Package: fmrigds (via r-universe)

May 7, 2026

Type Package

Title Lazy, Format-Agnostic Group Analysis for fMRI

Version 0.1.0.9000

Depends R (>= 4.1.0)

Description Provides a Group Data Set (GDS) abstraction and lazy analysis pipeline for first-level fMRI statistical outputs. Unifies tabular, NIfTI, HDF5, and fmristore inputs under a common sample x subject x contrast representation; supports space-aware transformations, group and meta-analytic reducers, provenance tracking, and reproducible export.

License MIT + file LICENSE

Encoding UTF-8

LazyData false

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports Matrix, digest, jsonlite, data.table, hdf5r

LinkingTo Rcpp, RcppArmadillo

URL https://github.com/bbuchsbaum/fmrigds

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

Suggests testthat (>= 3.0.0), arrow, RNifti, lme4, neuroim2, neurotabs, neurothresh, knitr, rmarkdown, pkgload, covr, pkgdown

VignetteBuilder knitr

Config/testthat/edition 3

Config/pak/sysreqs libhdf5-dev

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

Date/Publication 2026-05-07 15:04:45 UTC

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

RemoteRef HEAD

RemoteSha 2549d29da6e079265db72e39a23dbd255eba5ac8

Contents

`fmrigds-package` *fmrigds*

Description

Lazy, format-agnostic group analysis for fMRI.

Details

fmrigds provides a common Group Data Set (GDS) abstraction for first-level fMRI outputs together with a lazy, plan-based workflow for group analysis. The package centers on a compact public grammar:

- gds() to open supported sources and create a plan
- subset(), derive(), align(), mask(), map_to(), reduce(), posthoc(), and write_out() to describe analysis steps
- compute() to execute the plan and return a realised GDS

Core capabilities include:

- Constructors for GDS objects and spaces
- Storage adapters (tabular, NIfTI, HDF5, fmristore)
- Statistical derivations and variance propagation helpers
- Reducer and post-hoc registries for group and meta-analytic workflows
- Provenance-aware export and inspection helpers

See the package reference and vignettes for supported workflows and release guidance.

Author(s)

Maintainer: Brad Buchsbaum <brad.buchsbaum@gmail.com>

See Also

Useful links:

- https://github.com/bbuchsbaum/fmrigds
- Report bugs at https://github.com/bbuchsbaum/fmrigds/issues

---

add_op *Append an operation to a plan*

Description

Append an operation to a plan

Usage

```
add_op(plan, node)
```

Arguments

plan	A gds_plan
node	Operation node (list)

Value

Updated plan

---

add_provenance_node *Append a provenance node to metadata*

Description

Append a provenance node to metadata

Usage

```
add_provenance_node(metadata, op_name, params, inputs = list())
```

Arguments

metadata	Metadata list from gds_metadata()
op_name	Operation name
params	Named list of parameters
inputs	Parent node identifiers

Value

Updated metadata list

---

align *Align subjects into a consensus space*

Description

Align subjects into a consensus space

Usage

```
align(x, family)
```

Arguments

x	A plan, source, or realised GDS
family	A MapFamily object describing per-subject transforms

Value

Updated plan

align_eager *Eagerly align subjects and compute immediately*

Description

Eagerly align subjects and compute immediately

Usage

```
align_eager(x, ...)
```

Arguments

x	Plan, source, or realized GDS
...	Arguments passed to align(), then compute()

Value

A realized GDS object

alignment-registry *Register, list, and get alignments (map families)*

Description

Syntactic sugar around the map-family registry helpers.

Usage

```
register_alignment(x, family, overwrite = FALSE)

list_alignments(x)

get_alignment(x, name)
```

Arguments

x	A gds or gds_plan object
family	A MapFamily to register
overwrite	Logical; overwrite existing family with the same name
name	Alignment family name

Value

The updated object with the family registered

A character vector of registered alignment family names

The `MapFamily` entry or `NULL` if not found

`apply_common_mask` *Apply a common mask to two realised GDS objects*

Description

Apply a common mask to two realised GDS objects

Usage

```
apply_common_mask(g1, g2, rule = c("intersection", "union"))
```

Arguments

`g1`	First GDS
`g2`	Second GDS
`rule`	Mask rule ("intersection" or "union")

Value

A list with subsetted `g1`, `g2`, and indices `idx1`, `idx2`

`as_gds` *Coerce common R objects into a GDS*

Description

Provides a lightweight front door for external packages holding data in memory (lists, arrays, data.frames) to construct a validated GDS without going through on-disk adapters.

Usage

```
as_gds(x, ...)

## S3 method for class 'list'
as_gds(
  x,
  space = NULL,
  subjects = NULL,
  contrasts = NULL,
```

```
  col_data = NULL,
  row_data = NULL,
  metadata = list(),
  ...
)

## S3 method for class 'array'
as_gds(
  x,
  space = NULL,
  subjects = NULL,
  contrasts = NULL,
  assay_name = "beta",
  col_data = NULL,
  row_data = NULL,
  metadata = list(),
  ...
)

## S3 method for class 'data.frame'
as_gds(
  x,
  mapping = NULL,
  space = NULL,
  col_data = NULL,
  row_data = NULL,
  contrast_data = NULL,
  metadata = list(),
  ...
)

## S3 method for class 'nftab'
as_gds(x, feature = NULL, ...)
```

Arguments

x	Object to coerce (list/array/data.frame)
...	Additional arguments passed to methods
space	Optional space for sample axis; defaults to sample labels from data
subjects	Optional subject identifiers
contrasts	Optional contrast identifiers
col_data	Optional subject-level covariates
row_data	Optional sample-level metadata
metadata	Optional metadata list merged into gds_metadata()
assay_name	Name to use for the single-array input (default: "beta")

mapping	Optional list mapping column names for axes and assays. Default: list(sample="sample", subject="subject", contrast="contrast", assays=list(beta="beta", var="var"), contrast_data=NULL)
contrast_data	Optional contrast-level metadata. When omitted, mapping$contrast_data can name one or more long-table columns to collapse onto the contrast axis.
feature	Name of the neurotabs feature to use as the primary assay

Value

A gds object

`as_gds.image_catalog` *Convert image_catalog to GDS plan*

Description

Create a GDS plan from an image catalog. The catalog's assay mappings determine which files are used for beta and se assays, and subject-level metadata from the catalog is attached as col_data.

Usage

```
## S3 method for class 'image_catalog'
as_gds(x, mask = NULL, ...)
```

Arguments

x	An image_catalog object with assay mappings set via `map_assays()`.
mask	Optional mask file path or NeuroVol object.
...	Additional arguments passed to `gds()`.

Value

A gds_plan object.

Examples

```
## Not run:
catalog <- image_catalog("study/", "sub-*/stats/*.nii.gz",
                         path_regex = "sub-(?<subject>[^/]+)") |>
  map_assays(beta = "cope", se = "varcope")

plan <- as_gds(catalog, mask = "mask.nii.gz")
result <- compute(plan)

## End(Not run)
```

as_gds.NeuroVec *Convert a NeuroVec to GDS*

Description

Import a neuroim2::NeuroVec (4D brain volume) as a GDS object. The 4th dimension can represent subjects OR contrasts.

Usage

```
## S3 method for class 'NeuroVec'
as_gds(
  x,
  assay_name = "beta",
  along = c("subject", "contrast"),
  subjects = NULL,
  contrasts = NULL,
  mask = NULL,
  ...
)

## S3 method for class 'DenseNeuroVec'
as_gds(
  x,
  assay_name = "beta",
  along = c("subject", "contrast"),
  subjects = NULL,
  contrasts = NULL,
  mask = NULL,
  ...
)
```

Arguments

x	A neuroim2::NeuroVec object.
assay_name	Name for the assay (default: "beta").
along	What does the 4th dimension represent? One of: • "subject": Each volume is a different subject (single contrast) • "contrast": Each volume is a different contrast (single subject)
subjects	Character vector of subject IDs. Length must match the 4th dimension when along = "subject", or length 1 when along = "contrast".
contrasts	Character vector of contrast names. Length must match the 4th dimension when along = "contrast", or length 1 when along = "subject".
mask	Optional mask. See as_gds.NeuroVol for options.
...	Additional arguments passed to new_gds().

Value

A gds object with voxel space.

Examples

```
## Not run:
# 4D volume where each volume is a subject
vec4d <- neuroim2::read_vec("all_subjects.nii.gz")
gds_obj <- as_gds(vec4d, along = "subject",
                  subjects = c("sub-01", "sub-02", "sub-03"))

# 4D volume where each volume is a contrast
gds_obj <- as_gds(vec4d, along = "contrast",
                  contrasts = c("faces", "places", "objects"))

## End(Not run)
```

---

as_gds.NeuroVol *Convert a NeuroVol to GDS*

Description

Import a single `neuroim2::NeuroVol` (3D brain volume) as a GDS object. The volume becomes a single subject with a single contrast.

Usage

```
## S3 method for class 'NeuroVol'
as_gds(
  x,
  assay_name = "beta",
  subject = "subject1",
  contrast = "contrast1",
  mask = NULL,
  ...
)

## S3 method for class 'DenseNeuroVol'
as_gds(
  x,
  assay_name = "beta",
  subject = "subject1",
  contrast = "contrast1",
  mask = NULL,
  ...
)
```

Arguments

x	A neuroim2::NeuroVol object.
assay_name	Name for the assay (default: "beta").
subject	Subject identifier (default: "subject1").
contrast	Contrast identifier (default: "contrast1").
mask	Optional mask. Can be: • NULL: use all non-zero voxels as mask • A NeuroVol: use as binary mask • A logical array: use directly as mask • "none": include all voxels (no masking)
...	Additional arguments passed to new_gds().

