Package: neuroatlas (via r-universe)

June 5, 2026

Title Neuroimaging Atlases and Parcellations

Version 0.1.0.9000

Description Provides a unified interface to access and work with various neuroimaging atlases and parcellations including Schaefer, Brainnetome, Glasser, FreeSurfer ASEG, and Olsen MTL atlases. Integrates with TemplateFlow for standardized template access and supports interactive brain surface visualisation via triangle-mesh rendering with 'ggplot2' and 'ggiraph'.

License MIT + file LICENSE

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

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

Depends R (>= 4.1.0)

Imports assertthat, cli, crayon, downloader, dplyr, ggiraph, ggplot2, lifecycle, magrittr, memoise, methods, neuroim2, neurosurf, rlang, Rnanoflann, scales, scico, stringr, templateflow, tibble, tools, utils

Suggests albersdown, corpcor, DT, echarts4r, FNN, future.apply, geojsonio, geojsonsf, ggseg, ggsegSchaefer, gifti, htmltools, igraph, jsonlite, knitr, mockery, patchwork, reticulate, rmarkdown, shiny, sf, testthat (>= 3.0.0), tidyr, uwot

VignetteBuilder knitr

Remotes bbuchsbaum/neurosurf, bbuchsbaum/albersdown, bbuchsbaum/templateflow

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

LazyData true

LazyDataCompression xz

RoxygenNote 7.3.3

Config/Needs/website albersdown

Collate 'alignment_registry.R' 'all_generic.R' 'aseg_subcort.R' 'atlas.R' 'atlas_brainnetome.R' 'atlas_colors.R' 'atlas_connectivity.R' 'atlas_constructor.R' 'atlas_download.R' 'atlas_graph.R' 'atlas_hierarchy.R' 'atlas_methods.R' 'atlas_operators.R' 'atlas_overlap.R' 'atlas_provenance.R' 'atlas_ref.R' 'atlas_registry.R' 'atlas_utils.R' 'batch_reduce.R' 'ce_atlas.R' 'ce_compute.R' 'ce_overlay.R' 'ce_picking.R' 'ce_plot.R' 'ce_plugins.R' 'ce_selection.R' 'plot_brain.R' 'ce_shared_exports.R' 'chart.R' 'cluster_explorer.R' 'coordinate_spaces.R' 'data-schaefer.R' 'dilate_parcels.R' 'fsaverage.R' 'fsl_atlas.R' 'ggschaefer.R' 'glasser.R' 'neuroatlas-package.R' 'olsen_mtl.R' 'parcel_data.R' 'plot_brain_grid.R' 'print-plot-methods.R' 'query_point.R' 'reduce_atlas_vec.R' 'roi_colors.R' 'roi_metadata.R' 'schaefer.R' 'spin_test.R' 'subcortical_atlases.R' 'template_flow.R' 'transform_registry.R' 'visfatlas.R' 'visual_atlas.R' 'wang_atlas.R' 'zzz.R'

Config/pak/sysreqs libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev libglpk-dev libglu1-mesa-dev make texlive libicu-dev libpng-dev libuv1-dev libxml2-dev libgl1-mesa-dev libssl-dev libx11-dev zlib1g-dev

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

Date/Publication 2026-06-05 15:46:20 UTC

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

RemoteRef HEAD

RemoteSha d6dcb50bbde333a755db17b7d2a62b064aaa15b1

Contents

`%where%` *Filter Atlas with Infix Operator*

Description

A concise infix operator for filtering atlas regions by metadata attributes. Equivalent to calling `filter_atlas` but allows a more fluent pipe-friendly syntax.

Usage

```
atlas %where% expr
```

Arguments

`atlas`	An atlas object.
`expr`	An unquoted filter expression using column names from `roi_metadata`. Multiple conditions can be combined with `&` and `|`.

Value

A new atlas object containing only the matching ROIs.

See Also

`filter_atlas` for the underlying function, `roi_metadata` for available filter columns

Examples

```
## Not run:
atlas <- get_schaefer_atlas(parcels = "200", networks = "7")

# Filter to left-hemisphere Default network
sub <- atlas %where% (network == "Default" & hemi == "left")

# Filter by label pattern
vis <- atlas %where% grepl("Vis", label)

## End(Not run)
```

as_igraph *Convert Atlas Connectivity to igraph*

Description

Convert Atlas Connectivity to igraph

Usage

```
as_igraph(x, ...)

## S3 method for class 'atlas_connectivity'
as_igraph(x, weighted = TRUE, ...)
```

Arguments

x	An atlas_connectivity matrix.
...	Additional arguments (currently ignored).
weighted	Logical. If TRUE (default), edge weights are the correlation values. If FALSE, a binary adjacency is used.

Value

Dispatches to methods.

as_parcel_data *Convert an Object to 'parcel_data'*

Description

Convert an Object to 'parcel_data'

Usage

```
as_parcel_data(x, ...)

## S3 method for class 'parcel_data'
as_parcel_data(x, ...)

## S3 method for class 'atlas'
as_parcel_data(
  x,
  values = NULL,
  value_col = "value",
  atlas_id = NULL,
```

```
  atlas_version = NULL,
  atlas_space = NULL,
  schema_version = "1.0.0",
  ...
)

## Default S3 method:
as_parcel_data(x, ...)
```

Arguments

x	Object to convert.
...	Additional arguments passed to methods.
values	Optional values to attach to parcel rows. - numeric/integer vector of length ‘length(x$ids)‘ - data frame/tibble with join column ‘id‘ or ‘label‘ and one or more value columns
value_col	Column name used when ‘values‘ is a vector.
atlas_id	Optional canonical atlas id override.
atlas_version	Optional atlas version.
atlas_space	Optional atlas space/template identifier.
schema_version	Schema version for the returned object.

Value

An object of class ‘"parcel_data"‘.

---

atlas_alignment *Atlas Alignment Lookup*

Description

Returns a structured compatibility/alignment summary between two atlas representations based on atlas provenance metadata.

Usage

```
atlas_alignment(x, y)
```

Arguments

x	Source atlas object.
y	Target atlas object.

Details

For same family/model/representation comparisons across templates, this function consults the space transform registry when available.

Value

A list of class '"atlas_alignment"' with fields: 'from', 'to', 'compatible', 'relation', 'method', 'confidence', 'status', 'requires_transform', and 'notes'.

Examples

```
ref <- new_atlas_ref(
  family = "schaefer",
  model = "Schaefer2018",
  representation = "volume",
  template_space = "MNI152NLin6Asym",
  coord_space = "MNI152",
  confidence = "high"
)
a <- structure(list(atlas_ref = ref), class = c("schaefer", "atlas"))
b <- structure(list(atlas_ref = ref), class = c("schaefer", "atlas"))
atlas_alignment(a, b)$relation
```

---

atlas_artifacts *Get Atlas Artifact Metadata*

Description

Get Atlas Artifact Metadata

Usage

```
atlas_artifacts(x, ...)
```

Arguments

x	An atlas object.
...	Additional arguments passed to methods.

