

Package: fmrigds (via r-universe)

October 10, 2026

Type Package

Title Lazy, Format-Agnostic Group Analysis for fMRI

Version 0.1.0

Depends R (>= 4.1.0)

Description Provides lazy, spatially typed group-analysis plans for observation-by-feature functional magnetic resonance imaging (fMRI) frames together with compatibility workflows for first-level statistical outputs represented as sample by subject by contrast Group Data Sets (GDS). Supports bounded feature-block reducers, tabular, 'NIfTI', 'HDF5', and 'fmrstore' inputs, space-aware transformations, provenance tracking, and reproducible export.

License MIT + file LICENSE

Copyright See inst/COPYRIGHTS for bundled vignette fonts.

Encoding UTF-8

LazyData false

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports Matrix, Rcpp, digest, jsonlite, data.table, hdf5r, stats, tools, utils

LinkingTo Rcpp, RcppArmadillo

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

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

Suggests testthat (>= 3.0.0), fmridataset, multidesign, arrow, RNifti, lme4, neuroim2, neurotabs, neurothresh, knitr, rmarkdown, pkgload, covr, pkgdown, albersdown, metafor, ggplot2, ps, ragg, systemfonts

Remotes github::bbuchsbaum/fmridataset,
github::bbuchsbaum/multidesign, github::bbuchsbaum/neuroim2,
github::bbuchsbaum/neurotabs, github::bbuchsbaum/neurothresh

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

VignetteBuilder knitr

Config/testthat/edition 3

Config/Needs/website albersdown

Config/pak/sysreqs libhdf5-dev

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

Date/Publication 2026-10-10 19:03:56 UTC

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

RemoteRef HEAD

RemoteSha fca311e7f9eb63f1198c93376a10e57c6b4a8763

Contents

fmrigds-package	5
align	6
alignment-registry	6
apply_common_mask	7
as_fmri_frame.gds	7
as_gds	8
as_gds.fmri_frame	10
as_gds.image_catalog	11
as_gds.NeuroVec	12
as_gds.NeuroVol	13
as_neurovec	14
as_neurovol_list	16
as_plan	17
assay	17
assay_info	18
assays	18
assert_compatible_spaces	19
assign_meta	19
attach_weight	20
coef_array	21
coef_cov_tri	21
col_data	22
common_mask	22
compute	23
contrast_data	24
contrasts	24
derive	25
detect_adapter	25
examination_control	26
examine_group	27
explain	29
explain_plan	29

extract_group	30
find_maps	31
fit_group	32
gds	33
gds_from_neurovol_nested	34
gds_from_neurovols	35
gds_from_nifti_maps	36
gds_from_scalar_maps	38
gds_metadata	40
gds_source	41
gds_to_tibble	41
get_adapter	42
get_map_family	42
get_posthoc	43
get_reducer	43
group_ols	44
group_plan	45
harmonise_contrasts	46
image_catalog	46
join_meta	47
list_assays	48
list_map_families	49
list_posthoc	50
list_reducers	50
load_plan	51
make_linear_family	51
make_warp_family	52
map_assays	53
map_linear	54
map_to	55
MapFamily	55
mask	56
MaskPolicy	57
metadata	57
model_matrix	58
new_image_catalog	58
nifti_source	59
OrthogonalFamily	60
OTFamily	61
plot.gds_examination	61
posthoc	62
posthoc-methods	63
preview	64
print.catalog_validation_report	65
print.image_catalog	65
read_catalog	66
reduce	66
reducer-lmm	69

reducer-ols-voxelwise	71
register_adapter	73
register_assay	74
register_map	74
register_nftab_adapter	75
register_posthoc	75
register_reducer	76
relabel_subjects	77
result_frame	78
row_data	79
sample_groups	79
sample_labels	80
save_plan	80
space	81
space_basis	81
space_from_nifti	82
space_parcel	82
space_portability	83
space_sample_labels	83
space_subset	84
space_surface	84
space_voxel	85
split.gds	86
subjects	87
subset.gds_plan	87
subset.image_catalog	88
summary.gds_examination	89
summary.image_catalog	89
UncertaintyRule	90
unique.image_catalog	90
unregister_posthoc	91
use_weight	91
validate	92
validate.image_catalog	92
WarpFamily	93
with_col_data	94
with_contrast_data	94
with_row_data	95
write_catalog	95
write_nifti_assays	96
write_out	98
write_report	98

fmrigds-package	<i>fmrigds</i>
-----------------	----------------

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/fmrigrds>
- Report bugs at <https://github.com/bbuchsbaum/fmrigrds/issues>

align	<i>Align subjects into a consensus space</i>
-------	--

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

alignment-registry	<i>Register, list, and get alignments (map families)</i>
--------------------	--

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	<i>Apply a common mask to two realised GDS objects</i>
-------------------	--

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 subsetting g1, g2, and indices idx1, idx2

as_fmri_frame.gds	<i>Convert a legacy rectangular GDS cube to a canonical frame</i>
-------------------	---

Description

Convert a legacy rectangular GDS cube to a canonical frame

Usage

```
as_fmri_frame.gds(
  x,
  subject = NULL,
  contrast = NULL,
  assay_map = c(var = "variance", se = "std_error"),
  ...
)
```

Arguments

x	A realized legacy gds object.
subject, contrast	Names for the generated observation key columns. NULL recovers the names stored by <code>as_gds()</code> when available, otherwise defaults to "subject_id" and "contrast_id".
assay_map	Named character vector mapping legacy GDS assay names to canonical frame names.
...	Reserved for future use.

Value

An `fmri_frame` with one row per subject-contrast cell.

as_gds	<i>Coerce common R objects into a GDS</i>
--------	---

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 <code>gds_metadata()</code>
assay_name	Name to use for the single-array input (default: "beta")
mapping	Optional list mapping column names for axes and assays. Default: <code>list(sample="sample", subject="subject", contrast="contrast", assays=list(beta="beta", var="var"), contrast_data=NULL)</code>
contrast_data	Optional contrast-level metadata. When omitted, <code>mapping\$contrast_data</code> 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.fmri_frame	<i>Project a rectangular frame to the legacy GDS cube</i>
-------------------	---

Description

This compatibility projection is deliberately strict. The selected observation annotations must define one complete, unique Cartesian product of subject and contrast keys. Ragged frames remain frames and are rejected rather than padded with implicit slabs.

Usage

```
## S3 method for class 'fmri_frame'
as_gds(
  x,
  subject = "subject_id",
  contrast = "contrast_id",
  subjects = NULL,
  contrasts = NULL,
  assay_names = names(fmridataset::assays(x)),
  assay_map = NULL,
  memory_budget = 2 * 1024^3,
  metadata = list(),
  ...
)
```

Arguments

x	An fmri_frame or synchronized frame view.
subject, contrast	Observation columns defining the rectangular axes. Either may be NULL only for an explicit singleton axis.
subjects, contrasts	Optional explicit axis order or singleton label. A NULL subject column defaults to a singleton "meta" axis; a NULL contrast column defaults to "contrast1".
assay_names	Assays to realize into the cube.
assay_map	Named character vector mapping frame assay names to GDS assay names. By default variance becomes var and std_error becomes se.
memory_budget	Maximum total bytes realized across projected assays.
metadata	Additional GDS metadata.
...	Reserved for future use.

Value

A realized legacy gds compatibility projection.

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 the beta assay and, optionally, one uncertainty assay (se or var); mapping any other assay is an error. When the catalog has a subject column, its values become the GDS subject labels and uncertainty files are paired with beta files by subject; 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

```
if (requireNamespace("neuroim2", quietly = TRUE) &&
    requireNamespace("RNifti", quietly = TRUE)) {
  # Write tiny 4x4x2 beta and SE maps for three subjects
  root <- tempfile("study_")
  for (s in c("01", "02", "03")) {
    d <- file.path(root, paste0("sub-", s), "stats")
    dir.create(d, recursive = TRUE)
    RNifti::writeNifti(array(rnorm(32), c(4, 4, 2)),
                       file.path(d, paste0("sub-", s, "_beta.nii.gz")))
    RNifti::writeNifti(array(0.5, c(4, 4, 2)),
                       file.path(d, paste0("sub-", s, "_se.nii.gz")))
  }

  catalog <- image_catalog(root, "sub-*/stats/*.nii.gz",
                           path_regex = "sub-(?<subject>[^/]+)") |>
    map_assays(beta = "_beta", se = "_se")

  plan <- as_gds(catalog)
  result <- compute(plan)
  print(result)
```

```

    unlink(root, recursive = TRUE)
}

```

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 <code>neuroim2::NeuroVec</code> 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 <code>along = "subject"</code> , or length 1 when <code>along = "contrast"</code> .

contrasts	Character vector of contrast names. Length must match the 4th dimension when <code>along = "contrast"</code> , or length 1 when <code>along = "subject"</code> .
mask	Optional mask. See <code>as_gds.NeuroVol</code> for options.
...	Additional arguments passed to <code>new_gds()</code> .

Value

A gds object with voxel space.