Value

A gds object with voxel space.

Examples

```
## Not run:
vol <- neuroim2::read_vol("beta.nii.gz")
gds_obj <- as_gds(vol, subject = "sub-01", contrast = "faces")

## End(Not run)
```

---

as_neurovec *Convert GDS assay to a 4D NeuroVec*

Description

Extract a single assay from a GDS object and return it as a neuroim2::NeuroVec (4D neuroimaging array), stacking along the 4th dimension.

Usage

```
as_neurovec(
  x,
  assay = "beta",
  along = c("subject", "contrast", "both"),
  subset_subjects = NULL,
  subset_contrasts = NULL
)
```

Arguments

x	A GDS object with a voxel space.
assay	Name of the assay to extract (default: "beta").
along	Which dimension to stack along the 4th dimension. One of: • "subject": Stack subjects (collapses contrasts if multiple). • "contrast": Stack contrasts (collapses subjects if multiple). • "both": Stack all subject x contrast combinations.
subset_subjects	Optional character vector of subject IDs to include.
subset_contrasts	Optional character vector of contrast names to include

Value

A neuroim2::NeuroVec object with dimensions [x, y, z, n] where n depends on the along parameter.

Examples

```
## Not run:
# Stack all subjects into a 4D volume
vec4d <- as_neurovec(gds, assay = "beta", along = "subject")

# Stack specific contrasts
vec4d <- as_neurovec(gds, assay = "t", along = "contrast",
                     subset_contrasts = c("faces", "places"))

## End(Not run)
```

---

as_neurovol_list *Convert GDS assay to list of NeuroVol objects*

Description

Extract a single assay from a GDS object and return it as a list of neuroim2::NeuroVol objects, one per subject/contrast combination.

Usage

```
as_neurovol_list(
  x,
  assay = "beta",
  by = c("subject", "contrast"),
  drop_dim = TRUE
)
```

Arguments

x	A GDS object with a voxel space.
assay	Name of the assay to extract (default: "beta").
by	How to organize the output list. One of: • "subject": list indexed by subject, each element is a NeuroVol (only works when there's a single contrast) • "contrast": list indexed by contrast, each element is a NeuroVol (only works when there's a single subject)
drop_dim	If TRUE (default), drop singleton dimensions when extracting. If FALSE, always return a nested list structure.

Value

A named list of neuroim2::NeuroVol objects. The structure depends on the by parameter and the dimensions of the data:

- Single contrast, by = "subject": list of NeuroVol named by subject
- Single subject, by = "contrast": list of NeuroVol named by contrast
- Multiple of both: nested list [[subject]][[contrast]]

Examples

```
## Not run:
# Extract beta maps as list of NeuroVols (one per subject)
vols <- as_neurovol_list(gds, assay = "beta")
vols$sub01  # NeuroVol for subject "sub01"

# Extract by contrast
vols_by_con <- as_neurovol_list(gds, assay = "t", by = "contrast")

## End(Not run)
```

---

as_plan *Ensure an object is a plan*

Description

Ensure an object is a plan

Usage

```
as_plan(x)
```

Arguments

x	Plan, source, or realized GDS

Value

A plan object

---

assay *Extract a single assay from a GDS object*

Description

Extract a single assay from a GDS object

Usage

```
assay(x, name = "beta", ...)
```

Arguments

x	A GDS object
name	Assay name (default: "beta")
...	Additional arguments

Value

A 3D array

---

assay_info *Retrieve assay metadata*

Description

Retrieve assay metadata

Usage

```
assay_info(name)
```

Arguments

name	Assay name

Value

Assay metadata list or NULL if not found

assays *Extract assays from a GDS object*

Description

Extract assays from a GDS object

Usage

```
assays(x)
```

Arguments

x	A GDS object

Value

Named list of 3D arrays

assert_compatible_spaces *Assert compatibility between two spaces*

Description

Assert compatibility between two spaces

Usage

```
assert_compatible_spaces(g1, g2, fields = c("type", "dim", "template_id"))
```

Arguments

g1	First GDS or space
g2	Second GDS or space
fields	Fields to compare (subset of c("type","dim","template_id"))

Value

TRUE if compatible; otherwise errors

Examples

```
sp1 <- space_voxel(c(2,2,2), diag(4), storage = "dense")
sp2 <- space_voxel(c(2,2,2), diag(4), storage = "dense")
assert_compatible_spaces(sp1, sp2)
```

`assign_meta` *Assign metadata via regex pattern matching*

Description

Apply regex patterns to filenames or paths to extract values for new metadata columns.

Usage

```
assign_meta(
  catalog,
  column,
  pattern,
  source = "basename",
  replacement = "\\1"
)
```

Arguments

`catalog`	An image_catalog object.
`column`	Name of new column to create.
`pattern`	Regex pattern to match (should contain a capture group).
`source`	Which existing column to match against. Default `"basename"`.
`replacement`	Replacement string using backreferences. Default `"\\1"`.

Value

Updated image_catalog.

Examples

```
## Not run:
catalog <- assign_meta(catalog, "run", "run-([0-9]+)", source = "basename")
catalog <- assign_meta(catalog, "contrast", "([^_]+)\\.nii", replacement = "\\1")

## End(Not run)
```

attach_weight *Attach a custom weight array to a GDS*

Description

Attach a custom weight array to a GDS

Usage

```
attach_weight(g, name, array)
```

Arguments

g	A realised GDS
name	Assay-like name for the weights (e.g., "n_eff" or "w_custom")
array	Numeric 3D array matching `sample x subject x contrast`

Value

Updated GDS with weights stored in assays

Examples

```
beta <- array(0, dim = c(2,2,1)); var <- array(1, dim = c(2,2,1))
g <- new_gds(list(beta = beta, var = var), space_sample_labels(c("a","b")), c("s1","s2"), "c1")
w <- array(1, dim = c(2,2,1))
g2 <- attach_weight(g, "w_custom", w)
```

can_map_linear *Test whether an assay can be linearly mapped*

Description

Test whether an assay can be linearly mapped

Usage

```
can_map_linear(name)
```

Arguments

name	Assay name

Value

Logical, TRUE if assay can be linearly mapped

canonicalize_node *Canonicalize an operation node*

Description

Canonicalize an operation node

Usage

```
canonicalize_node(node)
```

Arguments

node	Operation node list

Value

Canonicalized node list

coef_array *Stack coefficient assays into an array*

Description

Returns a samples x terms x contrasts array by stacking per-term coefficient assays (e.g., coef:(Intercept), coef:age, ...) along a new terms dimension.

Usage

```
coef_array(gds, prefix = "coef:")
```

Arguments

gds	A realised GDS
prefix	Assay name prefix to stack (default: "coef:")

Value

A 3D numeric array `samples x terms x contrasts`

coef_cov_tri *Retrieve packed covariance triangles for OLS coefficients*

Description

Retrieve packed covariance triangles for OLS coefficients

Usage

```
coef_cov_tri(gds, method = "ols:voxelwise", contrast)
```

Arguments

gds	A realised GDS
method	Reducer method key (default: "ols:voxelwise")
contrast	Contrast name to retrieve

Value

A list with fields type, terms, pack, and cov_tri (L x samples)

col_data *Extract column (subject) metadata from a GDS object*

Description

Extract column (subject) metadata from a GDS object

Usage

```
col_data(x)
```

Arguments

x	A GDS object

Value

Data frame with subject-level metadata

common_mask *Compute a common mask between two spaces*

Description

Compute a common mask between two spaces

Usage

```
common_mask(g1, g2, rule = c("intersection", "union"))
```

Arguments

g1	First GDS or space
g2	Second GDS or space
rule	Either "intersection" or "union"

Value

A list with integer indices `idx1` and `idx2` for subsetting each

Examples

```
m1 <- array(c(TRUE, FALSE, TRUE, TRUE), c(2, 2, 1))
m2 <- array(c(TRUE, TRUE, FALSE, TRUE), c(2, 2, 1))
sp1 <- space_voxel(c(2,2,1), diag(4), mask_bitmap = m1,
  storage = "packed")
sp2 <- space_voxel(c(2,2,1), diag(4), mask_bitmap = m2,
  storage = "packed")
common_mask(sp1, sp2, rule = "intersection")
```

compute *Materialise a plan into a realised GDS*

Description

Materialise a plan into a realised GDS

Usage

```
compute(
  x,
  sink = c("memory", "h5"),
  path = NULL,
  sink_options = list(),
  block = NULL,
  assays = NULL,
  ...
)
```

Arguments

x	Plan or source
sink	Storage sink ("memory" for in-memory data, "h5" to write a GDS HDF5 store)
path	Output path when sink = "h5"
sink_options	Optional list of sink-specific options
block	Optional list specifying block indices to request from the adapter
assays	Optional character vector to return raw arrays instead of GDS object
...	Reserved for future use

Value

A realised gds object

---

contrast_data *Extract contrast-level metadata from a GDS object*

Description

Extract contrast-level metadata from a GDS object

Usage

```
contrast_data(x)
```

Arguments

x	A GDS object or plan

Value

Data frame with one row per contrast/repeated-measure level

contrasts *Extract contrast identifiers from a GDS object*

Description

Extract contrast identifiers from a GDS object

Usage

```
contrasts(x)
```

Arguments

x	A GDS object

Value

Character vector of contrast names

derive *Record a derivation operation in a plan*

Description

Record a derivation operation in a plan

Usage

```
derive(x, what = c("var", "se", "t", "z"), options = list())
```

Arguments

x	Plan or source
what	Character vector of assays to derive
options	Optional named list of options

Value

Updated plan

derive_eager *Eagerly derive statistics and compute immediately*

Description

Eagerly derive statistics and compute immediately

Usage

```
derive_eager(...)
```

Arguments

...	Arguments passed to derive(), then compute()

Value

A realized GDS object

detect_adapter *Detect the best adapter for a source*

Description

Detect the best adapter for a source

Usage

```
detect_adapter(source, prefer = NULL)
```

Arguments

source	Source specification (path, list, etc.)
prefer	Optional adapter name to prefer when available

Value

Adapter name

digest_plan *Compute a stable digest for a plan*

Description

Compute a stable digest for a plan

Usage

```
digest_plan(plan)
```

Arguments

plan	A gds_plan object

Value

Character digest hash

explain *Explain objects and plans*

Description

Human-readable summary of a realised GDS or a lazy gds_plan.