Value

A tibble with one row per upstream artifact used to construct the atlas.

`atlas_connectivity` *Compute Connectivity Matrix from Atlas Parcellations*

Description

Extracts mean time-series for each atlas region from a 4D NeuroVec and computes pairwise correlations to produce a connectivity matrix.

Usage

```
atlas_connectivity(
  data_vol,
  atlas,
  method = c("pearson", "spearman", "partial"),
  threshold = NULL,
  stat_func = mean,
  ...
)
```

Arguments

`data_vol`	A `NeuroVec` (4D) containing the time-series data.
`atlas`	An atlas object defining the parcellation.
`method`	Character string specifying the correlation method: `"pearson"` (default), `"spearman"`, or `"partial"`. Partial correlation requires the **corpcor** package.
`threshold`	Optional numeric value. If supplied, entries with `abs(r) < threshold` are set to zero.
`stat_func`	Function used to summarise voxel values within each parcel (default: `mean`).
`...`	Additional arguments passed to `reduce_atlas`.

Value

A symmetric matrix of class `c("atlas_connectivity", "matrix")` with region labels as dimnames.

See Also

`reduce_atlas` for the underlying extraction, `as_igraph.atlas_connectivity` for graph conversion

Examples

```
## Not run:
atlas <- get_schaefer_atlas(parcels = "100", networks = "7")
# data_vol is a 4D NeuroVec from an fMRI run
conn <- atlas_connectivity(data_vol, atlas)
```

```
conn_sparse <- atlas_connectivity(data_vol, atlas, threshold = 0.3)

## End(Not run)
```

`atlas_coord_space` *Atlas Coordinate-Space Convenience Accessor*

Description

Atlas Coordinate-Space Convenience Accessor

Usage

```
atlas_coord_space(x)
```

Arguments

x	An atlas object.

Value

Character scalar (or 'NA_character_').

`atlas_family` *Atlas Family Convenience Accessor*

Description

Atlas Family Convenience Accessor

Usage

```
atlas_family(x)
```

Arguments

x	An atlas object.

Value

Character scalar.

atlas_graph *Compute Parcel Adjacency Graph from an Atlas*

Description

Builds a region adjacency graph from a volumetric atlas. Two parcels are considered adjacent when at least one pair of their voxels are neighbours under the chosen connectivity scheme.

Usage

```
atlas_graph(
  atlas,
  connectivity = c("6", "18", "26"),
  as = c("matrix", "igraph", "tibble"),
  include_weight = TRUE
)
```

Arguments

atlas	An atlas object (must contain $atlas, $ids, and $labels).
connectivity	Character. Neighbourhood type: "6" (face), "18" (face + edge), or "26" (face + edge + corner). Default: "6".
as	Character. Output format: "matrix" (K x K adjacency matrix), "igraph" (**igraph** graph object), or "tibble" (edge-list data frame with from, to, weight columns). Default: "matrix".
include_weight	Logical. If TRUE (default), edge values are the count of shared boundary voxel pairs. If FALSE, edges are binary (0/1).

Value

Depending on as:

matrix A symmetric integer matrix of dimension K x K with row/column names set to region labels.

igraph An undirected **igraph** graph (requires the **igraph** package).

tibble A data.frame (tibble if available) with columns from, to, and weight.

Examples

```
## Not run:
atlas <- get_schaefer_atlas(100)
adj   <- atlas_graph(atlas)
adj_b <- atlas_graph(atlas, include_weight = FALSE)
el    <- atlas_graph(atlas, as = "tibble")

## End(Not run)
```

atlas_hierarchy *Atlas Hierarchy*

Description

Extracts hierarchical level information from an atlas, describing how parcels map to higher-level groupings (networks, hemispheres).

Usage

```
atlas_hierarchy(atlas)
```

Arguments

atlas	An atlas object.

Value

A list of class "atlas_hierarchy" with components:

levels Character vector of hierarchy level names, from finest to coarsest (e.g., c("parcel", "network", "hemisphere")).

mappings Named list of named character vectors. Each element maps parcel labels to the corresponding grouping at that level. Names of the list correspond to levels[-1] (all levels above parcel).

Examples

```
## Not run:
atlas <- get_schaefer_atlas(parcels = "200", networks = "7")
h <- atlas_hierarchy(atlas)
h$levels
# [1] "parcel" "network" "hemisphere"
h$mappings$network[1:5]

## End(Not run)
```

atlas_history *Get Atlas Processing History*

Description

Get Atlas Processing History

Usage

```
atlas_history(x, ...)
```

Arguments

x	An atlas object.
...	Additional arguments passed to methods.

Value

A tibble with one row per processing step tracked by neuroatlas.

---

atlas_overlap *Cross-Atlas Overlap Analysis*

Description

Computes spatial overlap between regions of two atlases using Dice and/or Jaccard similarity coefficients. This is useful for comparing parcellations, assessing correspondence between atlases, or mapping regions from one atlas to another.

Usage

```
atlas_overlap(atlas1, atlas2, metrics = c("dice", "jaccard"), min_overlap = 0)
```

Arguments

atlas1	An atlas object (with $atlas, $ids, $labels).
atlas2	An atlas object (with $atlas, $ids, $labels).
metrics	Character vector of overlap metrics to compute. One or more of "dice" and "jaccard". Default is both.
min_overlap	Integer. Minimum number of overlapping voxels for a pair to be included in the results. Default is 0 (include all pairs with any overlap).

Value

A tibble with columns:

atlas1_id Integer region ID from atlas1

atlas1_label Character label from atlas1

atlas2_id Integer region ID from atlas2

atlas2_label Character label from atlas2

dice Dice similarity coefficient (if requested)

jaccard Jaccard similarity coefficient (if requested)

n_overlap Number of overlapping voxels

n_atlas1 Total voxels in the `atlas1` region

n_atlas2 Total voxels in the `atlas2` region

Rows are sorted by Dice coefficient descending (or Jaccard if Dice was not requested).

Examples

```
## Not run:
atlas_a <- get_schaefer_atlas(parcels = "100", networks = "7")
atlas_b <- get_schaefer_atlas(parcels = "200", networks = "7")
overlap <- atlas_overlap(atlas_a, atlas_b)
head(overlap)

## End(Not run)
```

`atlas_provenance` *Atlas Provenance Accessors*

Description

Access structured provenance metadata for atlas objects, including the canonical atlas identity, upstream artifacts, and processing history.

