_source(matrix(1:4, nrow = 2)), memory_source(matrix(5:8, nrow = 2)))
shard_manifest(row_sharded_source(shards))
```

---

source_descriptor *Inspect and validate an array-source contract*

Description

A valid canonical source is two-dimensional, has an explicit supported dtype and chunk grid, advertises serializable block slicing, provides a stable non-empty fingerprint, and contains no runtime handles or closures.

Usage

```
source_descriptor(x)

validate_array_source(x)
```

Arguments

x	An `array_source` descriptor.

Value

`source_descriptor()` returns a plain serializable contract list. `validate_array_source()` invisibly returns `x` or raises a structured source-contract error.

Examples

```
src <- memory_source(matrix(seq_len(6), nrow = 2))
source_descriptor(src)
validate_array_source(src)
```

`source_error` *Build the structured conditions an array source may signal*

Description

Storage packages implement the array source protocol and must fail the way built-in sources fail, so that callers can handle every source alike. `source_error()` builds the condition object for the three failure kinds the protocol defines; signal it with `stop()`.

Usage

```
source_error(message, type = c("stale", "io", "contract"), ...)
```

Arguments

`message`	One-sentence description of the failure.
`type`	One of `"stale"`, `"io"`, or `"contract"`.
`...`	Named fields stored on the condition, such as `source`, `expected`, `actual`, `file`, `operation`, or `field`.

Details

- `"stale"` (`fmridataset_error_source_stale`): the physical store no longer matches the descriptor's revision evidence (a file was rewritten, a store was replaced). Supply `source`, `expected`, and `actual` so the caller can report what changed.
- `"io"` (`fmridataset_error_backend_io`): a genuine read, open, or close failure. Supply `file` and `operation` where known.
- `"contract"` (`fmridataset_error_source_contract`): the descriptor or a method violates the protocol. Supply `field`.

Every condition also carries the `fmridataset_error` class, so a single handler can catch all package errors.

Value

A condition object inheriting from the type-specific class, `fmridataset_error`, `error`, and `condition`.

Examples

```
cond <- source_error(
  "Store was rewritten after the descriptor was built.",
  type = "stale", source = "example.h5",
  expected = "abc", actual = "def"
)
inherits(cond, "fmridataset_error_source_stale")
tryCatch(stop(cond), fmridataset_error_source_stale = function(e) e$actual)
```

source_view *Construct a lazy view over an array source*

Description

A view stores its selectors in the package's normalized selection form rather than as expanded position vectors: a select-all axis stores no vector, one contiguous run stores its bounds, and only an arbitrary subset stores positions. A view over a view composes into one view over the root source. Selectors follow the package-wide normalization law: logical masks must match the axis length without NA; numeric positions must be whole numbers, may reorder, may be negative (but not mixed with positive), drop zero, must be in bounds, and may not repeat an element; an empty selection is legal. Fingerprints hash the normalized form, so equal selections agree however they were expressed and a select-all view fingerprints in constant time.

Usage

```
source_view(source, observations = NULL, features = NULL)
```

Arguments

source	An array_source.
observations	Stored observation selector.
features	Stored feature selector.

Value

A serializable source view.

Examples

```
src <- memory_source(matrix(seq_len(6), nrow = 2))
view <- source_view(src, observations = 1)
source_shape(view)
```

space *Feature-space accessor*

Description

Feature-space accessor

Usage

```
space(x, ...)
```

Arguments

x	An object with spatial identity.
...	Additional arguments.

Value

A FeatureSpace.

Examples

```
sp <- volume_space(dim = c(2, 2, 2), affine = diag(4))
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(4 * n_features(sp)), nrow = 4)),
  observations = data.frame(.obs_id = sprintf("vol-%d", 1:4)),
  space = sp
)
space(frame)
```

---

sparse_relation *Describe an explicit sparse or many-to-many relation*

Description

Describe an explicit sparse or many-to-many relation

Usage