Usage

```
explain(x)

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

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

Arguments

x	A gds or gds_plan
...	Additional arguments ignored.

Value

Invisibly returns x (and prints a summary)

explain_plan *Explain a plan's operations in a tidy table*

Description

Explain a plan's operations in a tidy table

Usage

```
explain_plan(plan)
```

Arguments

plan	A gds_plan

Value

A data.frame with one row per node (printed and returned invisibly)

extract_group *Extract NeuroVol list for a specific group*

Description

Convenience function that combines split.gds() and as_neurovol_list() to extract neuroimaging volumes for subjects in a specific group.

Usage

```
extract_group(x, group_var, group_level, assay = "beta", ...)
```

Arguments

x	A GDS object.
group_var	Column name in col_data(x) containing group labels.
group_level	The specific group level to extract.
assay	Name of the assay to extract (default: "beta").
...	Additional arguments passed to as_neurovol_list().

Value

A named list of neuroim2::NeuroVol objects for subjects in the specified group.

Examples

```
## Not run:
# Get beta maps for all patients
patient_vols <- extract_group(gds, "diagnosis", "patient", assay = "beta")

# Get t-maps for controls
control_ts <- extract_group(gds, "diagnosis", "control", assay = "t")

## End(Not run)
```

`find_maps` *Find NIfTI map files with optional subject/contrast hooks*

Description

Convenience helper to discover NIfTI map files on disk using a flexible regular-expression pattern. This is intended for quick "scooping up" of first-level maps spread across a directory tree (e.g. `.*/maps/AUC.nii`).

Usage

```
find_maps(
  root,
  pattern = "\\.nii(\\.gz)?$",
  recursive = TRUE,
  subject_fun = NULL,
  contrast_fun = NULL
)
```

Arguments

`root`	Root directory to search.
`pattern`	Regular expression passed to `base::list.files()` to match candidate NIfTI files (e.g. `"AUC\\\\.nii(\\\\.gz)?$"`).
`recursive`	Logical; whether to search subdirectories (default `TRUE`).
`subject_fun`	Optional function applied to the character vector of file paths, expected to return a character vector of subject identifiers.
`contrast_fun`	Optional function applied to the character vector of file paths, expected to return a character vector of contrast labels.

Details

The function does not impose any file naming convention. Instead, callers can provide optional `subject_fun` and `contrast_fun` hooks to derive subject and contrast labels from file paths.

Value

A data.frame with columns `file`, `subject`, and `contrast`.

gds *Create a Group Data Set (GDS)*

Description

Create a Group Data Set (GDS)

Adapter front door for building a plan

Usage

```
new_gds(
  assays,
  space,
  subjects,
  contrasts,
  col_data = NULL,
  row_data = NULL,
  metadata = list()
)

gds(source, format = c("auto", ls(.adapter_registry)), prefer = NULL, ...)
```

Arguments

assays	Named list of 3-D arrays (sample x subject x contrast)
space	A space object describing the sample axis
subjects	Character vector of subject identifiers
contrasts	Character vector of contrast identifiers
col_data	Optional data frame keyed by subjects
row_data	Optional data frame keyed by samples
metadata	Optional list merged with gds_metadata()
source	Source specification (paths, etc.)
format	Optional adapter name or "auto"
prefer	Optional preferred adapter when multiple match
...	Adapter-specific arguments forwarded to open()/probe(). Common metadata arguments include col_data, row_data, and contrast_data. For stacked repeated-measures tabular data, contrast_data_cols can name one or more columns to collapse into contrast-level metadata during ingestion.

Value

An object of class c("gds", "group_data")

A gds_plan

gds_from_neurovol_nested *Create GDS from nested structure of NeuroVol objects*

Description

The most flexible import function for creating GDS objects from neuroim2 data. Supports multiple subjects and multiple contrasts, with proper variance/SE handling.

Usage

```
gds_from_neurovol_nested(
  beta,
  var = NULL,
  se = NULL,
  contrasts = NULL,
  mask = NULL,
  col_data = NULL,
  metadata = list(),
  ...
)
```

Arguments

beta	A nested list structure for beta/effect estimates. Can be: • Nested list: `list(subject1 = list(con1 = vol, con2 = vol), ...)` • List of 4D NeuroVec: `list(subject1 = vec4d, subject2 = vec4d, ...)` where each NeuroVec has contrasts along the 4th dimension
var	Optional matching structure for variance estimates.
se	Optional matching structure for standard errors (alternative to var).
contrasts	Character vector of contrast names. Required when using NeuroVec inputs; inferred from inner list names otherwise.
mask	Optional mask. See `as_gds.NeuroVol` for options.
col_data	Optional data frame of subject-level covariates.
metadata	Optional list of additional metadata to attach.
...	Additional arguments passed to `new_gds()`.

Value

A gds object with dimensions `samples x subjects x contrasts`.

Examples

```
## Not run:
# Example 1: Nested list structure (subjects x contrasts)
beta <- list(
  `sub-01` = list(
    faces = neuroim2::read_vol("sub-01_faces_beta.nii.gz"),
    places = neuroim2::read_vol("sub-01_places_beta.nii.gz")
  ),
  `sub-02` = list(
    faces = neuroim2::read_vol("sub-02_faces_beta.nii.gz"),
    places = neuroim2::read_vol("sub-02_places_beta.nii.gz")
  )
)
var <- list(
  `sub-01` = list(
    faces = neuroim2::read_vol("sub-01_faces_var.nii.gz"),
    places = neuroim2::read_vol("sub-01_places_var.nii.gz")
  ),
  `sub-02` = list(
    faces = neuroim2::read_vol("sub-02_faces_var.nii.gz"),
    places = neuroim2::read_vol("sub-02_places_var.nii.gz")
  )
)

gds_obj <- gds_from_neurovol_nested(beta, var = var)

# Example 2: List of 4D NeuroVecs (each subject has multi-contrast 4D file)
beta_vecs <- list(
  `sub-01` = neuroim2::read_vec("sub-01_allcons_beta.nii.gz"),
  `sub-02` = neuroim2::read_vec("sub-02_allcons_beta.nii.gz")
)
gds_obj <- gds_from_neurovol_nested(beta_vecs,
                                    contrasts = c("faces", "places", "objects"))

## End(Not run)
```

`gds_from_neurovols` *Create GDS from lists of NeuroVol objects*

Description

Import matched lists of NeuroVol objects (e.g., beta and variance maps) into a GDS. This handles the common case of multiple subjects with a single contrast. For multiple contrasts, use `gds_from_neurovol_nested()`.

Usage

```
gds_from_neurovols(
  beta,
```

```
  var = NULL,
  se = NULL,
  contrast = "contrast1",
  mask = NULL,
  col_data = NULL,
  ...
)
```

Arguments

beta	A named list of NeuroVol objects for beta/effect estimates. Names become subject IDs.
var	Optional named list of NeuroVol objects for variance estimates. Must have same names as beta.
se	Optional named list of NeuroVol objects for standard error. Must have same names as beta. Provide either var or se, not both.
contrast	Contrast name (default: "contrast1").
mask	Optional mask. See as_gds.NeuroVol for options.
col_data	Optional data frame of subject-level covariates. Row names must match subject IDs (names of beta).
...	Additional arguments passed to new_gds().

Value

A gds object.

Examples

```
## Not run:
# Load beta and variance maps for each subject
beta_vols <- list(
  `sub-01` = neuroim2::read_vol("sub-01_beta.nii.gz"),
  `sub-02` = neuroim2::read_vol("sub-02_beta.nii.gz")
)
var_vols <- list(
  `sub-01` = neuroim2::read_vol("sub-01_var.nii.gz"),
  `sub-02` = neuroim2::read_vol("sub-02_var.nii.gz")
)

gds_obj <- gds_from_neurovols(beta_vols, var = var_vols)

# With subject covariates
covars <- data.frame(age = c(25, 30), group = c("A", "B"),
                     row.names = c("sub-01", "sub-02"))
gds_obj <- gds_from_neurovols(beta_vols, var = var_vols, col_data = covars)

## End(Not run)
```

gds_from_nifti_maps *Construct a GDS from NIfTI maps discovered on disk*

Description

This helper builds on the NIfTI adapter to materialise a realised GDS from a set of NIfTI map files, typically obtained via `find_maps()`. The heavy lifting (reading NIfTI, spatial metadata, masking) is delegated to the existing NIfTI adapter and the `neuroim2` package.

Usage