Examples

```
if (requireNamespace("neuroim2", quietly = TRUE)) {
  sp4 <- neuroim2::NeuroSpace(c(4, 4, 2, 3), spacing = c(2, 2, 2))
  vec4d <- neuroim2::NeuroVec(array(rnorm(96), c(4, 4, 2, 3)), sp4)

  # 4D volume where each volume is a subject
  gds_obj <- as_gds(vec4d, along = "subject",
    subjects = c("sub-01", "sub-02", "sub-03"))
  print(subjects(gds_obj))

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

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

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

Value

A `gds` object with voxel space.

Examples

```

if (requireNamespace("neuroim2", quietly = TRUE)) {
  sp <- neuroim2::NeuroSpace(c(4, 4, 2), spacing = c(2, 2, 2))
  vol <- neuroim2::NeuroVol(array(rnorm(32), c(4, 4, 2)), sp)
  gds_obj <- as_gds(vol, subject = "sub-01", contrast = "faces")
  print(gds_obj)
}

```

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

<code>x</code>	A GDS object with a voxel space.
<code>assay</code>	Name of the assay to extract (default: "beta").
<code>along</code>	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.
<code>subset_subjects</code>	Optional character vector of subject IDs to include.
<code>subset_contrasts</code>	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

```
if (requireNamespace("neuroim2", quietly = TRUE)) {
  sp4 <- neuroim2::NeuroSpace(c(4, 4, 2, 4), spacing = c(2, 2, 2))
  vec4d <- neuroim2::NeuroVec(array(rnorm(128), c(4, 4, 2, 4)), sp4)
  gds <- as_gds(vec4d, along = "subject", subjects = paste0("sub0", 1:4))

  # Stack all subjects into a 4D volume
  vec <- as_neurovec(gds, assay = "beta", along = "subject")
  print(dim(vec))

  # Stack specific subjects
  vec <- as_neurovec(gds, assay = "beta", along = "subject",
    subset_subjects = c("sub01", "sub02"))
  print(dim(vec))
}
```

as_neurovol_list	<i>Convert GDS assay to list of NeuroVol objects</i>
------------------	--

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

<code>x</code>	A GDS object with a voxel space.
<code>assay</code>	Name of the assay to extract (default: "beta").
<code>by</code>	How to organize the output list. One of: • "subject": list indexed by subject, each element is a <code>NeuroVol</code> (only works when there's a single contrast) • "contrast": list indexed by contrast, each element is a <code>NeuroVol</code> (only works when there's a single subject)
<code>drop_dim</code>	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

```
if (requireNamespace("neuroim2", quietly = TRUE)) {
  sp4 <- neuroim2::NeuroSpace(c(4, 4, 2, 4), spacing = c(2, 2, 2))
  vec4d <- neuroim2::NeuroVec(array(rnorm(128), c(4, 4, 2, 4)), sp4)
  gds <- as_gds(vec4d, along = "subject", subjects = paste0("sub0", 1:4))

  # Extract beta maps as list of NeuroVols (one per subject)
```

```

vols <- as_neurovol_list(gds, assay = "beta")
print(names(vols))
print(vols$sub01) # NeuroVol for subject "sub01"

# Extract by contrast (here: one subject with three contrasts)
sp3 <- neuroim2::NeuroSpace(c(4, 4, 2, 3), spacing = c(2, 2, 2))
vec_con <- neuroim2::NeuroVec(array(rnorm(96), c(4, 4, 2, 3)), sp3)
gds_con <- as_gds(vec_con, along = "contrast",
                  contrasts = c("faces", "places", "objects"))
vols_by_con <- as_neurovol_list(gds_con, assay = "beta", by = "contrast")
print(names(vols_by_con))
}

```

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	<i>Retrieve assay metadata</i>
------------	--------------------------------

Description

Retrieve assay metadata

Usage

assay_info(name)

Arguments

name Assay name

Value

Assay metadata list or NULL if not found

assays	<i>Extract assays from a GDS object</i>
--------	---

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

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

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

```
root <- tempfile("study_")
d <- file.path(root, "sub-01")
dir.create(d, recursive = TRUE)
invisible(file.create(file.path(d, c("run-1_faces.nii", "run-2_faces.nii",
                                     "run-1_houses.nii"))))

catalog <- image_catalog(root, "sub-*/*.nii")

catalog <- assign_meta(catalog, "run", "run-([0-9]+)", source = "basename")
catalog <- assign_meta(catalog, "contrast", "_([^_]+)\\.nii", replacement = "\\1")
catalog$metadata[, c("basename", "run", "contrast")]

unlink(root, recursive = TRUE)
```

attach_weight	<i>Attach a custom weight array to a GDS</i>
---------------	--

Description

Stores a weight array as an assay. Note that none of the built-in registered reducers currently consume custom weight assays: `reduce()` refuses `weights = "custom"` (and `"n_eff"`) unless the selected reducer declares support for that scheme, rather than silently ignoring it.

Usage

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

Arguments

g	A realised GDS
name	Assay-like name for the weights (e.g., <code>"n_eff"</code> or <code>"w_custom"</code>)
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)
```

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	<i>Extract column (subject) metadata from a GDS object</i>
----------	--

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	<i>Compute a common mask between two spaces</i>
-------------	---

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	<i>Extract contrast-level metadata from a GDS object</i>
---------------	--

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	<i>Extract contrast identifiers from a GDS object</i>
-----------	---

Description

For objects that are not GDS objects or plans (for example factors), the default method delegates to `stats::contrasts()`, so attaching `fmrgrids` does not break base model-contrast code.

Usage

```
contrasts(x, ...)
```

Arguments

x A GDS object or plan
 ... Passed to `stats::contrasts()` by the default method.

Value

Character vector of contrast names

derive	<i>Record a derivation operation in a plan</i>
--------	--

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

detect_adapter	<i>Detect the best adapter for a source</i>
----------------	---

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

examination_control *Configure group examination*

Description

`examination_control()` validates the bounded execution, residual-geometry, review, staging, and numerical settings used by `examine_group()`. Keeping these settings in a typed object makes defaults visible and reproducible.

Usage

```
examination_control(
  block_size = 1024L,
  geometry = list(rank = 32L, oversample = 8L, cap = 8, balance_contrasts = TRUE,
    stability_replicates = 2L),
  review = list(surprise = list(energy_threshold = 2.5, tail_threshold = 0.01,
    residual_threshold = 3), influence = list(energy_threshold = 1, max_abs_threshold =
    2), quality = list(coverage_fraction = list(direction = "low", threshold = 0.8)),
    min_stability = 0.7),
  exact_refit_n = 5L,
  retain_n = 5L,
  staging = list(tempdir = NULL),
  tolerance = list(rank = sqrt(.Machine$double.eps), leverage = 1e-08, degeneracy =
    1e-12)
)
```

Arguments

<code>block_size</code>	Positive number of samples read per scan block.
<code>geometry</code>	Named list controlling the deterministic low-rank residual representation. Supported entries are <code>rank</code> , <code>oversample</code> , <code>cap</code> , <code>balance_contrasts</code> , and <code>stability_replicates</code> . With <code>balance_contrasts = TRUE</code> each contrast's residuals are scaled by its own eligible feature count so contrasts contribute comparably to the geometry. <code>stability_replicates = 0</code> disables split-feature stability assessment; review criteria are then not gated on stability.
<code>review</code>	Named list controlling review status. Supported entries are <code>surprise</code> , <code>influence</code> , <code>quality</code> , and <code>min_stability</code> .
<code>exact_refit_n</code>	Maximum number of retained random-effects subjects, taken in decreasing <code>review_priority</code> , that receive an exact heterogeneity refit. Other retained subjects keep screening-mode maps.
<code>retain_n</code>	Number of highest-priority subjects retained for localization when no explicit retain set is supplied.
<code>staging</code>	Named list with optional <code>tempdir</code> . Staging stores are always removed on success or failure.
<code>tolerance</code>	Named list of numerical tolerances: <code>rank</code> , <code>leverage</code> , and <code>degeneracy</code> .

Value

An object of class `gds_examination_control`.

examine_group	<i>Examine subjects under an intended group model</i>
---------------	---

Description

`examine_group()` branches from subject-level data at a group reducer. It reports model-conditioned predictive surprise separately from each subject's exact influence on requested group estimands. It never deletes a subject or rewrites the source analysis.

Usage

```
examine_group(
  x,
  method = NULL,
  formula = NULL,
  options = list(),
  quality = NULL,
  estimands = NULL,
  retain = NULL,
  control = examination_control(),
  na_action = c("fail", "exclude")
)
```

Arguments

<code>x</code>	A subject-level gds_plan , gds_source , or gds , optionally containing one group reducer and a post-hoc conclusion tail.
<code>method</code>	Reducer method when <code>x</code> has no reducer.
<code>formula</code>	Model formula when <code>x</code> has no reducer.
<code>options</code>	Reducer options when <code>x</code> has no reducer.
<code>quality</code>	Optional character vector naming first-level quality columns in <code>col_data(x)</code> . Metrics are descriptive unless an explicit direction and absolute threshold are declared in <code>control\$review\$quality</code> .
<code>estimands</code>	Named linear estimands or design-column names. See examination_control() for execution controls.
<code>retain</code>	Optional subject IDs retained for later localization regardless of review rank.
<code>control</code>	A validated object from examination_control() .
<code>na_action</code>	Missing-covariate policy. The default fails; "exclude" deterministically records and excludes affected subjects from this examination only.