```
sparse_relation(
  data,
  from,
  to,
  from_col = ".from_id",
  to_col = ".to_id",
  weight = NULL,
  directed = TRUE,
  metadata = list()
)
```

Arguments

data	Scalar edge table.
from	Source domain.
to	Target domain.
from_col	Column containing source stable IDs.
to_col	Column containing target stable IDs.
weight	Optional numeric weight column.
directed	Whether edge direction is semantically meaningful.
metadata	Additional serializable metadata.

Value

A serializable sparse_relation descriptor.

Examples

```
edges <- sparse_relation(
  data = tibble::tibble(
    .from_id = c("obs-1", "obs-2"),
    .to_id = c("stim-1", "stim-2"),
    weight = c(0.7, 0.3)
  ),
  from = "observation",
  to = "entity:stimulus",
  weight = "weight"
)
edges$from
```

---

spatial_map *Recover a spatial map for one observation*

Description

Recover a spatial map for one observation

Usage

```
spatial_map(x, observation, assay = active_assay(x))
```

Arguments

x	An fmri_frame or view.
observation	Observation ID or one integer position.
assay	Assay name.

Value

A reconstructed spatial object.

Examples

```
sp <- volume_space(dim = c(2, 2, 2), affine = diag(4))
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(4 * n_features(sp)), nrow = 4)),
  observations = data.frame(.obs_id = sprintf("vol-%d", 1:4)),
  space = sp
)
spatial_map(frame, observation = 1L)
```

study-accessors *Study representation accessors*

Description

Study representation accessors

Usage

```
study_frames(x, contextual = TRUE)

study_frame(x, name, contextual = TRUE)

study_ids(x)
```

Arguments

x	An fmri_study.
contextual	Replace frame-local entity stubs with shared study entities.
name	Stable representation name.

Value

Named representations, one representation, or representation IDs.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
study <- fmri_study(list(main = frame))
study_ids(study)
study_frame(study, "main")
```

study-registries *Study link and table accessors*

Description

Study link and table accessors

Usage

```
study_links(x)

study_link(x, name)

study_tables(x)

study_table(x, name)
```

Arguments

x	An fmri_study.
name	Stable link or table name.

Value

A registry or one descriptor/table.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
study <- fmri_study(list(main = frame))
study_links(study)
study_tables(study)
```

---

study_digest *Compute a deterministic study digest*

Description

Compute a deterministic study digest

Usage

```
study_digest(x)
```

Arguments

x	An fmri_study or filtered view.

Value

A SHA-256 digest computed without numerical reads.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
study <- fmri_study(list(main = frame))
study_digest(study)
```

---

study_from_fds_manifest

Reconstruct a study from semantic and physical components

Description

Reconstruct a study from semantic and physical components

Usage

```
study_from_fds_manifest(manifest, representations, bindings = list())
```

Arguments

manifest	A valid FDS v2 study manifest.
representations	Named lazy frames or collections matching the representation manifests.
bindings	Named physical bindings for shared study arrays.

Value

An fmri_study.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
study <- fmri_study(list(main = frame))
manifest <- fds_study_manifest(study)
rebuilt <- study_from_fds_manifest(
  manifest, list(main = frame), fds_study_bindings(study)
)
study_ids(rebuilt)
```

surface_space *Construct a packed cortical surface feature space*

Description

Construct a packed cortical surface feature space

Usage

```
surface_space(
  vertex_ids,
  hemisphere,
  support = NULL,
  topology = NULL,
  geometry = NULL,
  medial_wall = NULL,
  template = NULL,
  units = "mm",
  surf_to_world = diag(4),
  metadata = list()
)
```

Arguments

vertex_ids	Stable IDs for every vertex in the full mesh.
hemisphere	One "left" or "right" label per full-mesh vertex.
support	Active vertex positions or IDs. By default, all non-medial-wall vertices are active.
topology	A three-column face matrix or asset descriptor with reference, digest, and optional data/faces.
geometry	A vertex-by-three coordinate matrix or asset descriptor with reference, digest, and optional data/coordinates.
medial_wall	Logical full-mesh medial-wall mask.
template	Optional template identity such as "fsLR-32k".
units	Coordinate units.
surf_to_world	A finite 4 by 4 surface-to-world transform, following the neurosurf::SurfaceGeometry convention.
metadata	Additional serializable metadata.

Value

A surface_space.

Examples