```
gds_from_nifti_maps(maps, mask = NULL, ...)
```

Arguments

`maps`	A data.frame (or tibble) with at least a `file` column of NIfTI paths. Optional `subject` and `contrast` columns can be used to relabel the realised GDS.
`mask`	Optional mask passed through to `gds()` when constructing the NIfTI plan. Can be a NIfTI file path or a `neuroim2::NeuroVol`, matching the NIfTI adapter contract.
`...`	Additional arguments forwarded to `gds()` when building the NIfTI-backed plan.

Details

Subject and contrast labels can be injected post-hoc using the `subject` and `contrast` columns when available in `maps`. This avoids hard-coding any filename convention while still allowing callers to impose semantics via hooks.

Beta-only raw maps are accepted. When no variance or standard-error maps are available, the realised GDS contains a synthetic `var` assay filled with 1 and a warning is emitted. This placeholder variance is useful for satisfying the GDS assay contract, but it is not meaningful subject-level uncertainty. For group analyses over subject-level searchlight or stat-map metrics, prefer an unweighted reducer such as `method = "ols:voxelwise"` instead of fixed/random-effects meta-analysis.

Value

A realised GDS object.

Examples

```
## Not run:
maps <- data.frame(
  file = c("1001_era_partition/maps/naive_diag_mean.nii.gz",
           "1002_era_partition/maps/naive_diag_mean.nii.gz"),
  subject = c("1001", "1002"),
  contrast = "naive_diag_mean"
)
```

```
group_info <- data.frame(
  group = c("control", "patient"),
  row.names = maps$subject
)

g <- gds_from_nifti_maps(maps) |>
  with_col_data(group_info)

fit <- reduce(g, method = "ols:voxelwise", formula = ~ group) |>
  compute()

## End(Not run)
```

`gds_from_scalar_maps` *Create a GDS from subject-level scalar NIfTI maps*

Description

`gds_from_scalar_maps()` is a semantic wrapper for beta-only/stat-map workflows: each NIfTI is a subject-level metric map rather than a first-level beta image with meaningful uncertainty. The resulting GDS uses the NIfTI adapter's unit-variance placeholder behavior for beta-only sources.

Usage

```
gds_from_scalar_maps(
  files,
  subject = NULL,
  contrast = "map",
  col_data = NULL,
  mask = NULL,
  ...
)

as_scalar_map_gds(
  files,
  subject = NULL,
  contrast = "map",
  col_data = NULL,
  mask = NULL,
  ...
)
```

Arguments

`files`	Character vector of NIfTI paths, a data.frame with a `file` column, or an `image_catalog()`.
`subject`	Optional subject identifiers. Required for character `files` unless `files` is named; ignored when `files` already has a `subject` column.

contrast	Contrast label(s). A scalar value is recycled across files. Ignored when files already has a contrast column unless explicitly supplied.
col_data	Optional subject-level covariates. If a subject column is present and row names are absent, it is used as the row key.
mask	Optional mask passed to gds_from_nifti_maps().
...	Additional arguments forwarded to gds_from_nifti_maps().

Value

A realised gds object.

Examples

```
## Not run:
g <- gds_from_scalar_maps(
  files = c("sub-01/maps/auc.nii.gz", "sub-02/maps/auc.nii.gz"),
  subject = c("sub-01", "sub-02"),
  contrast = "auc",
  col_data = participants
)
fit <- group_ols(g, ~ group) |> compute()

## End(Not run)
```

---

gds_metadata *Construct default metadata for a GDS object*

Description

Construct default metadata for a GDS object

Usage

```
gds_metadata(
  schema_version = "0.1.0",
  units = list(),
  provenance = list(graph = list(), log = character(), digest = NULL),
  software = list(package = "fmrigds", version = .pkg_version(), R_version =
    getRversion()),
  alignment = NULL,
  map_families = list(),
  mask_info = NULL,
  contrast_info = NULL,
  design_mats = NULL,
  notes = NULL,
  created = Sys.time()
)
```

Arguments

schema_version	Schema version string
units	Named list of assay units
provenance	Provenance structure (graph, log, digest)
software	Software metadata (package name, version, R version)
alignment	Optional alignment metadata
map_families	Named list of registered map families
mask_info	Optional mask metadata
contrast_info	Optional contrast metadata
design_mats	Optional design matrices metadata
notes	Optional user notes
created	Timestamp for when the metadata was created

Value

Metadata list

---

gds_plan *Construct a lazy GDS plan*

Description

Construct a lazy GDS plan

Usage

```
gds_plan(source, nodes = list(), meta = list())
```

Arguments

source	A gds_source object
nodes	List of operation nodes
meta	Metadata list (planner bookkeeping)

Value

A plan object (gds_plan)

gds_source *Define a plan source (adapter binding)*

Description

Define a plan source (adapter binding)

Usage

```
gds_source(adapter, source_spec, probe_result = NULL)
```

Arguments

adapter	Adapter identifier string
source_spec	Source specification (paths, handles, etc.)
probe_result	Optional probe metadata

Value

Source descriptor

gds_to_tibble *Tidy long-form export for GDS*

Description

Public wrapper around the internal tidy frame builder used by CSV/Parquet exporters. Returns a data.frame; if you prefer a tibble, wrap with `tibble::as_tibble()`.

Usage

```
gds_to_tibble(gds, assays = NULL, drop_na = FALSE, include_col_data = FALSE)
```

Arguments

gds	A realised GDS object
assays	Character vector of assay names to include (default: all assays in the GDS)
drop_na	If TRUE, drop rows with all-NA assays
include_col_data	If TRUE, join columns from `col_data`

Value

A data.frame in long form with columns `sample`, `subject`, `contrast`, plus one column per requested assay.

get_adapter *Retrieve a registered adapter*

Description

Retrieve a registered adapter

Usage

```
get_adapter(name)
```

Arguments

name	Adapter name

Value

Adapter list

get_map_family *Retrieve a registered map family by name*

Description

Retrieve a registered map family by name

Usage

```
get_map_family(x, name)
```

Arguments

x	A plan or realised GDS
name	Map family name

Value

A MapFamily object

get_posthoc *Get post-hoc method by name*

Description

Get post-hoc method by name

Usage

```
get_posthoc(name)
```

Arguments

name	Method identifier (e.g., "fdr:bh")

Value

A list with fields name, fun, requires, provides, or NULL if not found

get_reducer *Get reducer by name*

Description

Get reducer by name

Usage

```
get_reducer(name)
```

Arguments

name	Reducer name

Value

Reducer list or NULL if not found

group_ols *Voxelwise OLS helpers for scalar-map group analyses*

Description

These helpers wrap `reduce(method = "ols:voxelwise")` for subject-level map workflows. They return a lazy GDS plan; call `compute()` to materialise the fitted maps.

Usage

```
group_ols(x, formula = ~1, col_data = NULL, options = list(), ...)

one_sample(x, subset = NULL, col_data = NULL, options = list(), ...)

two_sample(
  x,
  group = "group",
  baseline = NULL,
  level = NULL,
  col_data = NULL,
  options = list(),
  ...
)
```

Arguments

`x`	A realised `gds` or `gds_plan`.
`formula`	One-sided model formula for `group_ols()`.
`col_data`	Optional subject-level covariates to attach before fitting.
`options`	Reducer options passed to `reduce()`.
`...`	Additional arguments passed to `reduce()`.
`subset`	Optional expression evaluated in `col_data(x)` to select subjects before fitting the one-sample model.
`group`	Name of the grouping column in `col_data(x)`.
`baseline`	Optional reference level for the grouping factor.
`level`	Optional non-baseline level to compare against `baseline`. When omitted, all non-baseline levels are retained in the model.

Value

A `gds_plan` with an OLS reduce node.

Examples

```
## Not run:
fit <- group_ols(g, ~ group) |> compute()
one <- one_sample(g) |> compute()
two <- two_sample(g, group = "diagnosis", baseline = "control") |> compute()

## End(Not run)
```

---

harmonise_contrasts *Harmonise contrast names in a GDS*

Description

Harmonise contrast names in a GDS

Usage

```
harmonise_contrasts(g, map)
```

Arguments

g	A realised GDS
map	Named character vector mapping old -> new contrast names

Value

Updated GDS with renamed contrasts

Examples

```
beta <- array(0, dim = c(2,1,2), dimnames = list(NULL, "s1", c("a","b")))
var <- beta
g <- new_gds(list(beta = beta, var = var), space_sample_labels(c("x","y")), "s1", c("a","b"))
harmonise_contrasts(g, c(a = "A", b = "B"))
```

---

image_catalog *Discover image files and create a catalog*

Description

Sweeps a directory tree for files matching a glob pattern and extracts metadata from file paths using regex capture groups.

Usage

```
image_catalog(root, pattern = "**/*.nii*", path_regex = NULL, recursive = TRUE)
```

Arguments

root	Root directory to search.
pattern	Glob pattern for file discovery (e.g., "sub-*/analysis/*.nii.gz"). Supports * for single directory level and ** for recursive matching.
path_regex	Optional regex with named capture groups for metadata extraction. Named groups like (?<subject>[^/]+) become metadata columns.
recursive	Whether to search subdirectories. Default TRUE. Only used if pattern does not contain **.

Value

An image_catalog object.

Examples

```
## Not run:
# Discover files with subject extraction
catalog <- image_catalog(
  root = "study",
  pattern = "sub-*/analysis/*.nii.gz",
  path_regex = "sub-(?<subject>[^/]+)"
)

## End(Not run)
```

---

join_meta *Join external metadata to catalog*

Description

Merge a data frame with the catalog metadata by a key column.

Usage

```
join_meta(catalog, data, by, suffix = ".external")
```

Arguments

catalog	An image_catalog object.
data	External data.frame to join.
by	Column name(s) for joining (must exist in both catalog$metadata and data).
suffix	Suffix for duplicate column names from data. Default ".external".

Value

Updated image_catalog.

Examples

```
## Not run:
demographics <- data.frame(subject = c("01", "02"), age = c(25, 30))
catalog <- join_meta(catalog, demographics, by = "subject")

## End(Not run)
```

`list_map_families` *List registered map families*

Description

List registered map families

Usage