Details

When `x` already contains one reducer, the examination inherits that reducer's method, formula, options, and weighting semantics. Supplying model overrides in that case is an error. When no reducer is present, method is required and formula-driven reducers default to ~ 1 .

Value

A `gds_examination` object.

Result contract

The returned object keeps spatial maps in ordinary GDS objects and keeps subject, contrast, estimand, availability, sensitivity, and provenance data in tables or named lists. `subject_data$review_priority` orders inspection; `subject_data$review_status` changes to "review" only when an absolute, stability-aware criterion is met. Random-effects screening is labeled "tau2_fixed_full"; exact retained-subject refits are labeled "tau2_refit_exact" and never alter the original review queue.

Examples

```
subject_ids <- paste0("sub-", 1:8)
participant_data <- data.frame(
  group = factor(rep(c("control", "patient"), each = 4)),
  row.names = subject_ids
)
beta <- array(rnorm(12 * 8, sd = 0.2), c(12, 8, 1))
beta[, participant_data$group == "patient", 1] <-
  beta[, participant_data$group == "patient", 1] + 0.4
variance <- array(0.1, dim(beta))
subject_gds <- new_gds(
  assays = list(beta = beta, var = variance),
  space = space_sample_labels(paste0("feature-", 1:12)),
  subjects = subject_ids,
  contrasts = "task",
  col_data = participant_data
)
analysis <- as_plan(subject_gds) |>
  reduce(method = "meta:re_reg", formula = ~ group)
exam <- examine_group(
  analysis,
  estimands = "grouppatient",
  control = examination_control(retain_n = 2L)
)
summary(exam)
if (requireNamespace("ggplot2", quietly = TRUE)) plot(exam)
```

explain	<i>Explain objects and plans</i>
---------	----------------------------------

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	<i>Explain a plan's operations in a tidy table</i>
--------------	--

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	<i>Extract NeuroVol list for a specific group</i>
---------------	---

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

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

Value

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

Examples

```
if (requireNamespace("neuroim2", quietly = TRUE)) {
  sp4 <- neuroim2::NeuroSpace(c(4, 4, 2, 4), spacing = c(2, 2, 2))
  vec4d <- neuroim2::NeuroVec(array(rnorm(128), c(4, 4, 2, 4)), sp4)
  gds <- as_gds(vec4d, along = "subject", subjects = paste0("sub0", 1:4))
  gds <- with_col_data(gds, data.frame(
    diagnosis = c("patient", "control", "patient", "control"),
    row.names = subjects(gds)
  ))

  # Get beta maps for all patients
  patient_vols <- extract_group(gds, "diagnosis", "patient", assay = "beta")
  print(names(patient_vols))

  # Get beta maps for controls
  control_vols <- extract_group(gds, "diagnosis", "control", assay = "beta")
  print(names(control_vols))
}
```

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

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

fit_group

Fit a registered group reducer over frame feature blocks

Description

fit_group() is the eager convenience wrapper for `group_plan()` followed by `compute()`. The default known-variance random-intercept model is $\text{diag}(\text{sampling_variance}) + \text{vc_resid } I + \text{vc_intercept } Z Z'$; other registered frame kernels may use different input and model contracts.

Usage

```
fit_group(
  frame,
  estimate = "beta",
  variance = "variance",
  design,
  memory_budget = 256 * 1024^2,
  block_size = NULL,
  fit = c("REML", "ML"),
  method = "lmm:ri_knownvar",
  options = list()
)
```

Arguments

frame	An fmri_frame or synchronized view.
estimate	Name of the effect assay.
variance	Name of the known sampling-variance assay.
design	A design_spec or compiled_design.
memory_budget	Maximum bytes for the two input feature blocks.
block_size	Optional explicit feature-block width.
fit	Likelihood criterion, "REML" or "ML".
method	Registered frame-capable reducer. The default is "lmm:ri_knownvar"; "ols:voxelwise" provides unweighted fixed-effects fitting when no sampling-variance assay is available.
options	Named reducer options.

Value

An fmri_group_fit whose result is an fmri_frame sharing the input feature IDs and spatial domain.

gds	<i>Create a Group Data Set (GDS)</i>
-----	--------------------------------------

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 <code>open()/probe()</code> . Common metadata arguments include <code>col_data</code> , <code>row_data</code> , and <code>contrast_data</code> . For stacked repeated-measures tabular data, <code>contrast_data_cols</code> 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: <code>list(subject1 = list(con1 = vol, con2 = vol), ...)</code> • List of 4D NeuroVec: <code>list(subject1 = vec4d, subject2 = vec4d, ...)</code> 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 <code>as_gds.NeuroVol</code> for options.
col_data	Optional data frame of subject-level covariates.
metadata	Optional list of additional metadata to attach.
...	Additional arguments passed to <code>new_gds()</code> .

Value

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

Examples

```
if (requireNamespace("neuroim2", quietly = TRUE)) {
  sp <- neuroim2::NeuroSpace(c(4, 4, 2), spacing = c(2, 2, 2))
  make_vol <- function(x) neuroim2::NeuroVol(array(x, c(4, 4, 2)), sp)
  subj <- c("sub-01", "sub-02")

  # Example 1: Nested list structure (subjects x contrasts)
  beta <- lapply(stats::setNames(subj, subj), function(s)
    list(faces = make_vol(rnorm(32)), places = make_vol(rnorm(32))))
  var <- lapply(stats::setNames(subj, subj), function(s)
    list(faces = make_vol(runif(32, 0.5, 1.5)),
         places = make_vol(runif(32, 0.5, 1.5))))

  gds_obj <- gds_from_neurovol_nested(beta, var = var)
  print(gds_obj)

  # Example 2: List of 4D NeuroVecs (each subject has a multi-contrast 4D image)
  sp4 <- neuroim2::NeuroSpace(c(4, 4, 2, 3), spacing = c(2, 2, 2))
  beta_vecs <- lapply(stats::setNames(subj, subj), function(s)
    neuroim2::NeuroVec(array(rnorm(96), c(4, 4, 2, 3)), sp4))
  # Without variance maps a unit-variance placeholder is used (with a warning)
  gds_obj <- gds_from_neurovol_nested(beta_vecs,
                                     contrasts = c("faces", "places", "objects"))
  print(contrasts(gds_obj))
}
```

gds_from_neurovols	<i>Create GDS from lists of NeuroVol objects</i>
--------------------	--

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 <code>as_gds.NeuroVol</code> for options.
col_data	Optional data frame of subject-level covariates. Row names must match subject IDs (names of beta).
...	Additional arguments passed to <code>new_gds()</code> .

Value

A gds object.

Examples

```
if (requireNamespace("neuroim2", quietly = TRUE)) {
  sp <- neuroim2::NeuroSpace(c(4, 4, 2), spacing = c(2, 2, 2))
  make_vol <- function(x) neuroim2::NeuroVol(array(x, c(4, 4, 2)), sp)

  # Beta and variance maps for each subject
  beta_vols <- list(
    `sub-01` = make_vol(rnorm(32)),
    `sub-02` = make_vol(rnorm(32)),
    `sub-03` = make_vol(rnorm(32))
  )
  var_vols <- lapply(beta_vols, function(v) make_vol(runif(32, 0.5, 1.5)))

  gds_obj <- gds_from_neurovols(beta_vols, var = var_vols)
  print(gds_obj)

  # With subject covariates
  covars <- data.frame(age = c(25, 30, 35), group = c("A", "B", "A"),
    row.names = c("sub-01", "sub-02", "sub-03"))
  gds_obj <- gds_from_neurovols(beta_vols, var = var_vols, col_data = covars)
  print(col_data(gds_obj))
}
```

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

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

This helper is **single-contrast**: every file in `maps$file` is loaded as one map per subject along a single contrast axis. A `contrast` column is used only to *label* that one contrast (all non-missing values must agree); it does not pivot a long subject x contrast table into a grid. For multi-contrast workflows, call `gds_from_nifti_maps()` once per contrast and combine the results, or assemble the source manually with `nifti_source()`.

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

```
if (requireNamespace("neuroim2", quietly = TRUE) &&
    requireNamespace("RNifti", quietly = TRUE)) {
  # Write tiny 4x4x2 subject-level maps for four subjects
  dir <- tempfile("maps_")
  dir.create(dir)
  subj <- c("1001", "1002", "1003", "1004")
  files <- file.path(dir, paste0(subj, "_naive_diag_mean.nii.gz"))
  for (f in files) RNifti::writeNifti(array(rnorm(32), c(4, 4, 2)), f)
```

```

# NOTE: `contrast` is a single label here. gds_from_nifti_maps() is
# single-contrast; to analyse several contrasts, subset `maps` and call
# once per contrast (a multi-valued `contrast` column triggers a warning).
maps <- data.frame(
  file = files,
  subject = subj,
  contrast = "naive_diag_mean"
)

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

# Beta-only maps: a unit-variance placeholder is added (with a warning)
g <- gds_from_nifti_maps(maps) |>
  with_col_data(group_info)

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

unlink(dir, recursive = TRUE)
}

```

`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

<code>files</code>	Character vector of NIfTI paths, a data.frame with a file column, or an image_catalog() .
<code>subject</code>	Optional subject identifiers. Required for character files unless files is named; ignored when files already has a subject column.
<code>contrast</code>	Contrast label(s). A scalar value is recycled across files. Ignored when files already has a contrast column unless explicitly supplied.
<code>col_data</code>	Optional subject-level covariates. If a subject column is present and row names are absent, it is used as the row key.
<code>mask</code>	Optional mask passed to gds_from_nifti_maps() .
<code>...</code>	Additional arguments forwarded to gds_from_nifti_maps() .