```
surface_space(
  vertex_ids = c("L-1", "L-2", "L-3"),
  hemisphere = rep("left", 3)
)
```

`surface_space_from_neurosurf` *Adapt a neurosurf geometry to a surface feature space*

Description

`neurosurf` remains the owner of mesh geometry and algorithms. This adapter extracts its stable topology, coordinates, hemisphere, and world transform into the backend-neutral identity required by an `fmri_frame`.

Usage

```
surface_space_from_neurosurf(
  geometry,
  vertex_ids = NULL,
  support = NULL,
  medial_wall = NULL,
  template = NULL,
  units = "mm",
  metadata = list()
)
```

Arguments

`geometry`	A `neurosurf::SurfaceGeometry`.
`vertex_ids`	Optional stable full-mesh vertex IDs.
`support`, `medial_wall`, `template`, `units`, `metadata`	Passed to `surface_space()`.

Value

A `surface_space`.

Examples

```
if (requireNamespace("neurosurf", quietly = TRUE)) {
  old_rgl <- Sys.getenv("RGL_USE_NULL", unset = NA)
  Sys.setenv(RGL_USE_NULL = "TRUE")
  vertices <- matrix(c(0, 0, 0, 1, 0, 0, 0, 1, 0), ncol = 3, byrow = TRUE)
  geom <- neurosurf::SurfaceGeometry(
    vertices, matrix(c(0, 1, 2), nrow = 1),
```

```
    hemi = "lh", label = "pial"
  )
  x <- surface_space_from_neurosurf(geom, template = "toy-surface")
  feature_ids(x)
 if (is.na(old_rgl)) Sys.unsetenv("RGL_USE_NULL") else Sys.setenv(RGL_USE_NULL = old_rgl)
}
```

`temporal-schema` *The observation-axis temporal contract*

Description

The canonical frame model carries no acquisition timing of its own: an `fmri_frame` is observations by features, and when the observations happen to be volumes acquired in runs, that fact is ordinary observation metadata. The package already writes it that way – `read_bids_bold()` emits `run_id` and `TR` columns – but nothing validated the convention, so every consumer reinvented it and none could rely on it.

Usage

```
temporal_schema(
  x,
  run_col = NULL,
  tr_col = "TR",
  start_time_col = "start_time",
  censor_col = "censor",
  order_col = NULL
)

has_temporal_schema(x, ...)

as_sampling_frame(x, ...)
```

Arguments

`x`	An `fmri_frame` or `fmri_view`.
`run_col`, `tr_col`, `start_time_col`, `censor_col`	Observation metadata columns holding the run label, repetition time, first-volume offset, and censoring indicator. Each is a single column name; `run_col` may also be `NULL` (the default) to use the key of an observation relation to a run entity, else `scan_id`, else `run_id`. A named column other than `run_col` that is absent is simply not used.
`order_col`	`NULL` (default) to use the first of `volume_index` and `run_time` that is present, or a single column name holding a numeric within-run acquisition index or time, which must then exist.
`...`	Passed to `temporal_schema()`.

Details

These functions make it a contract. `temporal_schema()` derives a validated description from the observation metadata; `as_sampling_frame()` reconstructs the `fmrihrf::sampling_frame` that design and HRF modelling code expects.

The schema is derived, never stored. The columns are the truth, so the schema cannot go stale, needs no serialization of its own, and follows subsetting, reordering, and binding for free.

Value

`temporal_schema()` returns a `frame_temporal_schema`. `has_temporal_schema()` returns a scalar logical. `as_sampling_frame()` returns an `fmrihrf::sampling_frame`.

Contract

`run_id` Required. One value per observation naming the acquisition run it belongs to. Any type; compared as character. No missing values.

`TR` Optional. Repetition time in seconds, positive and finite, and constant within each run (compared with a relative tolerance of `sqrt(.Machine$double.eps)`, so floating-point noise is not a second value). Runs may differ from one another, matching `fmrihrf::sampling_frame()`.

`start_time` Optional. Non-negative finite offset in seconds of the first volume of each run (the slice-time-correction reference, or any other first-volume acquisition time). Constant within each run; runs may differ. When absent, `as_sampling_frame()` leaves the offset to `fmrihrf::sampling_frame()`'s default of `TR/2`.

`censor` Optional. Logical, one value per observation, `TRUE` where the observation is to be excluded. No missing values.

order column Optional. A numeric within-run acquisition index or time, such as `volume_index` or `run_time` from `read_bids_bold()`. No missing values.

Timing of samples

`read_bids_bold()` emits `run_time`, the acquisition onset of each volume relative to its run (`(volume_index - 1) * TR`). The sampling frame built by `as_sampling_frame()` places samples at `start_time + (volume_index - 1) * TR`, and when no `start_time` column is supplied fmrihrf's default `start_time = TR/2` applies, so sample times are `run_time + TR/2`. This modelling default is kept on purpose. For onset-aligned sampling, add a `start_time` column of `0` when constructing the frame, or set it on the schema before converting:

```
schema <- temporal_schema(frame)
schema$start_time <- stats::setNames(rep(0, schema$n_runs), names(schema$run_lengths))
as_sampling_frame(schema)
```

Order and contiguity

Runs are numbered in order of first appearance, not by sorting, so `block_ids` is stable under any operation that preserves observation order. A frame is *contiguous* when each run occupies one unbroken stretch of observations. Frames are not required to be contiguous – `filter_obs()` and ID-based reordering both produce legal interleaved views – but a `sampling_frame` is a run-length encoding

and cannot represent one, so `as_sampling_frame()` refuses a non-contiguous frame rather than silently reordering it.

Contiguity says nothing about order *within* a run. When an order column is present (`order_col`, by default the first of `volume_index` and `run_time` found), the schema records in `ordered` whether it increases strictly within every run, and `as_sampling_frame()` refuses a frame whose volumes are out of acquisition order. Without such a column the order within a run cannot be checked: `ordered` is `NA`, and the observation order is taken as acquisition order.

A zero-observation frame has an empty schema but no sampling frame; `as_sampling_frame()` refuses it.

Examples

```
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(12), 6, 2)),
  observations = data.frame(
    .obs_id = sprintf("t%02d", 1:6),
    run_id = rep(c("run-1", "run-2"), each = 3),
    TR = 2
  )
)
schema <- temporal_schema(frame)
schema$run_lengths
if (requireNamespace("fmrihrf", quietly = TRUE)) {
  as_sampling_frame(frame)
}
```

---

unaligned_record *Construct a typed unaligned metadata record*

Description

Container metadata is deliberately unaligned. Array- or table-valued data belong in assays, axis blocks, relations, typed tables, or linked frames. A vector field is an unaligned value whatever its length; a length that coincides with an axis or entity size is not treated as alignment.

Usage

```
unaligned_record(x = list())
```

Arguments

`x`	A named serializable list.

Value

An `unaligned_record`.

Examples

```
rec <- unaligned_record(list(task = "rest", tr = 2))
rec$task
```

`validate_event_table` *Validate a keyed event table*

Description

Validate a keyed event table

Usage

```
validate_event_table(x)
```

Arguments

x	An `fmri_event_table`.

Value

`x`, invisibly.

Examples

```
et <- event_table(
  data.frame(event_id = c("e1", "e2"), onset = c(0, 10)),
  key = "event_id"
)
validate_event_table(et)
```

`validate_fmri_collection`
Validate an fMRI collection

Description

Validate an fMRI collection

Usage

```
validate_fmri_collection(x)
```

Arguments

x	An `fmri_collection`.

Value

x, invisibly, or a structured collection error.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
collection <- fmri_collection(list(sub01 = frame, sub02 = frame))
validate_fmri_collection(collection)
```

---

validate_fmri_study *Validate a study*

Description

Validate a study

Usage

```
validate_fmri_study(x)
```

Arguments

x	An fmri_study.

Value

x, invisibly.

Examples

```
voxels <- volume_space(dim = c(2, 2, 1), affine = diag(4), template = "toy")
frame <- fmri_frame(
  assays = list(bold = matrix(rnorm(3 * n_features(voxels)), nrow = 3)),
  observations = data.frame(.obs_id = paste0("vol-", 1:3)),
  space = voxels
)
study <- fmri_study(list(main = frame))
validate_fmri_study(study)
```

validate_relation_registry

Validate a relation registry

Description

Validate a relation registry

Usage

```
validate_relation_registry(
  x,
  observations = NULL,
  features = NULL,
  entities = NULL
)
```

Arguments

x	A relation_registry.
observations	Optional observation axis_frame.
features	Optional feature axis_frame.
entities	Optional entity_registry.