```
list_map_families(x)
```

Arguments

`x`	A plan or realised GDS

Value

Character vector of family names

`list_posthoc` *List registered post-hoc methods*

Description

List registered post-hoc methods

Usage

```
list_posthoc()
```

`list_reducers` *List registered reducers*

Description

List registered reducers

Usage

```
list_reducers()
```

load_plan *Load a plan from JSON*

Description

Load a plan from JSON

Usage

```
load_plan(file)
```

Arguments

file	JSON file produced by save_plan

Value

A gds_plan

make_linear_family *Convenience helpers to create and register alignments*

Description

Convenience helpers to create and register alignments

Usage

```
make_linear_family(
  name,
  from,
  to,
  operators_by_subject,
  uncertainty = UncertaintyRule("independent")
)
```

Arguments

name	Family name
from	Source space
to	Target space
operators_by_subject	Named list of per-subject linear operators (matrices)
uncertainty	Uncertainty rule for variance propagation

Value

A MapFamily

Examples

```
from <- space_parcels(c("R1","R2"))
to   <- space_parcels(c("G1","G2"))
ops <- list(s1 = diag(2), s2 = diag(2))
fam <- make_linear_family("lin", from, to, ops)
```

`make_warp_family` *Create a warp-based alignment family from on-disk paths*

Description

Create a warp-based alignment family from on-disk paths

Usage

```
make_warp_family(
  name,
  from,
  to,
  warp_paths,
  loader,
  uncertainty = UncertaintyRule("independent")
)
```

Arguments

`name`	Family name
`from`	Source space
`to`	Target space
`warp_paths`	Named character vector of file paths per subject
`loader`	Function(path) -> operator (e.g., matrix) for a single subject
`uncertainty`	Uncertainty rule

Value

A MapFamily (type = deform3d)

Examples

```
loader <- function(path) as.matrix(utils::read.csv(path, header = FALSE))
# paths <- c(s1 = "warp_s1.csv", s2 = "warp_s2.csv")
# fam <- make_warp_family("warp", from, to, paths, loader)
```

`map_assays` *Map files to assay types*

Description

Specify which files correspond to which statistical assays (beta, se, var, t). This mapping is used by `as_gds.image_catalog()` to construct the GDS.

Usage

```
map_assays(catalog, ...)
```

Arguments

`catalog`	An image_catalog object.
`...`	Named arguments where names are assay types and values are either: • A regex pattern to match against filenames, OR • A named list like `list(column = value)` for exact column matching

Value

Updated image_catalog with assay_map.

Examples

```
## Not run:
# By filename pattern
catalog <- map_assays(catalog,
  beta = "cope[0-9]+\\.nii",
  se = "varcope[0-9]+\\.nii"
)

# By metadata column value
catalog <- map_assays(catalog,
  beta = list(stat_type = "cope"),
  se = list(stat_type = "varcope")
)

## End(Not run)
```

map_linear *Create a linear map between spaces*

Description

Create a linear map between spaces

Usage

```
map_linear(
  from_space,
  to_space,
  operator,
  by_subject = NULL,
  traits = list(orthogonal = FALSE, mass_preserving = FALSE),
  uncertainty = UncertaintyRule("independent"),
  name = NULL
)
```

Arguments

from_space	Source space descriptor or name
to_space	Target space descriptor or name
operator	Mapping operator (matrix/function)
by_subject	Optional named list of subject-specific operator descriptors/functions
traits	List of map traits (orthogonal, mass_preserving)
uncertainty	UncertaintyRule controlling variance propagation
name	Optional map name

Value

Object of class c("map_linear", "gds_map")

map_to *Add a space transformation to a plan*

Description

Add a space transformation to a plan

Usage

```
map_to(
  x,
  target_space,
  map,
  uncertainty = UncertaintyRule("independent"),
  combine = NULL
)
```

Arguments

x	Plan, source, or realised GDS
target_space	Target space descriptor or identifier
map	Mapping object (map_linear or matrix)
uncertainty	UncertaintyRule instance controlling variance propagation
combine	Optional evidence combiner when effect scale absent

Value

Updated plan

---

map_to_eager *Eagerly apply space transformation and compute immediately*

Description

Eagerly apply space transformation and compute immediately

Usage

```
map_to_eager(x, ...)
```

Arguments

x	Plan, source, or realized GDS
...	Arguments passed to map_to(), then compute()

Value

A realized GDS object

MapFamily *Create a subject-aware map family*

Description

Create a subject-aware map family

Usage

```
MapFamily(
  name,
  from_space,
  to_space,
  type = c("linear", "orthogonal", "affine3d", "deform3d", "ot"),
  by_subject,
  traits = list(orthogonal = FALSE, mass_preserving = FALSE),
  uncertainty = UncertaintyRule("independent")
)
```

Arguments

name	Family name
from_space	Source space
to_space	Target space
type	Map type identifier
by_subject	Named list of per-subject operator descriptors
traits	Map traits
uncertainty	UncertaintyRule

Value

Object of class c("gds_map_family", "gds_map")

mask *Apply a mask policy lazily*

Description

Apply a mask policy lazily

Usage

```
mask(x, policy = MaskPolicy())
```

Arguments

x	Plan, source, or realised GDS
policy	Mask policy created by MaskPolicy()

Value

Updated plan

---

mask_eager *Eagerly apply mask policy and compute immediately*

Description

Eagerly apply mask policy and compute immediately

Usage

```
mask_eager(x, ...)
```

Arguments

x	Plan, source, or realized GDS
...	Arguments passed to mask(), then compute()

Value

A realized GDS object

MaskPolicy *Define a mask policy*

Description

Define a mask policy

Usage

```
MaskPolicy(
  scope = c("group", "subject"),
  rule = c("intersection", "union", "threshold", "custom"),
  threshold = 0.95,
  custom = NULL
)
```

Arguments

scope	Scope of mask computation ("group" or "subject")
rule	Mask rule ("intersection", "union", "threshold", "custom")
threshold	Fraction for threshold rule
custom	Custom function returning logical mask

Value

A mask policy object

---

metadata *Extract metadata from a GDS object*

Description

Extract metadata from a GDS object

Usage

```
metadata(x)
```

Arguments

x	A GDS object

Value

Metadata list

---

model_matrix *Build a design matrix from attached col_data*

Description

Convenience helper to construct a model matrix using subject-level covariates already attached to a plan via with_col_data() or stored in a realised GDS.

Usage

```
model_matrix(x, formula)
```

Arguments

x	A gds or gds_plan (or an object coercible via as_plan())
formula	Model formula (or string convertible via stats::as.formula)

Value

A numeric model matrix

`new_image_catalog` *Create a new image catalog object*

Description

Low-level constructor for image_catalog objects. Users should typically use `image_catalog()` for file discovery instead.

Usage

```
new_image_catalog(
  files,
  metadata = NULL,
  root_dir = NULL,
  pattern = NULL,
  assay_map = list()
)
```

Arguments

`files`	Character vector of absolute file paths.
`metadata`	Data frame with one row per file containing extracted path components and user-assigned metadata.
`root_dir`	Root directory used for discovery (for relative path display).
`pattern`	Original glob pattern used for discovery.
`assay_map`	Named list mapping assay names to file selection criteria.

Value

An object of class `c("image_catalog", "list")`.

nifti_source *Construct a NIfTI source specification*

Description

Build an explicit NIfTI source list for use with `gds(..., format = "nifti")`. If `beta` is provided without `se`, materialising the GDS creates a unit variance placeholder with a warning. This supports raw beta/stat-map workflows; the synthetic variance should not be interpreted as subject-level uncertainty.

Usage

```
nifti_source(beta = NULL, se = NULL)
```

Arguments

`beta`	Character vector of NIfTI paths or a directory containing beta/effect images (optional, can be `NULL` if only `se` is provided)
`se`	Character vector of NIfTI paths or a directory containing standard error images (optional, can be `NULL` if only `beta` is provided)

Value

A list suitable for `gds(source = <returned>, format = "nifti")`

Examples

```
src <- nifti_source(beta = c("sub-01_beta.nii.gz", "sub-02_beta.nii.gz"),
                    se   = c("sub-01_se.nii.gz",   "sub-02_se.nii.gz"))
str(src)
```

op_align_to_group *Create an align-to-group operation node*

Description

Create an align-to-group operation node

Usage

```
op_align_to_group(family = NULL, family_name = NULL)
```

Arguments

`family`	MapFamily object for alignment
`family_name`	Name of registered map family

Value

Operation node list

`op_derive` *Create a derive operation node*

Description

Create a derive operation node

Usage

```
op_derive(what, options = list())
```

Arguments

`what`	Character vector of statistics to derive (e.g., "t", "z", "p")
`options`	Optional list of derivation options

Value

Operation node list

`op_map` *Create a map operation node*

Description

Create a map operation node

Usage

```
op_map(target_space, map, uncertainty, combine = NULL)
```

Arguments

`target_space`	Target space descriptor
`map`	Mapping operator (matrix or function)
`uncertainty`	UncertaintyRule for variance propagation
`combine`	Optional combine method

Value

Operation node list

op_mask_policy *Create a mask policy operation node*

Description

Create a mask policy operation node

Usage

```
op_mask_policy(policy)
```

Arguments

policy	MaskPolicy object

Value

Operation node list

op_reduce *Create a reduce operation node*

Description

Create a reduce operation node

Usage

```
op_reduce(method, weights, by, options = list(), formula = NULL)
```

Arguments

method	Reduction method (e.g., "fixed", "random")
weights	Optional weight vector or method
by	Optional grouping variable
options	Optional reduction options
formula	Optional model formula for meta-regression

Value

Operation node list

op_subset_axis *Create a subset operation node*

Description

Create a subset operation node

Usage