Value

A realised [gds](#) object.

Examples

```

if (requireNamespace("neuroim2", quietly = TRUE) &&
    requireNamespace("RNifti", quietly = TRUE)) {
  # Write tiny 4x4x2 subject-level metric maps for four subjects
  dir <- tempfile("scalar_maps_")
  dir.create(dir)
  subj <- c("sub-01", "sub-02", "sub-03", "sub-04")
  files <- file.path(dir, paste0(subj, "_auc.nii.gz"))
  for (f in files) RNifti::writeNifti(array(rnorm(32), c(4, 4, 2)), f)
  participants <- data.frame(subject = subj,
                             group = c("control", "control", "patient", "patient"))

  # Beta-only maps: a unit-variance placeholder is added (with a warning)
  g <- gds_from_scalar_maps(
    files = files,
    subject = subj,
    contrast = "auc",
    col_data = participants
  )
  print(g)
  fit <- group_ols(g, ~ group) |> compute()

  unlink(dir, recursive = TRUE)
}

```

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 = "fmrigrds", 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_source	<i>Define a plan source (adapter binding)</i>
------------	---

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	<i>Tidy long-form export for GDS</i>
---------------	--------------------------------------

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 <code>col_data</code>

Value

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

get_adapter	<i>Retrieve a registered adapter</i>
-------------	--------------------------------------

Description

Retrieve a registered adapter

Usage

```
get_adapter(name)
```

Arguments

name	Adapter name
------	--------------

Value

Adapter list

get_map_family	<i>Retrieve a registered map family by name</i>
----------------	---

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	<i>Get post-hoc method by name</i>
-------------	------------------------------------

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	<i>Get reducer by name</i>
-------------	----------------------------

Description

Get reducer by name

Usage

```
get_reducer(name)
```

Arguments

name	Reducer name
------	--------------

Value

Reducer list or NULL if not found

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

<code>x</code>	A realised <code>gds</code> or <code>gds_plan</code> .
<code>formula</code>	One-sided model formula for <code>group_ols()</code> .
<code>col_data</code>	Optional subject-level covariates to attach before fitting.
<code>options</code>	Reducer options passed to <code>reduce()</code> .
<code>...</code>	Additional arguments passed to <code>reduce()</code> .
<code>subset</code>	Optional expression evaluated in <code>col_data(x)</code> to select subjects before fitting the one-sample model.
<code>group</code>	Name of the grouping column in <code>col_data(x)</code> .
<code>baseline</code>	Optional reference level for the grouping factor.
<code>level</code>	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

```
# Synthetic voxel-like data: 5 samples x 4 subjects x 1 contrast
beta <- array(rnorm(5 * 4), c(5, 4, 1))
var <- array(1, c(5, 4, 1))
g <- new_gds(list(beta = beta, var = var), space_sample_labels(letters[1:5]),
             subjects = paste0("s", 1:4), contrasts = "c1",
             col_data = data.frame(group = c("control", "control",
                                             "patient", "patient"),
                                   row.names = paste0("s", 1:4)))

fit <- group_ols(g, ~ group) |> compute()
names(assays(fit))
one <- one_sample(g) |> compute()
two <- two_sample(g, group = "group", baseline = "control") |> compute()
```

group_plan

Construct a lazy frame-native group-analysis plan

Description

group_plan() compiles design metadata and resolves stable axis IDs without reading imaging assays. Numerical values are read only by `compute()`, in bounded feature blocks, and dispatched through the selected reducer's frame execution contract.

Usage

```
group_plan(
  frame,
  design,
  method = "lmm:ri_knownvar",
  estimate = "beta",
  variance = "variance",
  memory_budget = 256 * 1024^2,
  block_size = NULL,
  options = list()
)
```

Arguments

frame	An fmri_frame or synchronized view.
design	A multidesign::design_spec or compiled design.
method	Registered reducer name with frame execution support.
estimate	Effect-assay name.
variance	Known sampling-variance assay name, or NULL for reducers that do not consume it.
memory_budget	Maximum bytes for input assay blocks.
block_size	Optional explicit feature-block width.
options	Reducer options.

Value

A serializable fmri_group_plan.

harmonise_contrasts	<i>Harmonise contrast names in a GDS</i>
---------------------	--

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	<i>Discover image files and create a catalog</i>
---------------	--

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

```
# Create a small study tree (empty placeholder files are enough for discovery)
root <- tempfile("study_")
for (s in c("01", "02", "03")) {
  d <- file.path(root, paste0("sub-", s), "analysis")
  dir.create(d, recursive = TRUE)
  invisible(file.create(file.path(d, c("cope1.nii.gz", "varcope1.nii.gz"))))
}

# Discover files with subject extraction
catalog <- image_catalog(
  root = root,
  pattern = "sub-*/analysis/*.nii.gz",
  path_regex = "sub-(?<subject>[^/]+)"
)
catalog
catalog$metadata[, c("basename", "subject")]

unlink(root, recursive = TRUE)
```

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

```
# Build a tiny study tree with empty placeholder files
root <- tempfile("study_")
for (s in c("01", "02")) {
  d <- file.path(root, paste0("sub-", s))
  dir.create(d, recursive = TRUE)
  invisible(file.create(file.path(d, c("cope1.nii.gz", "varcope1.nii.gz"))))
}
catalog <- image_catalog(root, "sub-*/*.nii.gz",
  path_regex = "sub-(?<subject>[^\s]+)")

demographics <- data.frame(subject = c("01", "02"), age = c(25, 30))
catalog <- join_meta(catalog, demographics, by = "subject")
catalog$metadata[, c("basename", "subject", "age")]

unlink(root, recursive = TRUE)
```

list_assays

List the assays present on a GDS (or that a reducer would produce)

Description

Discoverability helper for finding assay names to pass to [assay\(\)](#), [write_nifti_assays\(\)](#), or [write_out\(\)](#) without having to run a job and inspect `names(assays(fit))`. Works on a realised [gds](#), a lazy [gds_plan/gds_source](#) (reporting the *input* assays the source probed), or—when reducer is supplied—reports a reducer’s declared output stems without computing anything.

Usage

```
list_assays(x, reducer = NULL, info = TRUE)
```

Arguments

x	A realised gds , a gds_plan , or a gds_source .
reducer	Optional reducer name (e.g. "meta:re", "ols:voxelwise", or an alias like "random"). When supplied, x is ignored and the reducer's declared provides stems are returned. Regression/LMM reducers expand coef/se_coef/z_coef/t_coef/p_coef into per-term coef:<term> etc. at compute time, so realised names depend on the model formula.
info	Logical; if TRUE (default) return a data frame with assay, role, and units columns (from assay_info() where registered, else NA); if FALSE return a plain character vector of names.

Value

A data frame (or a character vector when info = FALSE).

See Also

[assays\(\)](#), [assay\(\)](#), [assay_info\(\)](#), [list_reducers\(\)](#), [list_posthoc\(\)](#), [write_nifti_assays\(\)](#)

Examples

```
beta <- array(rnorm(5 * 4), c(5, 4, 1))
var <- array(1, c(5, 4, 1))
g <- new_gds(list(beta = beta, var = var), space_sample_labels(letters[1:5]),
             subjects = paste0("s", 1:4), contrasts = "c1")

fit <- one_sample(g) |> compute()
list_assays(fit)                # names + roles of computed assays
list_assays(reducer = "meta:re") # what random-effects would produce
```

list_map_families	<i>List registered map families</i>
-------------------	-------------------------------------

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	<i>List registered post-hoc methods</i>
--------------	---

Description

List registered post-hoc methods

Usage

```
list_posthoc()
```

Value

A sorted character vector of registered post-hoc method names.

See Also

[posthoc-methods](#) for what each method outputs.

list_reducers	<i>List registered reducers</i>
---------------	---------------------------------

Description

List registered reducers

Usage

```
list_reducers()
```

Value

A sorted character vector of registered reducer names.

See Also

[list_assays\(\)](#) to preview a reducer's output assays.

load_plan	<i>Load a plan from JSON</i>
-----------	------------------------------

Description

Load a plan from JSON

Usage

```
load_plan(file)
```

Arguments

file	JSON file produced by save_plan
------	---

Value

A gds_plan. A warning is issued when the digest stored in the file does not match the reconstructed plan (e.g. the file was edited).

Security

Map families and map_linear operators are restored with `base::unserialize()`, which can execute arbitrary code embedded in a crafted file. Only call `load_plan()` on files you created or otherwise trust.

make_linear_family	<i>Convenience helpers to create and register alignments</i>
--------------------	--

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

make_warp_family	<i>Create a warp-based alignment family from on-disk paths</i>
------------------	--

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 <code>list(column = value)</code> for exact column matching

Value

Updated image_catalog with assay_map.