Value

Invisibly returns x; contextual validation also enforces all foreign-key and edge identities.

Examples

```
registry <- relation_registry(
  observation_stimulus = key_relation("stimulus_id", target = "stimulus")
)
validate_relation_registry(registry)
```

validate_unaligned_record

Validate a typed unaligned metadata record

Description

Validate a typed unaligned metadata record

Usage

```
validate_unaligned_record(x)
```

Arguments

x	An unaligned_record.

Value

x, invisibly.

Examples

```
rec <- unaligned_record(list(task = "rest", tr = 2))
validate_unaligned_record(rec)
```

---

validity_coverage *Summarize feature coverage without imposing an analysis policy*

Description

Summarize feature coverage without imposing an analysis policy

Usage

```
validity_coverage(x, name = NULL, domain = c("entity", "observation"))
```

Arguments

x	An fmri_frame or view.
name	Validity relation name.
domain	Weight unique entities or frame observations.

Value

Named fraction-valid vector over frame features.

Examples

```
space <- index_space(4, ids = paste0("f", 1:4), namespace = "validity-ex")
subjects <- entity_frame(
  data.frame(subject_id = c("sub-1", "sub-2")),
  key = "subject_id"
)
validity <- entity_feature_validity(
  entity = "subject", entity_ids = c("sub-1", "sub-2"),
  masks = rbind(
    c(TRUE, TRUE, FALSE, TRUE),
    c(TRUE, FALSE, FALSE, TRUE)
  ),
  space = space
)
```

```
frame <- fmri_frame(
  assays = list(signal = matrix(1:8, nrow = 2)),
  observations = data.frame(.obs_id = c("o1", "o2"), subject_id = c("sub-1", "sub-2")),
  space = space,
  entities = list(subject = subjects),
  relations = list(
    observation_subject = key_relation("subject_id"),
    subject_feature_validity = validity
  )
)
validity_coverage(frame)
```

`validity_masked_source` *Lazily mask invalid observation-feature cells with missing values*

Description

Lazily mask invalid observation-feature cells with missing values

Usage

```
validity_masked_source(source, observation_mask_id, bank)
```

Arguments

`source`	Observation-by-feature array source.
`observation_mask_id`	One mask-bank ID per source row.
`bank`	Compatible `mask_bank`.

Value

A serializable `validity_masked_source`.

Examples

```
space <- index_space(4, ids = paste0("f", 1:4), namespace = "validity-ex")
bank <- mask_bank(
  rbind(c(TRUE, TRUE, FALSE, TRUE), c(TRUE, FALSE, FALSE, TRUE)),
  space
)
src <- memory_source(matrix(1:8, nrow = 2))
masked <- validity_masked_source(src, bank$mask_ids[c(1, 2)], bank)
source_read(masked)
```

volume_space *Construct a packed volumetric feature space*

Description

Construct a packed volumetric feature space

Usage

```
volume_space(
  dim,
  affine = diag(4),
  support = NULL,
  template = NULL,
  units = "mm",
  metadata = list()
)
```

Arguments

dim	Three spatial dimensions.
affine	A 4 by 4 voxel-to-world affine.
support	Logical full-volume support or packed linear indices.
template	Optional template/native-space identity.
units	Spatial units.
metadata	Additional serializable metadata.

Value

A volume_space.

Examples

```
volume_space(c(2, 2, 2), affine = diag(4), support = 1:4)
```

write_frame *Persist and reopen an fmri frame*

Description

write_frame() writes an fmri_frame to an HDF5 file and open_frame() reopens one. Both require the fmristore package, which performs the physical HDF5 work. fmristore is not on CRAN; install it from https://bbuchsbaum.r-universe.dev, for example with install.packages("fmristore", repos = c("https://bbuchsbaum.r-universe.dev", getOption("repos"))). Without it, or with a version that lacks frame support, both functions fail with an error of class fmridataset_error_backend_io.