Usage

```
atlas_provenance(x, ...)

## S3 method for class 'atlas'
atlas_provenance(x, ...)

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

## S3 method for class 'atlas'
atlas_artifacts(x, ...)

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

## S3 method for class 'atlas'
atlas_history(x, ...)

## Default S3 method:
atlas_history(x, ...)
```

Arguments

x	An atlas object.
...	Additional arguments passed to methods.

Value

A list of class '"atlas_provenance"' with fields:

ref Canonical `atlas_ref()` identity metadata.

artifacts A tibble describing upstream files/resources.

history A tibble describing processing steps applied in `neuroatlas`.

`atlas_ref` *Atlas Reference Accessor*

Description

Returns the canonical atlas reference metadata for an atlas object.

Usage

```
atlas_ref(x, ...)

## S3 method for class 'atlas'
atlas_ref(x, ...)

## Default S3 method:
atlas_ref(x, ...)
```

Arguments

x	An atlas object.
...	Additional arguments passed to methods.

Value

An object of class '"atlas_ref"'.

atlas_roi_colors *Assign optimal colours to atlas regions*

Description

Bridge between an atlas object and the `roi_colors_*()` family of colour algorithms. Extracts ROI centroids, assembles a metadata tibble, and dispatches to the requested algorithm.

Usage

```
atlas_roi_colors(atlas, method = "rule_hcl", ...)
```

Arguments

`atlas`	An S3 atlas object (any type: Schaefer, ASEG, Olsen, etc.).
`method`	One of `"rule_hcl"` (default), `"network_harmony"`, `"maximin_view"`, `"embedding"`, or a named character vector of hex colours keyed by region ID.
`...`	Additional arguments passed to the underlying `roi_colors_*()` function.

Details

When `method` is a character vector of hex colours (named by region ID or in the same order as `atlas$ids`), no colour algorithm is invoked and the colours are returned directly.

Value

A `tibble` with columns `id` (integer) and `color` (hex string), one row per region in the atlas.

Examples

```
atlas <- get_aseg_atlas()
cols <- atlas_roi_colors(atlas)
head(cols)
```

atlas_space *Atlas Template-Space Convenience Accessor*

Description

Atlas Template-Space Convenience Accessor

Usage

```
atlas_space(x)
```

Arguments

x	An atlas object.

Value

Character scalar (or 'NA_character_').

`atlas_transform_manifest` *Atlas Transform Manifest*

Description

Returns currently known cross-representation alignment routes and their implementation status.

Usage

```
atlas_transform_manifest(scope = c("alignment", "space"))
```

Arguments

scope	Manifest scope. '"alignment"' returns family/model alignment routes. '"space"' returns template/space transform routes.

Value

A data frame manifest. For 'scope = "alignment"' this is a tibble of atlas-family representation routes. For 'scope = "space"' this is the space transform registry returned by [space_transform_manifest()].

Examples

```
# Atlas-family alignment routes
atlas_transform_manifest("alignment")

# Space-to-space routes
atlas_transform_manifest("space")
```

`atlas_transform_plan` *Plan a Transform Between Spaces*

Description

Computes a direct or two-hop transform plan between spaces using the packaged transform registry.

Usage

```
atlas_transform_plan(
  from_space,
  to_space,
  data_type = c("parcel", "vertex", "voxel"),
  mode = c("auto", "strict")
)
```

Arguments

`from_space`	Source space identifier.
`to_space`	Target space identifier.
`data_type`	Data type being transformed (‘"parcel"‘, ‘"vertex"‘, ‘"voxel"‘). Used for advisory warnings.
`mode`	Planning mode. ‘"auto"‘ returns ‘NULL‘ if no route exists, ‘"strict"‘ errors.

Details

Space identifiers are normalized internally, so aliases such as ‘"fslr32k"‘ are accepted.

Value

A list of class ‘"atlas_transform_plan"‘ with fields: ‘from_space‘, ‘to_space‘, ‘steps‘, ‘n_steps‘, ‘status‘, ‘confidence‘, and ‘warnings‘.

‘steps‘ is a data frame with one row per transform step and registry columns. In ‘mode = "auto"‘, returns ‘NULL‘ (with warning) if no route exists.

Examples

```
# Direct route
p1 <- atlas_transform_plan("MNI305", "MNI152")

# Alias normalization + planned route
p2 <- atlas_transform_plan("fsaverage", "fslr32k")
p2$status
```

batch_reduce *Batch Reduce Multiple Volumes by an Atlas*

Description

Applies reduce_atlas to multiple input volumes (or file paths) and combines the results into a single tibble with a subject column identifying each input.

Usage

```
batch_reduce(
  inputs,
  atlas,
  stat_func = mean,
  ...,
  format = "long",
  parallel = FALSE,
  .progress = TRUE
)
```

Arguments

inputs	A named list of inputs. Each element can be: • A NeuroVol (3D) or NeuroVec (4D) object • A character string file path (read via neuroim2::read_vol) If unnamed, subjects are auto-named "sub_001", "sub_002", etc.
atlas	An atlas object.
stat_func	Function to apply within each ROI (default: mean).
...	Additional arguments passed to reduce_atlas.
format	Character, output format passed to reduce_atlas. Default "long".
parallel	Logical. If TRUE, uses future.apply::future_lapply() for parallel processing. Requires the **future.apply** package.
.progress	Logical. If TRUE (default), displays a cli progress bar.

Value

A tibble with a subject column prepended to the reduce_atlas output for each input.

See Also

reduce_atlas for single-volume extraction

Examples

```
## Not run:
# With NeuroVol objects
vols <- list(sub01 = vol1, sub02 = vol2, sub03 = vol3)
results <- batch_reduce(vols, atlas, mean)

# With file paths
files <- list(sub01 = "path/to/sub01.nii.gz",
              sub02 = "path/to/sub02.nii.gz")
results <- batch_reduce(files, atlas, mean, parallel = TRUE)

## End(Not run)
```

---

brainnetome_labels *Brainnetome Atlas Label Table*

Description

Downloads (if needed) and returns the Brainnetome 246-region label table used by `get_brainnetome_atlas()`.

Usage

```
brainnetome_labels(use_cache = TRUE)
```

Arguments

use_cache	Logical. Use cached Brainnetome files when available.

Details

Brainnetome assets are downloaded on demand and cached locally. They are not bundled with neuroatlas; use is governed by the legal agreement on the Brainnetome download page.

Value

A tibble with one row per parcel and columns for parcel id, label, hemisphere, lobe/gyrus, Yeo network membership, RGB colour, and cytoarchitectonic description.

Source

https://atlas.brainnetome.org/download.html

`build_brain_polygon_data` *Build brain polygon render data for a surface atlas*

Description

Projects surface atlas geometry into 2D polygon panels for each hemisphere and view. Results are memoised so repeated calls with identical arguments return cached data.