Examples

```
# Build a tiny study tree with empty placeholder files
root <- tempfile("study_")
for (s in c("01", "02")) {
  d <- file.path(root, paste0("sub-", s))
  dir.create(d, recursive = TRUE)
  invisible(file.create(file.path(d, c("cope1.nii.gz", "varcope1.nii.gz"))))
}
catalog <- image_catalog(root, "sub-*/*.nii.gz",
  path_regex = "sub-(?<subject>[^/]+)")

# By filename pattern (regex matched against basenames)
catalog <- map_assays(catalog,
  beta = "^cope[0-9]+\\.nii",
  var = "^varcope[0-9]+\\.nii")
```

```

)

# By metadata column value
catalog <- assign_meta(catalog, "stat_type", "^[a-z+][0-9]+")
catalog <- map_assays(catalog,
  beta = list(stat_type = "cope"),
  var = list(stat_type = "varcope")
)
summary(catalog)

unlink(root, recursive = TRUE)

```

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	<i>Add a space transformation to a plan</i>
--------	---

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

MapFamily	<i>Create a subject-aware map family</i>
-----------	--

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	<i>Apply a mask policy lazily</i>
------	-----------------------------------

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

MaskPolicy	<i>Define a mask policy</i>
------------	-----------------------------

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	<i>Extract metadata from a GDS object</i>
----------	---

Description

Extract metadata from a GDS object

Usage

```
metadata(x)
```

Arguments

x	A GDS object
---	--------------

Value

Metadata list

model_matrix	<i>Build a design matrix from attached col_data</i>
--------------	---

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 <code>as_plan()</code>)
formula	Model formula (or string convertible via <code>stats::as.formula</code>)

Value

A numeric model matrix

new_image_catalog	<i>Create a new image catalog object</i>
-------------------	--

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, and variance-weighted reducers will refuse such a GDS.

Usage

```
nifti_source(
  beta = NULL,
  se = NULL,
  subject = NULL,
  subjects = NULL,
  contrast = NULL,
  contrasts = NULL
)
```

Arguments

<code>beta</code>	Character vector of NIfTI paths or a directory containing beta/effect images (optional, can be NULL if only <code>se</code> is provided)
<code>se</code>	Character vector of NIfTI paths or a directory containing standard error images (optional, can be NULL if only <code>beta</code> is provided)
<code>subject, subjects</code>	Optional subject labels. When both <code>beta</code> and <code>se</code> are supplied, these labels declare that the vectors are already aligned in matching subject order and bypass filename-key pairing.
<code>contrast, contrasts</code>	Optional contrast labels. Length must match the number of contrasts in each NIfTI image (one for ordinary 3D maps).

Details

When both `beta` and `se` are supplied, files are paired by subject key unless explicit `subject/subjects` labels are supplied. The fallback key is the file basename (extension removed) with a **trailing** statistic token stripped, matching `([_.-])(beta|cope|varcope|var|variance|effect|se|stderr|sterr|sigma|std(err)?)` (case insensitive). For example `sub-01_beta.nii.gz` and `sub-01_se.nii.gz` both reduce to the key `sub-01`. Filenames where the statistic token is not last (e.g. `sub-01_stat-beta_sm-2.nii.gz`) will fail to pair unless explicit subject labels are provided. For beta/stat-only maps, prefer `gds_from_scalar_maps()`, which takes an explicit subject vector and avoids filename heuristics entirely.

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)
```

OrthogonalFamily	<i>Orthogonal map family helper</i>
------------------	-------------------------------------

Description

Orthogonal map family helper

Usage

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

Arguments

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

Value

A MapFamily object with orthogonal traits

OTFamily	<i>Optimal transport family helper</i>
----------	--

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

plot.gds_examination	<i>Plot a group examination</i>
----------------------	---------------------------------

Description

The default action plane places model surprise on the horizontal axis and influence on the requested group estimand on the vertical axis. Secondary views show subject-by-diagnostic summaries or the residual geometry. When subject is supplied, the retained observed, expected, predictive-residual, and deletion maps are shown for that subject.

Usage

```
## S3 method for class 'gds_examination'
plot(
  x,
  subject = NULL,
  type = c("action", "heatmap", "embedding"),
  contrast = NULL,
  estimand = NULL,
  group_by = NULL,
  assays = NULL,
  slice = NULL,
  ...
)
```

Arguments

x	A gds_examination.
subject	Optional retained subject ID. Supplying it selects the linked subject view regardless of type.
type	One of "action", "heatmap", or "embedding".
contrast	Optional contrast subset.
estimand	Optional estimand subset.
group_by	Optional column in subject_data mapped to point shape.
assays	Optional selected-subject assays to display.
slice	Optional one-based axial slice for voxel-space subject maps.
...	Reserved for future plot controls.

Value

A ggplot object.

posthoc	<i>Add a post-hoc operation to a plan</i>
---------	---

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

See Also

[posthoc-methods](#) for the built-in methods and the assays each adds.

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
```

posthoc-methods	<i>Built-in post-hoc correction methods and their output assays</i>
-----------------	---

Description

Methods are selected by name in [posthoc\(\)](#) and add their adjusted-p assays to the realised GDS. The FDR methods adjust the object's p-value assay(s): a genuine bare p (from meta/combine reducers or a tabular p column) yields a single q; a per-term `p_coef:<term>` family (from regression/LMM reducers such as `ols:voxelwise`) yields one `q_coef:<term>` per term. Pass `options = list(source = "p_coef:<term>")` to adjust one specific term.

Value

This is a documentation page (no return value).

Methods and outputs

- "fdr:bh", "fdr:by": add q (or per-term `q_coef:<term>`), Benjamini-Hochberg / Benjamini-Yekutieli adjusted p-values, computed column-wise per subject x contrast.
- "fdr:spatial": adds q (or per-term `q_coef:<term>`), group-wise spatial FDR. Requires a discrete spatial grouping (one parcel/cluster id per sample) via `options$group` or a recognised `row_data` column (`feature_group`, `spatial_group`, `group`, `parcel`, `parcel_id`).

- "nt:tfce_fwer" (needs the optional neurothresh package): adds p_fwer (TFCE FWER-corrected p-values), sig_mask (1/0 at alpha), and tfce (the TFCE-enhanced statistic map).
- "nt:cluster_fdr_perm" (needs neurothresh): adds q (cluster-FDR q broadcast to voxels) and sig_mask.

Required input assays are auto-derived (and persisted) when absent: e.g. fdr:bh on a z-only GDS adds a derived p alongside q.

See Also

[posthoc\(\)](#), [list_posthoc\(\)](#), [list_assays\(\)](#)

```
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.

read_catalog	<i>Read catalog from JSON file</i>
--------------	------------------------------------

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

```
# Build a tiny study tree with empty placeholder files
root <- tempfile("study_")
for (s in c("01", "02")) {
  d <- file.path(root, paste0("sub-", s))
  dir.create(d, recursive = TRUE)
  invisible(file.create(file.path(d, c("cope1.nii.gz", "varcope1.nii.gz"))))
}
catalog <- image_catalog(root, "sub-*/*.nii.gz",
  path_regex = "sub-(?<subject>[^/]+)")

manifest <- tempfile(fileext = ".json")
write_catalog(catalog, manifest)