Usage

```
write_frame(x, path, format = "hdf5", ...)

open_frame(path, format = "hdf5", ...)
```

Arguments

x	An fmri_frame.
path	Destination or source path.
format	Storage format. Only "hdf5" is supported.
...	Arguments passed to the physical store implementation.

Details

Reopened assays are lazy sources: opening a frame does not read assay values.

Neither function computes a content hash. Persistence records semantic manifest digests and source fingerprints only; a caller who wants a value receipt for the written or reopened arrays requests it explicitly with content_hash() and records the result where it is needed.

Value

write_frame() invisibly returns the committed path, normalized with forward slashes on every platform. open_frame() returns an fmri_frame.

Examples

```
if (requireNamespace("fmristore", quietly = TRUE)) {
  src <- memory_source(matrix(seq_len(6), nrow = 2))
  obs <- tibble::tibble(.obs_id = c("o1", "o2"))
  space <- index_space(3, id_policy = "deterministic", namespace = "demo")
  frame <- fmri_frame(list(beta = src), obs, space = space)
  path <- tempfile(fileext = ".h5")
  committed <- write_frame(frame, path)
  reopened <- open_frame(committed)
  collect_assay(reopened, "beta")
}
```

---

zarr_array_source *Construct an experimental Zarr array source*

Description

zarr_array_source() describes one two-dimensional Zarr array whose logical axes are observations and features. The descriptor contains no open store, R6 object, external pointer, or loader function. Runtime handles are opened only for metadata discovery and numerical reads.

Usage

```
zarr_array_source(
  uri,
  array_path = "/",
  shape = NULL,
  dtype = NULL,
  chunks = NULL,
  physical_axes = c("observation", "feature")
)
```

Arguments

uri	One local path, file URI, or HTTP(S) location understood by zarr::open_zarr().
array_path	Absolute path of the array within the Zarr hierarchy. The default "/" denotes a single-array store.
shape	Optional logical observation-by-feature shape.
dtype	Optional logical storage dtype supported by ArraySource.
chunks	Optional logical observation-by-feature chunk shape.
physical_axes	Names of the two physical Zarr dimensions, permitting either observation-first or feature-first storage.

Details

The optional zarr package is needed only when metadata must be discovered or data are read. Supplying shape, dtype, and chunks together therefore permits metadata-only construction and serialization on workers where Zarr is not installed.

The fingerprint covers the descriptor: URI, array path, logical shape, dtype, chunks, and axis order, never the stored values. Opening a handle re-reads the store metadata and raises fmridataset_error_source_stale (with source, expected, and actual fields) if shape, chunks, or dtype changed; a store that cannot be opened is fmridataset_error_backend_io. See content_hash() to identify values.

Value

A serializable zarr_array_source descriptor.

Examples

```
# Metadata-only construction needs no Zarr store and no zarr package.
src <- zarr_array_source(
  "fixture.zarr",
  shape = c(5L, 6L), dtype = "float64", chunks = c(2L, 3L)
)
source_shape(src)
source_chunks(src)
```

Index

active_assay *(frame-accessors)*, 58
adjacency *(feature-space)*, 48
aligned_assay_set, 5
append_provenance *(provenance-graph)*, 80
append_source_shards, 5
apply_feature_validity, 6
array source, *9*, *88*
array-source, 7, *9*
as_array_source *(array-source)*, 7
as_delarr, 9
as_fmri_frame, 10
as_sampling_frame *(temporal-schema)*, 97
as_sampling_frame(), *82*
assay *(frame-accessors)*, 58
assays *(frame-accessors)*, 58
assert_same_space *(feature-space)*, 48
auxiliary-table, 10
auxiliary_table, 11
axis_block, 12
axis_block_data *(axis_block)*, 12
axis_blocks *(axis_frame)*, 13
axis_data *(axis_frame)*, 13
axis_frame, 13
axis_id_policy *(id-policy)*, 66
axis_ids *(axis_frame)*, 13