Usage

```
build_brain_polygon_data(surfatlas, views, surface, projection_smooth = 0L)
```

Arguments

`surfatlas`	A surfatlas object (e.g. from `schaefer_surf()`).
`views`	Character vector of views to render (e.g. `c("lateral", "medial")`).
`surface`	Character: surface type (`"inflated"`, `"pial"`, `"white"`).
`projection_smooth`	Integer: number of Laplacian smoothing iterations on projected coordinates (default `0L`).

Value

A list with two elements:

`polygons` Tibble of 2D projected polygon vertices with columns `x`, `y`, `poly_id`, `parcel_id`, `label`, `hemi`, `view`, `panel`.

`boundaries` Tibble of boundary edges with columns `x`, `y`, `xend`, `yend`, `panel`.

`build_cluster_explorer_data` *Build Cluster Explorer Data*

Description

Compute sign-aware volumetric connected components from a statistic map, annotate them with atlas parcels, and extract cluster-level 4D signal summaries aligned to sample rows.

Usage

```
build_cluster_explorer_data(
  data_source,
  atlas,
  stat_map,
  sample_table = NULL,
  threshold = 3,
  min_cluster_size = 20,
  connectivity = c("26-connect", "18-connect", "6-connect"),
  tail = c("two_sided", "positive", "negative"),
  signal_fun = mean,
  signal_fun_args = list(na.rm = TRUE),
  series_fun = NULL,
  prefetch = TRUE,
  prefetch_max_clusters = Inf,
  prefetch_max_voxels = Inf,
  series_cache_env = NULL
)
```

Arguments

`data_source`	A sample-wise data source supporting `neuroim2::series(data_source, i)` where `i` is voxel coordinates (`n x 3` matrix) or indices. Rows of the returned matrix correspond to samples/design rows.
`atlas`	A volumetric `atlas` object used for parcel annotation. When atlas and `stat_map` dimensions differ, the atlas is automatically resampled to `stat_map` space (nearest-neighbor labels) before cluster annotation.
`stat_map`	A `NeuroVol` statistic image used for thresholding and clustering.
`sample_table`	Optional data frame with one row per sample. If `NULL`, a default table with `.sample_index` is created.
`threshold`	Numeric threshold used for cluster formation.
`min_cluster_size`	Minimum number of voxels required to keep a cluster.
`connectivity`	Connectivity passed to `neuroim2::conn_comp()`.
`tail`	Clustering mode: `"positive"`, `"negative"`, or `"two_sided"`.
`signal_fun`	Function used to summarize cluster signal per sample.
`signal_fun_args`	Named list of additional arguments passed to `signal_fun`.
`series_fun`	Optional function override for extracting voxel-wise sample series. Must accept `(data_source, i)` and return a matrix-like object with one row per sample.
`prefetch`	Logical; if `TRUE`, precompute signal summaries for all clusters on recompute.
`prefetch_max_clusters`	Maximum clusters allowed for eager prefetch. Prefetch is skipped when exceeded.

prefetch_max_voxels
: Maximum total cluster voxels allowed for eager prefetch. Prefetch is skipped when exceeded.

series_cache_env
: Optional environment used to memoize voxel-coordinates-to-series extraction across cluster computations.

Value

A list with:

cluster_table Cluster summary tibble.

cluster_parcels Cluster-parcel overlap tibble.

cluster_ts Sample-level cluster signal tibble.

cluster_voxels Named list of voxel coordinate matrices by cluster ID.

cluster_index A NeuroVol of global cluster IDs.

sample_table Normalized sample table with .sample_index.

prefetch_info List describing whether eager signal prefetch was requested/applied and the effective guard thresholds.

---

build_conflict_edges *Build slice-aware conflict edges between ROIs*

Description

Constructs a sparse, symmetric edge list describing which ROIs are likely to collide visually across common slice views. The resulting tibble records the ROI indices, names, and a normalised conflict weight in \[0, 1\].

Usage

```
build_conflict_edges(
  rois,
  id_col = "roi",
  xyz_cols = c("x", "y", "z"),
  k = 12,
  sigma_3d = NULL,
  views = c("axial", "coronal", "sagittal"),
  view_weights = c(axial = 1, coronal = 1, sagittal = 1),
  sigma_xy = 25,
  sigma_slice = 10,
  hemi_col = NULL,
  cross_hemi_factor = 0.85,
  network_col = NULL,
  diff_network_factor = 1.15,
  weight_transform = NULL
)
```

Arguments

rois	Tibble with ROI metadata. Must contain ‘id_col‘ and the three columns listed in ‘xyz_cols‘.
id_col	Column containing ROI IDs.
xyz_cols	Character vector of length three giving the coordinates to use for distance calculations.
k	Number of nearest neighbours to evaluate when forming edges.
sigma_3d	Spatial decay parameter in millimetres. Defaults to the median neighbour distance.
views	Character vector of anatomical views to consider. Supported: ‘"axial"‘, ‘"coronal"‘, and ‘"sagittal"‘.
view_weights	Named numeric vector of weights for the supplied views.
sigma_xy	In-plane decay (mm) for slice visibility.
sigma_slice	Through-slice decay (mm) for slice visibility.
hemi_col	Optional hemisphere column to down-weight cross-hemisphere conflicts.
cross_hemi_factor	Multiplicative factor (< 1) applied to conflicts involving opposite hemispheres.
network_col	Optional network column to up-weight cross-network conflicts.
diff_network_factor	Multiplicative factor (> 1) applied to conflicts involving different networks.
weight_transform	Optional function ‘function(edges, rois)‘ that can modify the weights before returning.

Value

Tibble with columns ‘from_idx‘, ‘to_idx‘, ‘from‘, ‘to‘, and ‘w‘.

Examples

```
rois <- data.frame(
  roi = 1:6,
  x = c(10, 12, 50, 52, 10, 50),
  y = c(20, 22, 20, 22, 60, 60),
  z = c(30, 32, 30, 32, 30, 30)
)
edges <- build_conflict_edges(rois, xyz_cols = c("x", "y", "z"))
```

build_surface_polygon_data *Build Surface Polygon Data for Rendering*

Description

Build 2D projected polygon/boundary data from a `surfatlas` for custom rendering workflows. This exposes the mesh-projection data pipeline used internally by `plot_brain()`.