```
op_subset_axis(sample = NULL, subject = NULL, contrast = NULL)
```

Arguments

sample	Sample indices or labels
subject	Subject indices or IDs
contrast	Contrast indices or names

Value

Operation node list

op_write *Create a write operation node*

Description

Create a write operation node

Usage

```
op_write(path, format, options = list())
```

Arguments

path	Output file path
format	Output format (e.g., "h5", "csv", "parquet")
options	Optional format-specific options

Value

Operation node list

OrthogonalFamily *Orthogonal map family helper*

Description

Orthogonal map family helper

Usage

```
OrthogonalFamily(
  name,
  from_space,
  to_space,
  matrices_by_subject,
  uncertainty = UncertaintyRule("independent")
)
```

Arguments

`name`	Family name
`from_space`	Source space
`to_space`	Target space
`matrices_by_subject`	Named list of per-subject orthogonal matrices
`uncertainty`	Uncertainty rule (default: independent)

Value

A MapFamily object with orthogonal traits

OTFamily *Optimal transport family helper*

Description

Optimal transport family helper

Usage

```
OTFamily(
  name,
  from_space,
  to_space,
  plans_by_subject,
  uncertainty = UncertaintyRule("independent")
)
```

Arguments

name	Family name
from_space	Source space
to_space	Target space
plans_by_subject	Named list of per-subject optimal transport plans
uncertainty	Uncertainty rule (default: independent)

Value

A MapFamily object with mass-preserving traits

---

plan *Coerce to a lazy plan (alias)*

Description

Alias for as_plan() to improve ergonomics when converting realized GDS objects or sources into a gds_plan.

Usage

```
plan(x)
```

Arguments

x	A gds, gds_source, or gds_plan

Value

A gds_plan

---

posthoc *Add a post-hoc operation to a plan*

Description

Add a post-hoc operation to a plan

Usage

```
posthoc(x, method, options = list())
```

Arguments

x	A plan or object coercible via as_plan()
method	Post-hoc method name (e.g., "fdr:bh", "fdr:by")
options	Optional list of method-specific options

Value

Updated gds_plan

Examples

```
df <- data.frame(
  sample = rep(c("ROI_1","ROI_2"), each = 2),
  subject = rep(c("s1","s2"), times = 2),
  contrast = "c1",
  beta = c(0.5,0.6,0.7,0.8),
  var = c(0.04,0.05,0.06,0.07)
)
tmp <- tempfile(fileext = ".csv"); utils::write.csv(df, tmp, row.names = FALSE)
plan <- gds(tmp) |> reduce(method = "fixed") |> posthoc("fdr:bh")
# g <- compute(plan)  # returns q-values
```

---

preview *Preview a small block through the plan*

Description

Executes the plan with a small sample block to quickly inspect shapes or values. If assays is provided, returns raw arrays from the adapter stage (pre-operations); otherwise returns a realised GDS of the small block after applying plan nodes.

Usage

```
preview(plan, n = 3, assays = NULL)
```

Arguments

plan	A gds_plan or object coercible via as_plan()
n	Number of samples to preview (from start)
assays	Optional assays to request as raw arrays

Value

Either a list of arrays (when assays provided) or a small gds

print.catalog_validation_report *Print validation report*

Description

Print validation report

Usage

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

Arguments

x	A catalog_validation_report object.
...	Additional arguments (ignored).

Value

Invisibly returns the report.

print.image_catalog *Print method for image_catalog*

Description

Print method for image_catalog

Usage

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

Arguments

x	An image_catalog object.
...	Additional arguments (ignored).

Value

Invisibly returns the catalog.

provenance_node *Create a provenance node*

Description

Create a provenance node

Usage

```
provenance_node(
  op_name,
  params,
  inputs = list(),
  timestamp = Sys.time(),
  software = list(package = "fmrigds", version = .pkg_version()),
  hash = NULL
)
```

Arguments

op_name	Operation name
params	Named list of parameters
inputs	Vector/list of parent hashes/ids
timestamp	Timestamp of operation
software	Software metadata
hash	Optional pre-computed hash

Value

A provenance node list

read_catalog *Read catalog from JSON file*

Description

Load an image catalog previously saved with write_catalog().

Usage

```
read_catalog(file)
```

Arguments

file	Path to catalog JSON file.

Value

An image_catalog object.

Examples

```
## Not run:
catalog <- read_catalog("my_study_manifest.json")

## End(Not run)
```

`reduce` *Reduce across subjects (meta-analysis)*

Description

Reduce across subjects (meta-analysis)

Usage

```
reduce(
  x,
  method = c("fixed", "random", "stouffer", "fisher"),
  weights = c("1/var", "n_eff", "equal", "custom"),
  by = "contrast",
  options = list(),
  formula = NULL,
  data = NULL
)
```

Arguments

`x`	A `gds_plan`, `gds_source`, or realised `gds`
`method`	Reduction method. Built-ins include `"fixed"`, `"random"`, `"stouffer"`, and `"fisher"`. Registry-backed reducers include meta-analytic regression reducers such as `"meta:fe_reg"` and the restricted repeated-measures Gaussian LMM reducers `"lmm:ri"` and `"lmm:ri_slope1"`, plus permutation reducers `"perm:onesample"` and `"perm:twosample"`.
`weights`	Weighting scheme (`"1/var"`, `"n_eff"`, `"equal"`, `"custom"`)
`by`	Grouping variable (e.g., `"contrast"`)
`options`	Additional reducer options. Common examples include `tau2 = NULL`, `custom_weights`, or for restricted LMM reducers `list(fit = "REML", theta_mode = "pooled")` plus `list(slope = "time", covariance = "diag")` for `"lmm:ri_slope1"`. Note: Current grouping is per-contrast. Values other than "contrast" are not interpreted yet; future versions may extend grouping to other factors in `row_data`/`col_data`. Restricted repeated-measures LMM contract:

- `"lmm:ri"` fits a Gaussian random-intercept model.
- `"lmm:ri_slope1"` fits a Gaussian random intercept plus one within-subject slope.
- Repeated-measure metadata must live on the contrast axis via `with_contrast_data()` or tabular ingestion with `gds(..., contrast_data_cols = ...)`.
- `options$theta_mode` can be `"pooled"` or `"voxelwise"`.
- The supported family is intentionally narrow: one grouping factor only, Gaussian responses only, and no general random-effects formula grammar.

Permutation reducer contract:

- `"perm:onesample"` performs an unweighted one-sample sign-flip t test for each sample and contrast.
- `"perm:twosample"` performs an unweighted two-sample label-permutation t test. It can infer the tested two-level group column from `formula` and `col_data`, or use `options$group` directly.
- Common options include `n_perm`, `seed`, `alternative`, and for `"perm:twosample"` `variance = "welch"` or `"pooled"`.
- Outputs include `t_g`, parametric `p_g`, permutation `p_perm`, and max-|t| family-wise `p_fwer`.

`formula` Optional model formula. For meta-regression reducers the design is built from subject-level `col_data`. For `"lmm:ri"` and `"lmm:ri_slope1"`, the formula specifies fixed effects at the subject-by-repeat level and must not contain `lmer`-style random-effects syntax.

`data` Optional data frame for building `X` when `x` is not a realised GDS.

Details

`reduce()` always works on a lazy plan internally, so `x` is whichever stage of the fmrigds workflow you already have:

- Start from files or another external source with `gds()`. That returns a `gds_plan` you can pipe directly into `reduce()`.
- Start from an in-memory result with a realised `gds` returned by `compute()` or created directly with `new_gds()`. `reduce()` will convert it with `as_plan()` for you.
- You can also pass a low-level `gds_source` created by `gds_source()`, but most users do not need this because `gds()` creates the source binding automatically.

If you want that conversion to be explicit, use `as_plan()` or its alias `plan()` before calling `reduce()`.

For worked examples, see `vignette("fmrigds")` for the basic source -> plan -> compute workflow and `vignette("as-plan-and-spatial-fdr")` for chaining verbs on realised GDS objects.

Value

Updated plan

See Also

`gds()`, `compute()`, `as_plan()`, `plan()`, `new_gds()`, `gds_source()`

reduce_eager *Eagerly reduce across subjects and compute immediately*

Description

Eagerly reduce across subjects and compute immediately

Usage

```
reduce_eager(x, ...)
```

Arguments

x	Plan, source, or realized GDS
...	Arguments passed to reduce(), then compute()

Value

A realized GDS object

reducer-lmm *Restricted repeated-measures Gaussian LMM reducers*

Description

`fmrigds` includes two specialized repeated-measures Gaussian mixed-model reducers that are optimized for the case where every sample shares the same observation layout and the same fixed/random design.

Details

- `method = "lmm:ri"` fits a random-intercept model.
- `method = "lmm:ri_slope1"` fits a random intercept plus one within-subject slope supplied through `options$slope`.

These reducers operate on the full contrast axis jointly. Repeated-measure metadata therefore needs to be attached as contrast-level metadata, either with `with_contrast_data()` or during tabular ingestion via `gds(..., contrast_data_cols = ...)`.

Supported contract:

- one grouping factor only
- Gaussian outcomes only
- shared observation layout and shared fixed/random design across samples
- `theta_mode = "pooled"` for one shared variance structure across samples

- `theta_mode = "voxelwise"` for sample-specific variance parameters
- no crossed random effects and no general `lmer()` random-effects grammar

Output assays follow the same naming conventions as the regression reducers: `coef:<term>`, `se_coef:<term>`, `t_coef:<term>`, `p_coef:<term>`, plus variance-component assays such as `sigma2`, `vc_intercept`, `vc_slope`, `lambda_intercept`, and `lambda_slope`.

Usage