catalog2 <- read_catalog(manifest)
catalog2

unlink(c(manifest, root), recursive = TRUE)
```

reduce	<i>Reduce across subjects (meta-analysis)</i>
--------	---

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", "lmm:ri_slope1", "lmm:ri_knownvar", and "lmm:ri_slope1_knownvar", plus permutation reducers "perm:onesample" and "perm:twosample". |
| weights | Weighting scheme ("1/var", "n_eff", "equal", "custom"). Registered reducers apply their own weighting, so only the default "1/var" is accepted for every reducer. "equal" is accepted for unweighted and evidence-combining reducers (e.g. "ols:voxelwise", "stouffer"); other schemes raise an error unless the reducer lists them in <code>options_schema\$weights</code> . Unknown method names also raise an error. |
| by | Grouping variable (e.g., "contrast") |
| options | <p>Additional reducer options. Common examples include <code>tau2 = NULL</code>, <code>custom_weights</code>, or for restricted LMM reducers <code>list(fit = "REML", theta_mode = "pooled")</code> plus <code>list(slope = "time", covariance = "diag")</code> for "lmm:ri_slope1".</p> <p>Note: Current grouping is per-contrast. Values other than "contrast" are not interpreted yet; future versions may extend grouping to other factors in <code>row_data/col_data</code>.</p> <p>Restricted repeated-measures LMM contract:</p> <ul style="list-style-type: none"> • "lmm:ri" fits a Gaussian random-intercept model. • "lmm:ri_slope1" fits a Gaussian random intercept plus one within-subject slope. • The corresponding <code>*_knownvar</code> methods add the known diagonal sampling variance from <code>var</code> plus an estimated residual heterogeneity component. • Repeated-measure metadata must live on the contrast axis via with_contrast_data() or tabular ingestion with <code>gds(..., contrast_data_cols = ...)</code>. • Unweighted methods allow <code>theta_mode = "pooled"</code> or "voxelwise"; known-variance methods currently require "voxelwise". • The supported family is intentionally narrow: one grouping factor only, Gaussian responses only, and no general random-effects formula grammar. <p>Permutation reducer contract:</p> <ul style="list-style-type: none"> • "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-ltl family-wise p_fwer.
formula	Optional model formula. For meta-regression reducers the design is built from subject-level col_data. For repeated-measures LMM reducers, the formula specifies fixed effects at the subject-by-repeat level and must not contain lmer-style random-effects syntax.
data	Optional subject-keyed data frame to attach as col_data before execution-time construction of the model matrix.

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()` 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

Output assays

Reducers collapse the subject axis and write group-level assays (named on the realised GDS, retrievable with `assays()/assay()`):

- "ols: voxelwise": per-term coef: <term>, se_coef: <term>, t_coef: <term>, p_coef: <term>, plus sigma2, df_res, n_obs (see `reducer-ols-voxelwise`).
- "fixed"/"meta: fe" and "random"/"meta: re": beta_g, var_g, se_g, z_g, p_g, Q, I2, n_eff (random-effects also adds tau2). n_eff is the number of finite subject-level inputs used for each sample and contrast.
- "meta: fe_reg"/"meta: re_reg": per-term coef: <term>, se_coef: <term>, normal-approximation z_coef: <term> and p_coef: <term>, plus Q, df_res (random-effects also adds tau2). "meta: re_reg" uses the DerSimonian–Laird meta-regression denominator $\text{tr}\{W - WX(X^TWX)^{-1}X^TW\}$ and excludes rows with non-finite effects/design values or non-finite, nonpositive variances.

- "stouffer"/"combine:stouffer": z_g, p_g.
- "fisher"/"combine:fisher" and "combine:lancaster": chi2, df, p_g.
- "perm:onesample"/"perm:twosample": beta_g, se_g, t_g, df, p_g, p_perm, p_fwer.

See Also

[gds\(\)](#), [compute\(\)](#), [as_plan\(\)](#), [new_gds\(\)](#), [gds_source\(\)](#)

reducer-lmm

Restricted repeated-measures Gaussian LMM reducers

Description

fmrigds includes four 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.
- method = "lmm:ri_knownvar" and "lmm:ri_slope1_knownvar" fit the corresponding variance-aware models using the input var assay as known diagonal sampling variance.

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
- unweighted reducers support `theta_mode = "pooled"` for one shared random-effect-to-residual covariance shape across samples or `theta_mode = "voxelwise"` for sample-specific parameters
- known-variance reducers currently support voxelwise covariance estimation only
- no crossed random effects and no general `lmer()` random-effects grammar

Value

This is a documentation page (no return value); these reducers are invoked via [reduce\(\)](#).

Input variance contract

The unweighted lmm:ri and lmm:ri_slope1 reducers require beta only. An input var or se assay is ignored by their likelihood. Their sigma2 and vc_* outputs are second-level variance components estimated from the supplied effects.

The *_knownvar reducers require aligned beta and strictly positive var assays. For each sample, they fit marginal covariance $\text{diag}(\text{var}) + \text{vc_resid} * I + Z G Z'$. Thus, var is treated as known sampling variance, vc_resid (sigma2) is additional observation-level heterogeneity, and G contains the subject random-effect covariance. Only diagonal sampling covariance is currently accepted; correlated first-level estimates require a future full sampling-covariance input rather than pretending their marginal variances are independent.

Fixed-effect standard errors use the fitted covariance as a plug-in estimate, and p_coef:* uses per-coefficient containment degrees of freedom (reported in df_coef:>): coefficients of between-subject columns use $N - \text{rank}(X_{\text{between}})$, all others the residual df $N * K - p$. Satterthwaite and Kenward–Roger corrections are not applied.

Output assays

Both reducers expand fixed effects per model term (<term> = design-matrix column names of formula): coef:<term>, se_coef:<term>, t_coef:<term>, p_coef:<term>. In addition:

- lmm:ri provides sigma2, vc_intercept, vc_resid, df_res, logLik, converged (1/0), and lambda (random-intercept-to-residual variance ratio).
- lmm:ri_slope1 provides sigma2, vc_intercept, vc_slope, vc_cov_intercept_slope, vc_resid, df_res, logLik, converged, lambda_intercept, lambda_slope, lambda_cov_intercept_slope, and corr_intercept_slope.
- lmm:ri_knownvar provides sigma2, vc_intercept, vc_resid, and sampling_var_mean; lmm:ri_slope1_knownvar additionally provides vc_slope, vc_cov_intercept_slope, and corr_intercept_slope. The relative lambda_* assays are intentionally omitted because known sampling variance breaks the single residual-scale parameterization. Pass these names verbatim (including the colon and any parentheses) to [write_nifti_assays\(\)](#) / [write_out\(\)](#).

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
  )
)

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")

rm_df_known <- transform(rm_df, var = 0.04)
g_known <- gds(rm_df_known, contrast_data_cols = "time") |>
  reduce(
    method = "lmm:ri_slope1_knownvar",
    formula = ~ time,
    options = list(slope = "time", covariance = "diag", fit = "REML")
  ) |>
  compute()

```

Degrees of freedom (between/within)

Fixed-effect tests use per-coefficient degrees of freedom, exposed as `df_coef:<term>` assays (the scalar `df_res` assay is unchanged and still reports $N \times K - p$). A design column that is constant within every subject (a between-subject term, including the intercept) uses $df = N - \text{rank}(X_{\text{between}})$, where X_{between} holds all between-subject columns at one row per subject. All other (within-subject) columns use the residual $N \times K - p$. `p_coef:<term>` is computed from `df_coef:<term>`. This supersedes the single residual-df rule described above for between-subject terms; it is a containment-style approximation, not Satterthwaite or Kenward–Roger.

`converged` reflects the covariance-parameter optimizer: it is 0 when the optimizer reports failure or a variance parameter is pinned at its upper search bound (a diverging variance), and 1 otherwise (zero-variance boundary estimates count as converged).

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)
- `n_obs` (number of finite subjects contributing at each sample)

Optionally, a packed triangular covariance for the coefficient vector can be attached per contrast using `options = list(return_cov = "tri")`, which also promotes per-pair `cov:<term_i>:<term_j>` assays. Retrieve the packed form from a realised GDS via `coef_cov_tri()`.

Missing data are handled by per-sample listwise deletion: subjects whose effect is non-finite at a given sample are dropped for that sample only and the model is refit on the remaining rows. Samples with fewer than $p + 1$ finite observations (override with `options = list(min_subjects = ...)`) cannot be estimated and are returned as NA; `n_obs` records the effective sample size and a summary warning is emitted when any sample is reduced.

Value

This is a documentation page (no return value); the reducer is invoked via `reduce()`, `one_sample()`, or `group_ols()`.

Output assays

For an intercept-only fit (`formula = ~ 1`): `coef:(Intercept)`, `se_coef:(Intercept)`, `t_coef:(Intercept)`, `p_coef:(Intercept)`, plus `sigma2`, `df_res`, and `n_obs`. Each additional model term adds a matching `coef:<term>/se_coef:<term>/t_coef:<term>/p_coef:<term>` quartet. Pass these names verbatim (including the colon and any parentheses) to `write_nifti_assays()` / `write_out()`.

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	<i>Register a storage adapter</i>
------------------	-----------------------------------

Description

Register a storage adapter

Usage

```
register_adapter(
    name,
    detect,
    open,
    probe,
    read,
    close,
    capabilities = NULL,
    scan = NULL,
    ...
)
```

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
capabilities	Optional scan-capability declaration. Supported fields are sample_blocks, subject_blocks, contrast_blocks, persistent_handle, preferred_axis, cheap_revisit, and requires_staging. Omitted fields default conservatively.
scan	Optional adapter-specific scan callback.
...	Additional elements stored with adapter entry

Value

Invisibly, the adapter entry

register_assay	<i>Register an assay type</i>
----------------	-------------------------------

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	<i>Register a map family on a plan or realised GDS</i>
--------------	--

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,
  case_deletion = NULL
)
```

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
case_deletion	Optional declaration of whether and how the post-hoc conclusion can be recomputed for selected case deletions.

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.

Value

Invisibly, the registered post-hoc method name.

See Also

[posthoc-methods](#) for the built-in methods and their output assays.

register_reducer	<i>Register a reducer kernel</i>
------------------	----------------------------------

Description

Register a reducer kernel

Usage

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

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

model_contract	Optional declarative description of the reducer's design, weighting, missingness, variance, estimand, and deletion semantics. Custom reducers may omit this, in which case model-conditioned examination diagnostics are reported as unavailable.
diagnostics	Optional diagnostic implementation declaration. This is a list containing fun, capabilities, and modes; fun may be NULL while a capability is declared but not yet implemented.
frame_fun	Optional feature-block kernel for fmri_frame execution. It must accept arrays, design, and options, and return aligned result assays, result-observation meta-data, and per-feature diagnostics.

Value

Invisibly, the registered reducer name.

relabel_subjects	<i>Relabel subjects in a GDS</i>
------------------	----------------------------------

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"))
```

result_frame	<i>Construct a standardized statistical result frame</i>
--------------	--

Description

Construct a standardized statistical result frame

Usage

```
result_frame(  
  assays,  
  observations,  
  features,  
  method,  
  diagnostics = list(),  
  term_data = NULL,  
  source_observation_ids = NULL,  
  metadata = list(),  
  provenance = NULL  
)
```

Arguments

assays	Named observation-by-feature result assays.
observations	Result-row metadata containing stable .obs_id values.
features	The aligned spatial feature axis.
method	Registered statistical reducer name.
diagnostics	Per-feature diagnostics.
term_data	Compiled term metadata.
source_observation_ids	Stable source observation IDs.
metadata	Additional result metadata.
provenance	Serializable derivation provenance.