Usage

```
build_surface_polygon_data(
  surfatlas,
  views = c("lateral", "medial"),
  surface = "inflated",
  merged = TRUE,
  projection_smooth = 0L,
  use_cache = TRUE
)
```

Arguments

`surfatlas`	A surface atlas object inheriting from class `"surfatlas"`.
`views`	Character vector of views to include. Any combination of `"lateral"`, `"medial"`, `"dorsal"`, `"ventral"`.
`surface`	Character scalar identifying the surface type label.
`merged`	Logical. If `TRUE` (default), returns merged parcel-level polygons (faster, fewer shapes). If `FALSE`, returns per-triangle polygons.
`projection_smooth`	Non-negative integer controlling Laplacian-like smoothing iterations applied to projected vertex coordinates before polygon and boundary construction. `0` (default) keeps native mesh coordinates.
`use_cache`	Logical. If `TRUE` (default), use memoized builders.

Value

A list with components:

`polygons` A tibble of projected polygon vertices.

`boundaries` A tibble of projected boundary segments.

check_templateflow *Check TemplateFlow Installation Status*

Description

Checks whether the TemplateFlow R package is installed and provides cache information.

Usage

```
check_templateflow()
```

Value

Invisible NULL. Prints status information to the console.

Examples

```
check_templateflow()
```

clear_templateflow_cache *Clear neuroatlas TemplateFlow Cache*

Description

Removes cached files and clears in-memory memoisation.

Usage

```
clear_templateflow_cache(confirm = TRUE)
```

Arguments

confirm	Logical. If 'TRUE' (the default), asks for interactive confirmation.

Value

Invisibly returns 'TRUE' if the cache was cleared, 'FALSE' if aborted.

Examples

```
## Not run:
  clear_templateflow_cache()
  clear_templateflow_cache(confirm = FALSE)

## End(Not run)
```

`cluster_explorer` *Launch Cluster Explorer Shiny App*

Description

Interactive explorer linking a cluster summary table, parcel brain map, and design-aware signal plots. Clusters are formed volumetrically from `stat_map`; visualization is parcel-level on `surfatlas`.

Usage

```
cluster_explorer(
  data_source = NULL,
  atlas = NULL,
  stat_map = NULL,
  surfatlas = NULL,
  sample_table = NULL,
  design = NULL,
  threshold = 3,
  min_cluster_size = 20,
  connectivity = c("26-connect", "18-connect", "6-connect"),
  tail = c("two_sided", "positive", "negative"),
  series_fun = NULL,
  overlay_space = NULL,
  overlay_density = NULL,
  overlay_resolution = NULL,
  overlay_fun = c("avg", "nn", "mode"),
  overlay_sampling = c("midpoint", "normal_line", "thickness"),
  prefetch = TRUE,
  prefetch_max_clusters = 200,
  prefetch_max_voxels = 1e+05,
  palette = "vik",
  selection_engine = c("cluster", "parcel", "sphere", "custom"),
  parcel_ids = NULL,
  sphere_centers = NULL,
  sphere_radius = 6,
  sphere_units = c("mm", "voxels"),
  sphere_combine = c("separate", "union"),
  selection_provider = NULL,
  analysis_plugins = NULL,
  default_analysis_plugin = "none"
)
```

Arguments

`data_source`	A sample-wise data source supporting `neuroim2::series(data_source, i)` where `i` is voxel coordinates (`n x 3` matrix) or indices. Rows of the returned matrix correspond to samples/design rows.

`atlas` A volumetric `atlas` object used for parcel annotation. When atlas and `stat_map` dimensions differ, the atlas is automatically resampled to `stat_map` space (nearest-neighbor labels) before cluster annotation.

`stat_map` A `NeuroVol` statistic image used for thresholding and clustering.

`surfatlas` A surface atlas object used by `plot_brain()`. If `NULL`, the function attempts to infer a surface atlas from a compatible 'atlas' input (for example Schaefer or Glasser). If inference is not possible, and non-demo inputs are otherwise present, input validation fails.

`sample_table` Optional data frame with one row per sample. If `NULL`, a default table with `.sample_index` is created.

`design` Optional design table (one row per sample). When provided, it is column-bound to `sample_table`.

`threshold` Numeric threshold used for cluster formation.

`min_cluster_size` Minimum number of voxels required to keep a cluster.

`connectivity` Connectivity passed to `neuroim2::conn_comp()`.

`tail` Clustering mode: `"positive"`, `"negative"`, or `"two_sided"`.

`series_fun` Optional function override for extracting voxel-wise sample series. Must accept `(data_source, i)` and return a matrix-like object with one row per sample.

`overlay_space` Optional surface space override used to fetch white/pial meshes for volumetric cluster projection. Defaults to `surfatlas$surface_space`.

`overlay_density` Optional TemplateFlow density override for overlay surface loading.

`overlay_resolution` Optional TemplateFlow resolution override for overlay surface loading.

`overlay_fun` Reduction used by `neurosurf::vol_to_surf()`.

`overlay_sampling` Sampling strategy for `neurosurf::vol_to_surf()`.

`prefetch` Logical default for eager cluster signal prefetch.

`prefetch_max_clusters` Default max cluster count allowed for prefetch.

`prefetch_max_voxels` Default max total cluster voxels allowed for prefetch.

`palette` Continuous palette passed to `plot_brain()`.

`selection_engine` Selection backend. `"cluster"` uses the built-in connected-component workflow. `"custom"` delegates data assembly to `selection_provider`.

`parcel_ids` Optional parcel ids used when `selection_engine = "parcel"`. Default `NULL` uses all parcels.

`sphere_centers` Optional sphere centers used when `selection_engine = "sphere"`. Provide numeric vector length 3 or matrix/data.frame with 3 columns.

`sphere_radius` Sphere radius used when `selection_engine = "sphere"`.

`sphere_units` Units for `sphere_centers` and `sphere_radius`: `"mm"` (world coordinates) or `"voxels"` (grid coordinates).

`sphere_combine` How to combine multiple spheres: `"separate"` (one region per sphere) or `"union"`.

`selection_provider`
Optional function used when `selection_engine = "custom"`. Must return a list compatible with `build_cluster_explorer_data()` output.

`analysis_plugins`
Optional list of analysis plugin definitions. Plugins can transform extracted time-series/design data before plotting.

`default_analysis_plugin`
Optional default plugin id. Defaults to `"none"`.

Details

Calling `cluster_explorer()` with no arguments launches a synthetic demo dataset so the UI can be explored immediately.

Value

A `shiny.appobj`.

---

coord_spaces *Standard Coordinate Space Identifiers*

Description

Character constants for standard neuroimaging coordinate spaces.

Usage

```
coord_spaces
```

Format

A named list with the following elements:

MNI305 FreeSurfer/Talairach space (fsaverage native)

MNI152 ICBM 2009c space (common fMRI template)

SCANNER Native scanner space (subject-specific)

UNKNOWN Unknown or unspecified space

Examples

```
coord_spaces$MNI305
coord_spaces$MNI152
```

coordinate_spaces *Coordinate Space Transforms for Neuroimaging Templates*

Description

Functions and constants for transforming coordinates between standard neuroimaging coordinate spaces, particularly MNI305 (fsaverage) and MNI152 (common fMRI template space).