basis-operators, 14
basis_analysis *(basis-operators)*, 14
basis_projection_info *(basis-operators)*, 14
basis_space, 15
basis_space(), *16*, *48*
basis_space_from_decoder, 16
basis_space_from_fmrilatent, 17
basis_synthesis *(basis-operators)*, 14
bind_observations, 18
block_apply, 19
block_component_ids *(axis_block)*, 12
block_components *(axis_block)*, 12
block_manifest, 20

canonical-encoding, 20
canonical_bytes *(canonical-encoding)*, 20
canonical_sha256 *(canonical-encoding)*, 20
canonicalization_contract, 21
collect_assay, 22
collect_spatial_maps, 23
collection-accessors, 24
collection-spaces, 24
collection_common_space *(collection-spaces)*, 24
collection_digest, 25
collection_frame *(collection-accessors)*, 24
collection_frames *(collection-accessors)*, 24
collection_ids *(collection-accessors)*, 24
collection_space_data *(collection-spaces)*, 24
compare_frame_schema *(frame-schema-validation)*, 59
compose_frame_links, 26
composite-parts, 26
composite_part *(composite-parts)*, 26
composite_part_names *(composite-parts)*, 26
composite_parts *(composite-parts)*, 26
composite_space, 27
composite_space(), *48*
content_hash, 28
content_hash(), *8*, *66*, *67*, *72*, *73*, *106*, *107*
content_hash_contract, 30
counting_source, 30

dim.fmri_frame *(frame-accessors)*, 58
dim.fmri_view *(frame-accessors)*, 58

entities *(entity-accessors)*, 31
entity *(entity-accessors)*, 31

entity-accessors, 31
entity-frame-accessors, 32
entity_blocks (entity-frame-accessors), 32
entity_data (entity-frame-accessors), 32
entity_feature_validity, 33
entity_frame, 34
entity_ids (entity-frame-accessors), 32
entity_key (entity-frame-accessors), 32
entity_names (entity-accessors), 31
entity_registry, 34
entity_registry_digest, 35
event-accessors, 36
event_data (event-accessors), 36
event_key (event-accessors), 36
event_table, 36
execute_block_plan, 37
execute_spatial, 38
execution_path, 39
explain, 40

fault_source, 41
fds_frame_bindings, 41
fds_frame_manifest, 42
fds_manifest_digest, 43
fds_schema, 44
fds_schema_version (fds_schema), 44
fds_study_bindings, 44
fds_study_manifest, 45
fds_study_manifest_digest, 46
fds_study_representations, 46
feature-map-accessors, 47
feature-space, 48
feature-validity-accessors, 49
feature_axis, 50
feature_blocks (frame-accessors), 58
feature_data (feature-space), 48
feature_ids (feature-space), 48
feature_map, 51
feature_map_digest (feature-map-accessors), 47
feature_map_from_target, 52
feature_map_operator (feature-map-accessors), 47
feature_map_source_space (feature-map-accessors), 47
feature_map_target_space (feature-map-accessors), 47
feature_mapped_source, 53
features (frame-accessors), 58
filter_entities, 53
filter_obs, 54
fmri_collection, 55
fmri_frame, 56
fmri_study, 57
frame-accessors, 58
frame-schema-validation, 59
frame_from_fds_manifest, 60
frame_link, 61
frame_schema, 62
frame_schema_digest (frame-schema-validation), 59

has_temporal_schema (temporal-schema), 97
hierarchy-accessors, 63
hierarchy_complete (hierarchy-accessors), 63
hierarchy_digest, 64
hierarchy_groups (hierarchy-accessors), 63
hierarchy_ids (hierarchy-accessors), 63
hierarchy_index, 65
hierarchy_levels (hierarchy-accessors), 63
hierarchy_relations (hierarchy-accessors), 63

id-policy, 66
identity_descriptor, 66
identity_descriptor(), *28*, *29*
ids_are_durable (id-policy), 66
index_space, 67
index_space(), *48*

key_relation, 68

locate_source_rows, 69

map_features, 69
mask-bank-accessors, 70
mask_bank, 71
mask_bank_digest (mask-bank-accessors), 70
mask_values (mask-bank-accessors), 70
memory_source, 72
memory_source(), *8*