Value

An fmri_frame marked with a versioned statistical-result schema.

row_data	<i>Extract row (sample) metadata from a GDS object</i>
----------	--

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	<i>Extract sample-group metadata from a GDS object</i>
---------------	--

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	<i>Extract sample labels from a GDS object</i>
---------------	--

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	<i>Save a plan to JSON</i>
-----------	----------------------------

Description

Writes the plan's source binding, metadata, and operation nodes to a JSON file. Matrices (including `Matrix` objects) are stored as numeric data with their dimensions; map families and `map_linear` objects are stored as ASCII R serializations (the same encoding used for alignments in GDS HDF5 stores). Function-valued components that cannot be represented faithfully, such as a custom `MaskPolicy()` function or a covariance provider, cause an error at save time rather than being silently dropped.

Usage

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

Arguments

plan	Plan or object coercible via <code>as_plan()</code>
file	Path to JSON file

Details

The file records a digest of the serialized plan, which `load_plan()` verifies.

Value

Invisibly returns `file`, the path of the written JSON file. The file contains the plan's source, meta-data, operation nodes and digest and can be restored with `load_plan()`.

space	<i>Extract space descriptor from a GDS object</i>
-------	---

Description

For non-GDS objects (e.g. a **neuroim2** NeuroVol/NeuroVec) this generic falls back to `neuroim2::space()` when that package is available, so that attaching **fmrigds** does not mask `neuroim2::space()` on the search path.

Usage

`space(x, ...)`

Arguments

`x` A GDS object, or any object with a space method (such as a **neuroim2** image).
`...` Additional arguments passed to methods.

Value

A space object

space_basis	<i>Create a latent basis space descriptor</i>
-------------	---

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 = "fmrigns"))
# }
```

space_parcel	Create a parcels/ROI space descriptor
--------------	---------------------------------------

Description

Create a parcels/ROI space descriptor

Usage

```
space_parcel(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_portability	<i>Classify the portability of a GDS space</i>
-------------------	--

Description

Classify the portability of a GDS space

Usage

```
space_portability(space)
```

Arguments

space	A GDS space object.
-------	---------------------

Value

One of "portable", "portable_with_external_reference", or "nonportable_function".

space_sample_labels	<i>Create a simple label space for tabular samples</i>
---------------------	--

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	<i>Subset a space by sample indices</i>
--------------	---

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	<i>Create a surface space descriptor</i>
---------------	--

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

Usage

```
space_voxel(  
  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

```
beta <- array(rnorm(5 * 4), c(5, 4, 1))
var <- array(1, c(5, 4, 1))
gds <- new_gds(list(beta = beta, var = var), space_sample_labels(letters[1:5]),
               subjects = paste0("s", 1:4), contrasts = "c1",
               col_data = data.frame(group = c("patients", "controls",
                                               "patients", "controls"),
                                     row.names = paste0("s", 1:4)))

# Split by group column in col_data
gds_by_group <- split(gds, "group")
gds_by_group$patients # GDS with only patient subjects
subjects(gds_by_group$controls)

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

subjects	<i>Extract subject identifiers from a GDS object</i>
----------	--

Description

Extract subject identifiers from a GDS object

Usage

```
subjects(x)
```

Arguments

x	A GDS object
---	--------------

Value

Character vector of subject IDs

subset.gds_plan	<i>Lazily subset a plan, source, or realised GDS</i>
-----------------	--

Description

Adds a subset_axis node to the plan. Realised [gds](#) objects and [gds_source](#) bindings are first converted with [as_plan\(\)](#), so the result is always a lazy [gds_plan](#); call [compute\(\)](#) to materialise it.

Usage

```
## S3 method for class 'gds_plan'
subset(x, sample = NULL, subject = NULL, contrast = NULL, ...)
```

```
## S3 method for class 'gds'
subset(x, sample = NULL, subject = NULL, contrast = NULL, ...)
```

```
## S3 method for class 'gds_source'
subset(x, sample = NULL, subject = NULL, contrast = NULL, ...)
```

Arguments

x	A gds_plan, gds_source, or realised gds
sample, subject, contrast	Optional indices (positional, logical, or names) for the sample, subject, and contrast axes
...	Unused

Value

A gds_plan

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

```
# Build a tiny study tree with empty placeholder files
root <- tempfile("study_")
for (s in c("01", "02")) {
  d <- file.path(root, paste0("sub-", s))
  dir.create(d, recursive = TRUE)
  invisible(file.create(file.path(d, c("cope1.nii.gz", "varcope1.nii.gz"))))
}
catalog <- image_catalog(root, "sub-*/*.nii.gz",
  path_regex = "sub-(?<subject>[^/]+)")

catalog <- assign_meta(catalog, "stat_type", "^[a-z+][0-9]+")

# Keep only files from subject "01"
sub_catalog <- subset(catalog, subject == "01")
sub_catalog

# Multiple conditions
sub_catalog <- subset(catalog, subject %in% c("01", "02") & stat_type == "cope")
sub_catalog$metadata$basename

unlink(root, recursive = TRUE)
```

summary.gds_examination
Summarize a group examination

Description

Summarize a group examination

Usage

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

Arguments

object	A gds_examination.
...	Unused.

Value

A compact summary.gds_examination 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	<i>Specify how uncertainty is propagated through a map</i>
-----------------	--

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	<i>Get unique values of a metadata column</i>
----------------------	---

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	<i>Unregister a post-hoc method</i>
--------------------	-------------------------------------

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	<i>Convenience helper to use a stored weight array for reduction</i>
------------	--

Description

Custom weights are currently unsupported by the built-in registered reducers; passing the result to `reduce()` with such a reducer raises an informative error instead of silently running an inverse-variance analysis. Only reducers that declare "custom" in `options_schema$weights` accept it.

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	<i>Validate a GDS or plan</i>
----------	-------------------------------

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	<i>Validate catalog consistency</i>
------------------------	-------------------------------------

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

```
# Build a tiny study tree with empty placeholder files
root <- tempfile("study_")
for (s in c("01", "02")) {
  d <- file.path(root, paste0("sub-", s))
  dir.create(d, recursive = TRUE)
  invisible(file.create(file.path(d, c("cope1.nii.gz", "varcope1.nii.gz"))))
}
catalog <- image_catalog(root, "sub-*/*.nii.gz",
  path_regex = "sub-(?<subject>[^/]+)")

report <- validate(catalog)
print(report)

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

# Remove one file to see an inconsistency being reported
invisible(file.remove(file.path(root, "sub-02", "varcope1.nii.gz")))
catalog <- image_catalog(root, "sub-*/*.nii.gz",
  path_regex = "sub-(?<subject>[^/]+)")
validate(catalog)

unlink(root, recursive = TRUE)
```

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	<i>Attach subject-level covariates to a plan or GDS</i>
---------------	---

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	<i>Attach contrast-level metadata to a plan or GDS</i>
--------------------	--

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	<i>Attach sample-level metadata to a plan or GDS</i>
---------------	--

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	<i>Write catalog to JSON file</i>
---------------	-----------------------------------

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

```
# Build a tiny study tree with empty placeholder files
root <- tempfile("study_")
for (s in c("01", "02")) {
  d <- file.path(root, paste0("sub-", s))
  dir.create(d, recursive = TRUE)
  invisible(file.create(file.path(d, c("cope1.nii.gz", "varcope1.nii.gz"))))
}
catalog <- image_catalog(root, "sub-*/*.nii.gz",
  path_regex = "sub-(?<subject>[^/]+)")

manifest <- tempfile(fileext = ".json")
write_catalog(catalog, manifest)
file.exists(manifest)

unlink(c(manifest, root), recursive = TRUE)
```

write_nifti_assays	<i>Write image-like GDS assays as NIfTI files</i>
--------------------	---

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

<code>g</code>	A realised gds with voxel space.
<code>out_dir</code>	Output directory.
<code>prefix</code>	Optional filename prefix.
<code>assays</code>	Character vector of assays to write. Defaults to all assays.
<code>subjects</code>	Optional subject labels to write. Defaults to all subjects.

contrasts	Optional contrast labels to write. Defaults to all contrasts.
sanitize	Logical; if TRUE (default), make filename components filesystem-friendly. Set FALSE only with trusted prefix/labels: raw values are used verbatim, so path separators in them could write outside out_dir.
overwrite	Logical; if FALSE, existing files are skipped.

Details

Filenames are [`<prefix>_`]`contrast-<contrast>[_sub-<subject>]_<assay>.nii.gz`. The `contrast-` token is always included so that writing several contrasts to the same directory/prefix (e.g. one call per contrast) never silently overwrites; the `sub-` token is added when the GDS has more than one subject. Existing BIDS-style labels are not double-prefixed (a "sub-01" label stays sub-01, not sub-sub-01). A within-call collision (two distinct outputs resolving to the same path, e.g. via sanitisation) raises an error.