Details

FreeSurfer's fsaverage surfaces are defined in MNI305 (Talairach-like) space, while most modern fMRI pipelines output data in MNI152 space. The difference is approximately 4mm, which matters for accurate volume-to-surface projections.

Coordinate Spaces

MNI305 FreeSurfer/Talairach space. Native space for fsaverage, fsaverage5, and fsaverage6 surfaces. Based on 305 subjects with linear registration to approximate Talairach space.

MNI152 ICBM 2009c space. The de facto standard for volumetric fMRI analysis. Used by FSL, SPM, fMRIPrep, and TemplateFlow's MNI152NLin2009cAsym template.

References

FreeSurfer CoordinateSystems documentation: https://surfer.nmr.mgh.harvard.edu/fswiki/CoordinateSystems

Wu et al. (2018). Accurate nonlinear mapping between MNI volumetric and FreeSurfer surface coordinate systems. Human Brain Mapping, 39(9), 3793-3808. doi:10.1002/hbm.24213

create_templateflow *Create a TemplateFlow Interface Object*

Description

DEPRECATED: TemplateFlow is now accessed via the pure R 'templateflow' package. No initialization is needed. Use [get_template()], [tflow_spaces()], and [tflow_files()] directly.

Usage

```
create_templateflow(cache_dir = NULL, verbosity = 0, default_template = NULL)
```

Arguments

cache_dir	Ignored. Kept for backward compatibility.
verbosity	Ignored. Kept for backward compatibility.
default_template	Ignored. Kept for backward compatibility.

Value

An S3 object of class `templateflow` (deprecated stub).

Examples

```
## Not run:
  # Deprecated. Use get_template(), tflow_spaces(), etc. directly.

## End(Not run)
```

`dilate_atlas` *Dilate Atlas Parcellation Boundaries*

Description

Expands the boundaries of brain atlas parcels by dilating them into adjacent unassigned voxels within a specified mask. This is useful for filling small gaps between parcels or extending parcels into neighboring regions.

Usage

```
dilate_atlas(atlas, mask, radius = 4, maxn = 50)
```

Arguments

`atlas`	An object of class "atlas" containing the parcellation to be dilated
`mask`	A binary mask (NeuroVol object) specifying valid voxels for dilation. Dilation will only occur within non-zero mask values. May also be a TemplateFlow space identifier (string) or list of ‘get_template()‘ arguments, which will be resolved to a NeuroVol via ‘.resolve_template_input()‘.
`radius`	Numeric. The maximum distance (in voxels) to search for neighboring parcels when dilating. Default: 4
`maxn`	Integer. Maximum number of neighboring voxels to consider when determining parcel assignment. Default: 50

Details

The dilation process:

- Identifies unassigned voxels within the mask that are adjacent to existing parcels
- For each unassigned voxel, finds nearby assigned voxels within the specified radius
- Assigns the unassigned voxel to the nearest parcel
- Respects mask boundaries to prevent dilation into unwanted regions

The function uses a k-d tree implementation (via Rnanoflann) for efficient nearest neighbor searches in 3D space.

Value

A `ClusteredNeuroVol` object containing the dilated parcellation. The object maintains the original label mappings but may include additional voxels in existing parcels.

References

The algorithm uses efficient k-d tree based nearest neighbor searches for spatial queries in 3D voxel space.

See Also

`get_template_brainmask` for creating appropriate masks from TemplateFlow

Examples

```
## Not run:
# Load an atlas
atlas <- get_aseg_atlas()

# Create or load a brain mask
mask <- get_template_brainmask()

# Dilate the atlas within the mask
dilated <- dilate_atlas(atlas, mask, radius = 4)

# More conservative dilation with fewer neighbors
dilated_conservative <- dilate_atlas(atlas, mask, radius = 2, maxn = 20)

## End(Not run)
```

---

filter_atlas *Filter Atlas by ROI Attributes*

Description

Subsets an atlas object to include only ROIs matching specified criteria. Uses non-standard evaluation (NSE) similar to `dplyr::filter()`.

Usage

```
filter_atlas(x, ..., .dots = NULL)

## S3 method for class 'atlas'
filter_atlas(x, ..., .dots = NULL)
```

Arguments

x	An atlas object
...	Filter expressions using column names from roi_metadata(x). Multiple conditions are combined with AND.
.dots	A list of quosures for standard evaluation (advanced use)

Details

The filter operation creates a new atlas containing only the specified ROIs. For volume atlases, voxels belonging to excluded ROIs are set to zero. ROI IDs are preserved (not renumbered) to maintain consistency with the original atlas labeling.

Value

A new atlas object of the same class containing only the matching ROIs. The returned atlas has updated ids, labels, hemi, orig_labels, cmap, network (if present), and roi_metadata fields. The underlying volume/surface data is also subset.

See Also

roi_metadata for viewing available filter columns, roi_attributes for listing available attributes, get_roi for extracting ROI data

Examples

```
## Not run:
# Filter Schaefer atlas to left hemisphere visual network
atlas <- get_schaefer_atlas(parcels = "200", networks = "7")
left_vis <- filter_atlas(atlas, hemi == "left", network == "Vis")

# Filter ASEG to left hemisphere structures
aseg <- get_aseg_atlas()
left_aseg <- filter_atlas(aseg, hemi == "left")

# Filter by label pattern
hippo <- filter_atlas(aseg, grepl("Hippocampus", label))

## End(Not run)
```

---

fsaverage *Surface geometry for the fsaverage6 atlas*

Description

A list including left and right hemispheres for the orig, white, inflated, and pial surfaces.

Usage

```
data(fsaverage)
```

Format

A named list with elements `lh` and `rh`, each containing surface geometry objects for orig, white, inflated, and pial surfaces.

Examples

```
data(fsaverage)
names(fsaverage)
```

---

get_aseg_atlas *Get the FreeSurfer Subcortical Atlas (ASEG)*

Description

Loads and returns the FreeSurfer subcortical segmentation (ASEG) atlas, which provides probabilistic labels for key subcortical structures in the brain. The atlas includes bilateral structures such as the thalamus, caudate, putamen, and limbic regions, as well as midline structures like the brainstem.

Usage

```
get_aseg_atlas(outspace = NULL)
```

Arguments

`outspace`	Optional `NeuroSpace` object specifying the desired output space for resampling the atlas. If NULL (default), returns the atlas in its native space.