```
rm_df <- data.frame(
  sample = rep(c("ROI_1", "ROI_2"), each = 12),
  subject = rep(paste0("sub-", sprintf("%02d", 1:4)), each = 3, times = 2),
  contrast = rep(c("pre", "mid", "post"), times = 8),
  time = rep(c(-1, 0, 1), times = 8),
  beta = c(
    0.40, 0.80, 1.20, 0.55, 0.95, 1.35, 0.65, 1.05, 1.45, 0.85, 1.25, 1.65,
    0.70, 1.20, 1.70, 0.85, 1.35, 1.85, 0.95, 1.45, 1.95, 1.15, 1.65, 2.15
  ),
  var = 0.04
)

g <- gds(rm_df, contrast_data_cols = "time") |>
  reduce(
    method = "lmm:ri_slope1",
    formula = ~ time,
    options = list(
      slope = "time",
      covariance = "diag",
      fit = "REML",
      theta_mode = "voxelwise"
    )
  ) |>
  compute()

names(assays(g))
assay(g, "coef:time")
```

`reducer-ols-voxelwise` *Voxelwise OLS reducer (per-sample GLM across subjects)*

Description

The reducer identified by `method = "ols:voxelwise"` performs unweighted ordinary least squares across subjects independently for each sample and contrast. It accepts a model formula via `reduce(..., formula = ~ 1 + cov)` (design is built from `with_col_data()` or `gds(..., col_data=...)`).

Details

Outputs are expanded into per-term assays to preserve the 3-axis invariant:

- `coef:<term>` and `se_coef:<term>` (one assay per model term)
- `t_coef:<term>` and `p_coef:<term>` convenience statistics
- `sigma2` (residual variance) and `df_res` (residual degrees of freedom)

Optionally, a packed triangular covariance for the coefficient vector can be attached per contrast using `options = list(return_cov = "tri")`. Retrieve it from a realised GDS via `coef_cov_tri()`.

Usage

```
# Attach subject-level covariates; build design via formula
plan <- gds("group.csv", col_data = subj_cov) |
  reduce(method = "ols:voxelwise", formula = ~ 1 + age + sex,
         options = list(return_cov = "tri"))
g <- compute(plan)

# Per-term assays
assay(g, "coef:(Intercept)"); assay(g, "coef:age"); assay(g, "coef:sex")

# Packed covariance for first contrast
cov_info <- coef_cov_tri(g, contrast = contrasts(g)[1])
cov_info$terms   # term order
cov_info$cov_tri # L x samples packed upper triangle

# Stack all coefficients into [samples x terms x contrasts]
coef_arr <- coef_array(g)
```

`register_adapter` *Register a storage adapter*

Description

Register a storage adapter

Usage

```
register_adapter(name, detect, open, probe, read, close, ...)
```

Arguments

`name`	Adapter name
`detect`	Function (source) -> score in `[0,1]` or FALSE
`open`	Function (source, ...) -> handle
`probe`	Function (handle, ...) -> list(metadata)

read	Function (handle, assays, block = NULL, ...) -> named list of arrays
close	Function (handle) -> NULL
...	Additional elements stored with adapter entry

Value

Invisibly, the adapter entry

`register_assay` *Register an assay type*

Description

Register an assay type

Usage

```
register_assay(
  name,
  role,
  units = NULL,
  variance_of = NULL,
  derive_from = NULL
)
```

Arguments

name	Assay name
role	Semantic category (e.g., "location", "variance", "stdev", "z", "t", "F", "df", "n_eff", "p", "chi2", "evidence", "log_evidence", "posterior")
units	Optional units string/list
variance_of	Optional name of the assay whose variance this represents
derive_from	Optional character vector of prerequisite assays

Value

Invisibly, the registered assay definition

register_map *Register a map family on a plan or realised GDS*

Description

Register a map family on a plan or realised GDS

Usage

```
register_map(x, family, overwrite = FALSE)
```

Arguments

x	A gds_plan or realised gds
family	A MapFamily object
overwrite	Whether to overwrite an existing family with the same name

Value

Updated object (gds_plan or gds)

register_nftab_adapter *Register the neurotabs (NFTab) adapter*

Description

Makes nftab objects from the **neurotabs** package a first-class data source for gds(). Called automatically during .onLoad() when **neurotabs** is installed; can also be called manually.

Usage

```
register_nftab_adapter()
```

Value

Invisibly, the adapter entry.

register_posthoc *Register a post-hoc method*

Description

Register or replace a post-hoc handler identified by `name`.

Usage

```
register_posthoc(name, fun, requires, provides, overwrite = TRUE)
```

Arguments

`name`	String identifier, e.g., "fdr:bh"
`fun`	Function(arrays, opts) -> list of arrays (delta or full arrays)
`requires`	Character vector of required input assays, e.g., c("p")
`provides`	Character vector of output assay names, e.g., c("q")
`overwrite`	Logical; if FALSE and a handler with `name` exists, error

Details

The handler `fun` is called as `fun(arrays, opts)` where `arrays` is a named list of assays shaped `sample x subject x contrast`. It should return either (a) a named list of new/updated assays to merge (e.g., `list(q = ...)`), or (b) the full `arrays` list with modifications. Implementations must preserve the input array dimensions.

register_reducer *Register a reducer kernel*

Description

Register a reducer kernel

Usage

```
register_reducer(
  name,
  fun,
  requires,
  provides,
  options_schema = list(),
  input_shape = c("contrastwise", "joint_contrast")
)
```

Arguments

name	String identifier, e.g., "meta:fe"
fun	Function(beta, var, X, z, p, df, df1, df2, opts) -> named list
requires	Character vector of required inputs, e.g., c("beta","var")
provides	Character vector of outputs to be written
options_schema	Optional schema for options
input_shape	Reducer execution mode: "contrastwise" (default) or "joint_contrast" for reducers that consume the full contrast axis jointly

---

relabel_subjects *Relabel subjects in a GDS*

Description

Relabel subjects in a GDS

Usage

```
relabel_subjects(g, mapping)
```

Arguments

g	A realised GDS
mapping	Named character vector mapping old -> new subject ids

Value

Updated GDS with relabeled subjects

Examples

```
beta <- array(0, dim = c(2,2,1), dimnames = list(NULL, c("u1","u2"), "c1"))
var <- beta
g <- new_gds(list(beta = beta, var = var), space_sample_labels(c("x","y")), c("u1","u2"), "c1")
relabel_subjects(g, c(u1 = "s1", u2 = "s2"))
```

row_data *Extract row (sample) metadata from a GDS object*

Description

Extract row (sample) metadata from a GDS object

Usage

```
row_data(x)
```

Arguments

x	A GDS object

Value

Data frame with sample-level metadata

sample_groups *Extract sample-group metadata from a GDS object*

Description

Extract sample-group metadata from a GDS object

Usage

```
sample_groups(
  x,
  vars = c("feature_group", "spatial_group", "group", "parcel", "parcel_id")
)
```

Arguments

x	A GDS object or plan
vars	Candidate column names searched in `row_data(x)`

Value

A vector of group labels or `NULL` when unavailable

sample_labels *Extract sample labels from a GDS object*

Description

Extract sample labels from a GDS object

Usage

```
sample_labels(x)
```

Arguments

x	A GDS object or plan

Value

Character vector of sample labels

save_plan *Save a plan to JSON*

Description

Save a plan to JSON

Usage

```
save_plan(plan, file)
```

Arguments

plan	Plan or object coercible via as_plan()
file	Path to JSON file

space *Extract space descriptor from a GDS object*

Description

Extract space descriptor from a GDS object

Usage

```
space(x)
```

Arguments

x	A GDS object

Value

A space object

space_basis *Create a latent basis space descriptor*

Description

Create a latent basis space descriptor

Usage

```
space_basis(k, basis_name = NULL, projector = NULL, voxel_space = NULL)
```

Arguments

k	Number of components
basis_name	Optional name for the basis (e.g., "ICA")
projector	Optional matrix/function for voxel projection
voxel_space	Optional reference voxel space descriptor

Value

Space object

space_from_nifti *Create a voxel space from a NIfTI file*

Description

Create a voxel space from a NIfTI file

Usage

```
space_from_nifti(path, mask = NULL)
```

Arguments

path	Path to a NIfTI file (.nii or .nii.gz)
mask	Optional path to a mask NIfTI or a neuroim2::NeuroVol

Value

A space_voxel object

Examples

```
# if (requireNamespace("neuroim2", quietly = TRUE)) {
#   sp <- space_from_nifti(system.file("extdata", "tiny.nii", package = "fmrigds"))
# }
```

space_parcels *Create a parcels/ROI space descriptor*

Description

Create a parcels/ROI space descriptor

Usage

```
space_parcels(labels, lookup = NULL, membership = NULL)
```

Arguments

labels	Character vector of parcel labels
lookup	Optional lookup information (data frame or list)
membership	Optional mapping (list or sparse matrix) from parcels to voxels

Value

Space object

space_sample_labels *Create a simple label space for tabular samples*

Description

Create a simple label space for tabular samples

Usage

```
space_sample_labels(labels)
```

Arguments

labels	Character vector of sample labels

Value

Space object of type "sample_labels"

space_subset *Subset a space by sample indices*

Description

Subset a space by sample indices

Usage

```
space_subset(space, idx, pack = FALSE)
```

Arguments

space	A gds_space object
idx	Integer or logical index for the sample axis
pack	Logical; when TRUE and space is a dense voxel space, return a packed voxel space using idx as the new mask indices.

Value

A new space object with samples restricted

Examples

```
sp <- space_parcels(letters[1:5])
space_subset(sp, 2:4)
```

space_surface *Create a surface space descriptor*

Description

Create a surface space descriptor

Usage