Value

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

Examples

```
if (requireNamespace("neuroim2", quietly = TRUE) &&
    requireNamespace("RNifti", quietly = TRUE)) {
  # Write tiny 4x4x2 subject-level metric maps for four subjects
  dir <- tempfile("scalar_maps_")
  dir.create(dir)
  subj <- c("sub-01", "sub-02", "sub-03", "sub-04")
  files <- file.path(dir, paste0(subj, "_auc.nii.gz"))
  for (f in files) RNifti::writeNifti(array(rnorm(32), c(4, 4, 2)), f)
  participants <- data.frame(subject = subj,
                             group = c("control", "control", "patient", "patient"))

  g <- suppressWarnings(gds_from_scalar_maps(files, subject = subj,
                                             contrast = "auc",
                                             col_data = participants))

  fit <- group_ols(g, ~ group) |> compute()
  manifest <- write_nifti_assays(fit, file.path(dir, "group_maps"),
                                prefix = "auc",
                                assays = c("coef:grouppatient", "t_coef:grouppatient"))
  print(manifest[, c("assay", "written")])
  print(basename(manifest$path))

  unlink(dir, recursive = TRUE)
}
```

write_out	<i>Declare an output target for a plan</i>
-----------	--

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 <code>as_plan()</code>
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

write_report	<i>Write a self-contained group-examination report</i>
--------------	--

Description

The report uses inline CSS and SVG only. Its action plane and residual geometry link to retained subject drill-down sections; it has no external JavaScript, font, image, or stylesheet dependency.

Usage

```
write_report(
  x,
  file,
  title = "Group examination",
  subjects = NULL,
  overwrite = FALSE,
  ...
)
```

Arguments

<code>x</code>	A <code>gds_examination</code> .
<code>file</code>	Output HTML path.
<code>title</code>	Report title.
<code>subjects</code>	Optional retained subject IDs to include. By default, all subjects with retained maps are included.
<code>overwrite</code>	Whether an existing file may be replaced.
<code>...</code>	Reserved for future report controls.

Value

The normalized output path, invisibly.

Index

- * **mixed-models**
 - reducer-lmm, [69](#)
- * **models**
 - reducer-lmm, [69](#)
 - reducer-ols-voxelwise, [71](#)
- * **reduce**
 - reducer-lmm, [69](#)
 - reducer-ols-voxelwise, [71](#)
- * **regression**
 - reducer-ols-voxelwise, [71](#)
- [align](#), [6](#)
- [alignment-registry](#), [6](#)
- [apply_common_mask](#), [7](#)
- [as_fmri_frame.gds](#), [7](#)
- [as_gds](#), [8](#)
- [as_gds\(\)](#), [8](#)
- [as_gds.DenseNeuroVec](#) ([as_gds.NeuroVec](#)), [12](#)
- [as_gds.DenseNeuroVol](#) ([as_gds.NeuroVol](#)), [13](#)
- [as_gds.fmri_frame](#), [10](#)
- [as_gds.image_catalog](#), [11](#)
- [as_gds.image_catalog\(\)](#), [53](#)
- [as_gds.NeuroVec](#), [12](#)
- [as_gds.NeuroVol](#), [13](#)
- [as_neurovec](#), [14](#)
- [as_neurovol_list](#), [16](#)
- [as_neurovol_list\(\)](#), [30](#)
- [as_plan](#), [17](#)
- [as_plan\(\)](#), [58](#), [63](#), [64](#), [68](#), [69](#), [80](#), [87](#), [98](#)
- [as_scalar_map_gds](#)
 - ([gds_from_scalar_maps](#)), [38](#)
- [assay](#), [17](#)
- [assay\(\)](#), [48](#), [49](#), [68](#)
- [assay_info](#), [18](#)
- [assay_info\(\)](#), [49](#)
- [assays](#), [18](#)
- [assays\(\)](#), [49](#), [68](#)
- [assert_compatible_spaces](#), [19](#)
- [assign_meta](#), [19](#)
- [attach_weight](#), [20](#)
- [base::list.files\(\)](#), [31](#)
- [base::unserialize\(\)](#), [51](#)
- [coef_array](#), [21](#)
- [coef_cov_tri](#), [21](#)
- [coef_cov_tri\(\)](#), [72](#)
- [col_data](#), [22](#)
- [common_mask](#), [22](#)
- [compute](#), [23](#)
- [compute\(\)](#), [32](#), [44](#), [45](#), [68](#), [69](#), [87](#), [98](#)
- [contrast_data](#), [24](#)
- [contrasts](#), [24](#)
- [derive](#), [25](#)
- [detect_adapter](#), [25](#)
- [examination_control](#), [26](#)
- [examination_control\(\)](#), [27](#)
- [examine_group](#), [27](#)
- [examine_group\(\)](#), [26](#)
- [explain](#), [29](#)
- [explain_plan](#), [29](#)
- [extract_group](#), [30](#)
- [find_maps](#), [31](#)
- [find_maps\(\)](#), [37](#)
- [fit_group](#), [32](#)
- [fmrigds](#) ([fmrigds-package](#)), [5](#)
- [fmrigds-package](#), [5](#)
- [gds](#), [23](#), [27](#), [33](#), [39](#), [44](#), [48](#), [49](#), [67](#), [68](#), [74](#), [75](#), [87](#), [94–96](#)
- [gds\(\)](#), [11](#), [37](#), [68](#), [69](#)
- [gds_from_neurovol_nested](#), [34](#)
- [gds_from_neurovol_nested\(\)](#), [35](#)
- [gds_from_neurovols](#), [35](#)
- [gds_from_nifti_maps](#), [36](#)
- [gds_from_nifti_maps\(\)](#), [39](#)

gds_from_scalar_maps, 38
 gds_from_scalar_maps(), 59
 gds_metadata, 40
 gds_metadata(), 9, 33
 gds_plan, 27, 33, 44, 48, 49, 67, 68, 74, 87, 94, 95
 gds_source, 27, 41, 48, 49, 67, 68, 87
 gds_source(), 68, 69
 gds_to_tibble, 41
 get_adapter, 42
 get_alignment (alignment-registry), 6
 get_map_family, 42
 get_posthoc, 43
 get_reducer, 43
 group_ols, 44
 group_ols(), 44, 72
 group_plan, 45
 group_plan(), 32

 harmonise_contrasts, 46

 image_catalog, 46
 image_catalog(), 39, 58

 join_meta, 47

 list_alignments (alignment-registry), 6
 list_assays, 48
 list_assays(), 50, 64
 list_map_families, 49
 list_posthoc, 50
 list_posthoc(), 49, 64
 list_reducers, 50
 list_reducers(), 49
 load_plan, 51
 load_plan(), 80

 make_linear_family, 51
 make_warp_family, 52
 map_assays, 53
 map_assays(), 11
 map_linear, 54
 map_to, 55
 MapFamily, 6, 42, 55, 74
 mask, 56
 MaskPolicy, 57
 MaskPolicy(), 56, 80
 metadata, 57
 model_matrix, 58

 neuroim2::space(), 81
 new_gds (gds), 33
 new_gds(), 13, 14, 34, 36, 68, 69
 new_image_catalog, 58
 nifti_source, 59
 nifti_source(), 37

 one_sample (group_ols), 44
 one_sample(), 72
 OrthogonalFamily, 60
 OTFamily, 61

 plot.gds_examination, 61
 posthoc, 62
 posthoc(), 63, 64
 posthoc-methods, 50, 63, 63, 76
 preview, 64
 print.catalog_validation_report, 65
 print.gds (explain), 29
 print.gds_plan (explain), 29
 print.image_catalog, 65

 read_catalog, 66
 read_catalog(), 95
 reduce, 66
 reduce(), 20, 44, 69, 72, 91
 reducer-lmm, 69
 reducer-ols-voxelwise, 68, 71
 register_adapter, 73
 register_alignment
 (alignment-registry), 6
 register_assay, 74
 register_map, 74
 register_nftab_adapter, 75
 register_posthoc, 75
 register_reducer, 76
 relabel_subjects, 77
 result_frame, 78
 row_data, 79

 sample_groups, 79
 sample_labels, 80
 save_plan, 51, 80
 space, 81
 space_basis, 81
 space_from_nifti, 82
 space_parcels, 82
 space_portability, 83
 space_sample_labels, 83

- space_subset, [84](#)
- space_surface, [84](#)
- space_voxel, [85](#)
- split.gds, [86](#)
- split.gds(), [30](#)
- stats::as.formula, [58](#)
- stats::contrasts(), [24](#)
- subjects, [87](#)
- subset.gds(subset.gds_plan), [87](#)
- subset.gds_plan, [87](#)
- subset.gds_source(subset.gds_plan), [87](#)
- subset.image_catalog, [88](#)
- summary.gds_examination, [89](#)
- summary.image_catalog, [89](#)

- two_sample(group_ols), [44](#)

- UncertaintyRule, [54–56](#), [90](#)
- unique.image_catalog, [90](#)
- unregister_posthoc, [91](#)
- use_weight, [91](#)

- validate, [92](#)
- validate.image_catalog, [92](#)

- WarpFamily, [93](#)
- with_col_data, [94](#)
- with_col_data(), [58](#)
- with_contrast_data, [94](#)
- with_contrast_data(), [67](#), [69](#)
- with_row_data, [95](#)
- write_catalog, [95](#)
- write_catalog(), [66](#)
- write_nifti_assays, [96](#)
- write_nifti_assays(), [48](#), [49](#), [70](#), [72](#)
- write_out, [98](#)
- write_out(), [48](#), [70](#), [72](#)
- write_report, [98](#)