Details

The ASEG atlas is derived from FreeSurfer's automatic subcortical segmentation algorithm and has been transformed into standard space. Each voxel contains an integer ID corresponding to a specific anatomical structure. The atlas includes major subcortical structures for both hemispheres:

- Bilateral deep gray structures (thalamus, caudate, putamen, pallidum)
- Limbic structures (hippocampus, amygdala)
- Ventral structures (nucleus accumbens, ventral diencephalon)
- Midline structures (brainstem)

Value

A list with classes 'aseg' and 'atlas' containing:

atlas A `NeuroVol` object containing the 3D volume of atlas labels

cmap A data frame with RGB color specifications for each region

ids Integer vector of region IDs present in the atlas

labels Character vector of anatomical labels corresponding to each ID

hemi Character vector indicating hemisphere ('left', 'right', or NA) for each region

References

Fischl, B., et al. (2002). Whole brain segmentation: automated labeling of neuroanatomical structures in the human brain. Neuron, 33(3), 341-355.

See Also

`map_atlas` for mapping values onto atlas regions `get_roi` for extracting specific regions of interest

Examples

```
## Not run:
# Load the atlas in native space
aseg <- get_aseg_atlas()

# View the available region labels
aseg$labels

# Get the unique region IDs
aseg$ids

## End(Not run)
```

`get_atlas` *Load an Atlas by Registered ID*

Description

Dispatches to the loader registered for ‘name‘, forwarding ‘...‘ to it. This is a thin convenience wrapper: it lets callers request an atlas by string id (e.g. ‘"schaefer"‘, ‘"glasser"‘, ‘"aseg"‘) without having to know which specific ‘get_*_atlas()‘ function to call. Aliases registered alongside the canonical id are matched case/punctuation-insensitively.

Usage

```
get_atlas(name, ...)
```

Arguments

name	Atlas id or alias; see [list_atlases()] for available choices.
...	Arguments forwarded to the registered loader function.

Value

The loaded atlas object (class depends on the loader).

See Also

[list_atlases()].

Examples

```
## Not run:
# Equivalent to get_aseg_atlas()
aseg <- get_atlas("aseg")

# Equivalent to get_schaefer_atlas(parcels = "100", networks = "7")
schaef <- get_atlas("schaefer", parcels = "100", networks = "7")

## End(Not run)
```

---

get_brainnetome_atlas *Load Brainnetome 246-Region Atlas*

Description

Downloads and loads the Brainnetome Atlas 246-region MNI152 1 mm labelmap. Files are cached under the neuroatlas user cache directory rather than bundled with the package.

Usage

```
get_brainnetome_atlas(outspace = NULL, smooth = FALSE, use_cache = TRUE)
```

Arguments

outspace	Optional NeuroSpace object or TemplateFlow-style outspace descriptor. If supplied, the atlas is resampled to that space.
smooth	Logical. Whether to smooth parcel boundaries when resampling.
use_cache	Logical. Use cached Brainnetome files when available.

Details

The Brainnetome download page describes non-commercial use and attribution terms. This loader keeps the data outside the package source and records the upstream assets in atlas_artifacts().

Value

A list with classes `c("brainnetome", "volatlas", "atlas")`.

Source

https://atlas.brainnetome.org/download.html

References

Fan, L. et al. (2016). The Human Brainnetome Atlas: A New Brain Atlas Based on Connectional Architecture. Cerebral Cortex, 26(8), 3508-3526.

`get_fsl_atlas` *FSL Atlas Loaders*

Description

Load FSL-distributed atlases into the standard neuroatlas ‘atlas‘ object shape. FSL atlases are described by XML files under ‘$FSLDIR/data/atlases‘; many probabilistic atlases provide both a 4D probability image and a 3D maximum-probability summary image. The summary image labels are offset by one relative to XML/probability-volume indices, so ‘get_fsl_atlas()‘ applies that correction when loading max-probability summaries.

Usage

```
get_fsl_atlas(
  name,
  fsl_dir = Sys.getenv("FSLDIR"),
  resolution = NULL,
  image = c("summary", "probability"),
  outspace = NULL,
  path_only = FALSE
)
```

Arguments

`name`	Atlas identifier, FSL XML path, or known alias. Known aliases include ‘"harvard_oxford_cortical"‘, ‘"harvard_oxford_subcortical"‘, ‘"harvard_oxford_cortical_subcortical"‘, and ‘"julich"‘.
`fsl_dir`	FSL installation directory. Defaults to ‘Sys.getenv("FSLDIR")‘. ‘get_julich_brain_atlas()‘ downloads an FSL-style Julich-Brain cache when this is empty and ‘download = TRUE‘.
`resolution`	Preferred image resolution, e.g. ‘"1mm"‘ or ‘"2mm"‘. If ‘NULL‘, the first image entry in the XML file is used.

image	One of ‘"summary"‘ or ‘"probability"‘. ‘neuroatlas‘ atlas objects are discrete parcellations, so ‘"summary"‘ is the default. ‘"probability"‘ currently returns paths and metadata when ‘path_only = TRUE‘; loading 4D probabilistic images as atlas objects is intentionally deferred.
outspace	Optional ‘NeuroSpace‘ to resample the discrete atlas into.
path_only	Logical; return resolved paths and parsed metadata without loading image data.

Value

An ‘atlas‘ object, or a path/metadata list when ‘path_only = TRUE‘.

`get_ggseg_atlas` *Get ggseg-Compatible Schaefer Atlas*

Description

‘r lifecycle::badge("deprecated")‘

This function has been deprecated in favour of the native `plot_brain()` renderer. Use `schaefer_surf()` to obtain a surface atlas and `plot_brain()` for visualisation.

Usage

```
get_ggseg_atlas(atlas)
```

Arguments

atlas	An atlas object containing Schaefer parcellation information.

Value

A ggseg brain atlas object for visualization (if ggsegSchaefer is installed).

Examples

```
# Deprecated — use plot_brain(schaefer_surf(200, 17)) instead
```

`get_glasser_atlas` *Load Glasser Atlas*

Description

Retrieves and loads a volumetric representation of the Glasser/HCP-MMP 1.0 cortical parcellation atlas.

Usage

```
get_glasser_atlas(outspace = NULL, source = c("xcpengine", "mni2009c"))
```

Arguments

`outspace`	Optional `NeuroSpace` object specifying desired output space. If provided, the atlas will be resampled to this space. Default: NULL
`source`	Volume source to use. One of `"xcpengine"` (default) or `"mni2009c"`.

Details

The Glasser atlas divides each hemisphere into 180 areas (360 total) based on cortical architecture, function, connectivity, and topography.

Supported sources:

- `"xcpengine"` (default): xcpEngine Glasser360 volume (`glasser360MNI.nii.gz`) with stable runtime availability but less explicit template provenance.
- `"mni2009c"`: MNI152NLin2009cAsym-provenance volume file (`MMP_in_MNI_corr.nii.gz`). This source is attempted only when requested.