```
space_surface(vertices, faces, hemi, template_id = NULL)
```

Arguments

vertices	Matrix (n x 3) of vertex coordinates
faces	Integer matrix (m x 3) of triangle indices (1-based)
hemi	Hemisphere identifier ("L", "R", "LR")
template_id	Optional template identifier

Value

Space object

space_voxel *Create a voxel space descriptor*

Description

Create a voxel space descriptor

Alias for space_voxel

Usage

```
space_voxel(
  dim,
  affine,
  mask_bitmap = NULL,
  mask_idx = NULL,
  storage = c("dense", "packed"),
  template_id = NULL
)

space_voxels(
  dim,
  affine,
```

```
  mask_bitmap = NULL,
  mask_idx = NULL,
  storage = c("dense", "packed"),
  template_id = NULL
)
```

Arguments

dim	Integer vector of length 3 (x, y, z)
affine	4x4 affine matrix
mask_bitmap	Optional logical array matching dim
mask_idx	Optional integer vector of active voxel indices (mask order)
storage	Either "dense" or "packed"
template_id	Optional template identifier string

Value

An object of class c("space_voxel", "gds_space")

---

split.gds *Split a GDS object by a grouping variable*

Description

Divide a GDS object into a list of GDS objects based on a grouping factor from col_data (subject-level metadata).

Usage

```
## S3 method for class 'gds'
split(x, f, drop = TRUE, ...)
```

Arguments

x	A GDS object.
f	A factor or character vector for splitting, OR a column name (as a string) from col_data(x).
drop	Logical; if TRUE, drop unused factor levels. Default TRUE.
...	Additional arguments (ignored).

Value

A named list of GDS objects, one per level of f.

Examples

```
## Not run:
# Split by group column in col_data
gds_by_group <- split(gds, "group")
gds_by_group$patients  # GDS with only patient subjects
gds_by_group$controls  # GDS with only control subjects

# Or provide a factor directly
groups <- factor(c("A", "B", "A", "B"))
gds_split <- split(gds, groups)

## End(Not run)
```

---

subjects *Extract subject identifiers from a GDS object*

Description

Extract subject identifiers from a GDS object

Usage

```
subjects(x)
```

Arguments

x	A GDS object

Value

Character vector of subject IDs

---

subset.image_catalog *Subset an image catalog*

Description

Filter catalog files based on metadata conditions.

Usage

```
## S3 method for class 'image_catalog'
subset(x, subset, ...)
```

Arguments

x	An image_catalog object.
subset	Logical expression evaluated in the context of metadata columns.
...	Additional arguments (ignored).

Value

A filtered image_catalog.

Examples

```
## Not run:
# Keep only files from subject "01"
sub_catalog <- subset(catalog, subject == "01")

# Multiple conditions
sub_catalog <- subset(catalog, subject %in% c("01", "02") & run == "1")

## End(Not run)
```

---

subset_eager *Eagerly subset a GDS and compute immediately*

Description

Eagerly subset a GDS and compute immediately

Usage

```
subset_eager(...)
```

Arguments

...	Arguments passed to subset(), then compute()

Value

A realized GDS object

summary.image_catalog *Summary method for image_catalog*

Description

Summary method for image_catalog

Usage

```
## S3 method for class 'image_catalog'
summary(object, ...)
```

Arguments

object	An image_catalog object.
...	Additional arguments (ignored).

Value

Invisibly returns the catalog.

UncertaintyRule *Specify how uncertainty is propagated through a map*

Description

Specify how uncertainty is propagated through a map

Usage

```
UncertaintyRule(
  mode = c("independent", "cov_provider", "kernel", "none"),
  df_rule = c("satterthwaite", "none"),
  cov_provider = NULL,
  kernel = NULL
)
```

Arguments

mode	One of "independent", "cov_provider", "kernel", "none"
df_rule	Degrees of freedom aggregation rule
cov_provider	Optional covariance provider function
kernel	Optional kernel function

Value

Uncertainty rule descriptor

unique.image_catalog *Get unique values of a metadata column*

Description

Get unique values of a metadata column

Usage

```
## S3 method for class 'image_catalog'
unique(x, incomparables = FALSE, column = "subject", ...)
```

Arguments

x	An image_catalog object.
incomparables	Passed to base unique (ignored for catalog method).
column	Name of metadata column to get unique values from.
...	Additional arguments (ignored).

Value

Unique values from the specified column.

unregister_posthoc *Unregister a post-hoc method*

Description

Unregister a post-hoc method

Usage

```
unregister_posthoc(name)
```

Arguments

name	Method identifier to remove

Value

Invisibly, TRUE if removed, FALSE if not found

use_weight *Convenience helper to use a stored weight array for reduction*

Description

Convenience helper to use a stored weight array for reduction

Usage

```
use_weight(g, name)
```

Arguments

g	A realised GDS containing the weight assay
name	Assay name to use as custom weights

Value

A list(list) suitable to splice into reduce(): list(weights = "custom", options = list(custom_weights =))

Examples

```
# opts <- use_weight(g2, "w_custom")
```

validate *Validate a GDS or plan*

Description

Validate a GDS or plan

Usage

```
validate(x, ...)
```

Arguments

x	A gds, gds_plan, or image_catalog
...	Additional arguments passed to methods

Value

TRUE if valid; otherwise errors with a descriptive message

validate.image_catalog *Validate catalog consistency*

Description

Check that the catalog has consistent file coverage across subjects and report any missing or extra files.

Usage

```
## S3 method for class 'image_catalog'
validate(x, by = "subject", expect = NULL, ...)
```

Arguments

x	An image_catalog object.
by	Column name to group by for consistency check. Default "subject".
expect	Optional character vector of expected file basenames per subject.
...	Additional arguments (ignored).

Value

A catalog_validation_report object.

Examples

```
## Not run:
report <- validate(catalog)
print(report)

report <- validate(catalog, expect = c("cope1.nii.gz", "varcope1.nii.gz"))

## End(Not run)
```

WarpFamily *Deformable warp family helper*

Description

Deformable warp family helper

Usage

```
WarpFamily(
  name,
  from_space,
  to_space,
  warps_by_subject,
  uncertainty = UncertaintyRule("independent")
)
```

Arguments

`name`	Family name
`from_space`	Source space
`to_space`	Target space
`warps_by_subject`	Named list of per-subject deformation fields
`uncertainty`	Uncertainty rule (default: independent)

Value

A MapFamily object for deformable warps

`with_col_data` *Attach subject-level covariates to a plan or GDS*

Description

Attach subject-level covariates to a plan or GDS

Usage

```
with_col_data(x, col_data)
```

Arguments

`x`	Plan, source, or realised GDS
`col_data`	Data frame keyed by subject identifiers (rownames)

Value

A `gds_plan` (for plans/sources) or a `gds` (for realised objects)

with_contrast_data *Attach contrast-level metadata to a plan or GDS*

Description

Attach contrast-level metadata to a plan or GDS

Usage

```
with_contrast_data(x, contrast_data)
```

Arguments

x	Plan, source, or realised GDS
contrast_data	Data frame keyed by contrast identifiers (rownames)

Value

A gds_plan (for plans/sources) or a gds (for realised objects)

with_row_data *Attach sample-level metadata to a plan or GDS*

Description

Attach sample-level metadata to a plan or GDS

Usage

```
with_row_data(x, row_data)
```

Arguments

x	Plan, source, or realised GDS
row_data	Data frame keyed by sample identifiers (rownames) when available

Value

A gds_plan (for plans/sources) or a gds (for realised objects)

write_catalog *Write catalog to JSON file*

Description

Save an image catalog to a JSON file for later retrieval with `read_catalog()`.

Usage

```
write_catalog(catalog, file)
```

Arguments

`catalog`	An image_catalog object.
`file`	Output file path (should end in `.json`).

Value

Invisibly returns the file path.

Examples

```
## Not run:
write_catalog(catalog, "my_study_manifest.json")

## End(Not run)
```

write_nifti_assays *Write image-like GDS assays as NIfTI files*

Description

Writes each selected assay/subject/contrast image in a voxel-space GDS to a separate NIfTI file and returns a manifest describing written and skipped outputs. This is intended for model outputs such as `coef:*`, `t_coef:*`, and `p_coef:*` assays.

Usage

```
write_nifti_assays(
  g,
  out_dir,
  prefix = NULL,
  assays = NULL,
  subjects = NULL,
  contrasts = NULL,
  sanitize = TRUE,
  overwrite = FALSE
)
```

Arguments

g	A realised gds with voxel space.
out_dir	Output directory.
prefix	Optional filename prefix.
assays	Character vector of assays to write. Defaults to all assays.
subjects	Optional subject labels to write. Defaults to all subjects.
contrasts	Optional contrast labels to write. Defaults to all contrasts.
sanitize	Logical; if TRUE, make filenames filesystem-friendly.
overwrite	Logical; if FALSE, existing files are skipped.

Value

A data.frame manifest with assay, subject, contrast, path, written, and skipped_reason columns.

Examples

```
## Not run:
fit <- group_ols(g, ~ group) |> compute()
manifest <- write_nifti_assays(fit, "group_maps", prefix = "auc")

## End(Not run)
```

---

write_out *Declare an output target for a plan*

Description

Adds a lazy write operation to the plan. Data are only written when compute() executes.

Usage

```
write_out(
  x,
  path,
  format = c("h5", "csv", "parquet", "nifti"),
  options = list()
)
```

Arguments

x	A plan or object coercible via as_plan()
path	Output file path
format	Output format ("h5", "csv", "parquet", "nifti")
options	Optional list of format-specific options

Value

Updated plan with a write node appended

Index