n_features (feature-space), 48

n_masks *(mask-bank-accessors)*, 70
native_shape *(feature-space)*, 48
neuroim2::read_vec(), *73*
nifti_array_source, 73
nifti_array_source(), *82*
nifti_source_space, 74

obs_blocks *(frame-accessors)*, 58
observation_axis *(frame-accessors)*, 58
observation_ids *(frame-accessors)*, 58
observation_validity, 75
observations *(frame-accessors)*, 58
open_frame *(write_frame)*, 105

parcel-operators, 76
parcel_aggregation *(parcel-operators)*, 76
parcel_membership *(parcel-operators)*, 76
parcel_space, 76
parcel_space(), *48*, *78*
parcel_space_from_atlas, 78
parcel_space_from_atlas(), *76*
parent_space *(parcel-operators)*, 76
plan_blocks, 79
provenance-graph, 80
provenance_digest *(provenance-graph)*, 80
provenance_graph *(provenance-graph)*, 80
provenance_record, 81
provenance_records *(provenance-graph)*, 80
provenance_tips *(provenance-graph)*, 80

read_bids_bold, 81
reconstruct_space *(feature-space)*, 48
relation *(relation-accessors)*, 83
relation-accessors, 83
relation_names *(relation-accessors)*, 83
relation_registry, 84
relation_registry_digest, 84
relations *(relation-accessors)*, 83
reset_source_counts *(counting_source)*, 30
restrict_space *(feature-space)*, 48
restrict_space(), *77*
row_sharded_source, 85

same_space *(feature-space)*, 48
select_features, 86
shard_manifest, 86
source_capabilities *(array-source)*, 7
source_chunks *(array-source)*, 7
source_close *(array-source)*, 7
source_counts *(counting_source)*, 30
source_descriptor, 87
source_dtype *(array-source)*, 7
source_error, 88
source_fingerprint *(array-source)*, 7
source_fingerprint(), *29*, *73*
source_open *(array-source)*, 7
source_read *(array-source)*, 7
source_read(), *9*
source_read_native *(array-source)*, 7
source_realization_cost *(array-source)*, 7
source_shape *(array-source)*, 7
source_view, 89
space, 89
space_digest *(feature-space)*, 48
sparse_relation, 90
spatial_map, 91
study-accessors, 92
study-registries, 92
study_digest, 93
study_frame *(study-accessors)*, 92
study_frames *(study-accessors)*, 92
study_from_fds_manifest, 94
study_ids *(study-accessors)*, 92
study_link *(study-registries)*, 92
study_links *(study-registries)*, 92
study_table *(study-registries)*, 92
study_tables *(study-registries)*, 92
surface_space, 95
surface_space(), *48*, *96*
surface_space_from_neurosurf, 96

table_data *(auxiliary-table)*, 10
table_key *(auxiliary-table)*, 10
table_role *(auxiliary-table)*, 10
temporal-schema, 97
temporal_schema *(temporal-schema)*, 97

unaligned_record, 99

validate_against_schema *(frame-schema-validation)*, 59
validate_array_source *(source_descriptor)*, 87

validate_auxiliary_table
(auxiliary-table), 10
validate_entity_feature_validity
(feature-validity-accessors), 49
validate_entity_registry
(entity_registry), 34
validate_event_table, 100
validate_fds_manifest
(fds_frame_manifest), 42
validate_fds_study_manifest
(fds_study_manifest), 45
validate_feature_map
(feature-map-accessors), 47
validate_fmri_collection, 100
validate_fmri_study, 101
validate_frame_schema
(frame-schema-validation), 59
validate_mask_bank
(mask-bank-accessors), 70
validate_provenance_graph
(provenance-graph), 80
validate_relation_registry, 102
validate_unaligned_record, 102
validity_coverage, 103
validity_entity
(feature-validity-accessors), 49
validity_entity_ids
(feature-validity-accessors), 49
validity_mask_bank
(feature-validity-accessors), 49
validity_masked_source, 104
validity_matrix
(feature-validity-accessors), 49
validity_space
(feature-validity-accessors), 49
vectorize_space (feature-space), 48
volume_space, 105
volume_space(), *48*

write_frame, 105

zarr_array_source, 106