If `source = "mni2009c"` is unavailable at runtime, the loader automatically falls back to `"xcpengine"` and marks confidence as `"uncertain"`. In practice, some GitHub mirrors for this source may serve a small git-annex pointer stub rather than the real NIfTI payload.

Region labels are read from the xcpEngine node-name table to provide stable parcel naming across sources.

Value

A list with class 'glasser' and 'atlas' containing:

name Character string "Glasser360"

atlas A `ClusteredNeuroVol` object containing the parcellation

cmap Data frame with RGB color specifications for each region

ids Integer vector of region IDs (1:360)

labels Character vector of anatomical labels

hemi Character vector indicating hemisphere ('left' or 'right')

Source

Source-specific links are recorded in `atlas_ref(atlas)$provenance`.

References

Glasser, M. F., et al. (2016). A multi-modal parcellation of human cerebral cortex. Nature, 536(7615), 171-178.

Examples

```
## Not run:
# Load atlas in native space
atlas <- get_glasser_atlas()

# View region labels
head(atlas$labels)

# Check number of regions per hemisphere
table(atlas$hemi)

## End(Not run)
```

`get_harvard_oxford_atlas` *Load a Harvard-Oxford Atlas*

Description

Loads Harvard-Oxford cortical, subcortical, or combined cortical/subcortical parcellations. By default this uses TemplateFlow, which does not require a local FSL installation. Set ‘source = "fsl"‘ to read from ‘$FSLDIR‘.

Usage

```
get_harvard_oxford_atlas(
  type = c("cortical", "subcortical", "cortical_subcortical"),
  threshold = c(25, 0, 50),
  template_space = "MNI152NLin6Asym",
  resolution = "01",
  source = c("templateflow", "fsl"),
  outspace = NULL,
  use_cache = TRUE,
  path_only = FALSE
)

get_harvard_oxford_cortical_atlas(...)
```

```
get_harvard_oxford_subcortical_atlas(...)

get_harvard_oxford_cortical_subcortical_atlas(...)
```

Arguments

type	One of '"cortical"', '"subcortical"', or '"cortical_subcortical"'.
threshold	Maximum-probability threshold, one of '0', '25', or '50'.
template_space	TemplateFlow space.
resolution	TemplateFlow/FSL resolution. TemplateFlow accepts '"01"' or '"02"'; FSL accepts values such as '"1mm"' and '"2mm"'.
source	'"templateflow"' or '"fsl"'.
outspace	Optional 'NeuroSpace' to resample the atlas into.
use_cache	Passed through to 'get_template()'.
path_only	Return resolved paths and metadata without loading image data.
download	Logical; for 'get_julich_brain_atlas()', download the Julich-Brain atlas archive into the neuroatlas cache when 'fsl_dir' is unset.

Value

An 'atlas' object, or path metadata when 'path_only = TRUE'.

---

get_hipp_atlas *Extract Hippocampal Parcellation*

Description

Creates a hippocampus-specific atlas from the Olsen MTL atlas, with optional anterior-posterior subdivisions.

Usage

```
get_hipp_atlas(outspace = NULL, apsections = 1)
```

Arguments

outspace	Optional NeuroSpace object for resampling
apsections	Integer specifying number of anterior-posterior divisions. Default: 1 (no subdivision)

Details

This function extracts hippocampal regions from the full MTL atlas and can subdivide them into anterior-posterior segments. The resulting atlas maintains bilateral organization and can be used for targeted hippocampal analyses.

Value

A list with class c("hippocampus", "atlas") containing:

name Character string "hippocampus"

atlas NeuroVol object with hippocampal parcellation

ids Integer vector of region IDs

labels Character vector of region labels

hemi Character vector of hemisphere designations

cmap Matrix of RGB colors for visualization

orig_labels Full labels including hemisphere information

Examples

```
## Not run:
# Basic hippocampal atlas
hipp <- get_hipp_atlas()

# With anterior-posterior subdivisions
hipp_ap <- get_hipp_atlas(apsections = 3)

## End(Not run)
```

---

get_julich_brain_atlas

Load a Julich-Brain FSL Atlas

Description

Thin wrapper around ‘get_fsl_atlas()‘ for the FSL-distributed Julich-Brain cytoarchitectonic atlas.
This requires a local FSL-style atlas directory and uses the XML/image files under ‘$FSLDIR/data/atlases‘.

Usage

```
get_julich_brain_atlas(fsl_dir = Sys.getenv("FSLDIR"), download = TRUE, ...)
```

Arguments

fsl_dir	FSL installation directory. Defaults to ‘Sys.getenv("FSLDIR")‘. ‘get_julich_brain_atlas()‘ downloads an FSL-style Julich-Brain cache when this is empty and ‘download = TRUE‘.

Value

An ‘atlas‘ object, or path metadata when ‘path_only = TRUE‘.

get_olsen_mtl *Load Olsen MTL Atlas*

Description

Loads the Olsen medial temporal lobe atlas and optionally resamples it to a different space.

Usage

```
get_olsen_mtl(outspace = NULL)
```

Arguments

outspace	Optional NeuroSpace object specifying desired output space. If NULL (default), returns atlas in native 1mm MNI space.

Value

A list with class 'atlas' containing the MTL parcellation

See Also

get_hipp_atlas for hippocampus-specific parcellation

Examples

```
# Load in native space
mtl <- get_olsen_mtl()

# Load and resample to MNI152NLin2009cAsym space (requires neuroim2)
# space <- neuroim2::read_template_space("MNI152NLin2009cAsym")
# mtl_resampled <- get_olsen_mtl(outspace = space)
```

get_roi *Extract a region of interest (ROI) from an atlas*

Description

Extracts a specific region of interest from an atlas object based on label, ID, and hemisphere information.

Usage

```
get_roi(x, label = NULL, id = NULL, hemi = NULL)

## S3 method for class 'atlas'
get_roi(x, label = NULL, id = NULL, hemi = NULL)
```

Arguments

x	An atlas object
label	Character string specifying the ROI label/name
id	Numeric ID of the ROI in the atlas
hemi	Character string specifying hemisphere ('left' or 'right')

Value

Returns a subset of the atlas containing only the specified ROI

Examples

```
## Not run:
# Load the aseg atlas
atlas <- get_aseg_atlas()

# Extract the hippocampus ROI
roi <- get_roi(atlas, label = "Hippocampus")

## End(Not run)
```

---

get_schaefer_atlas *Load Schaefer Brain Parcellation Atlas*

Description

Retrieves and loads the Schaefer brain parcellation atlas, which provides a data-driven parcellation of the cerebral cortex based on both local gradient and global similarity approaches.

Usage

```
get_schaefer_atlas(
 parcels = c("100", "200", "300", "400", "500", "600", "700", "800", "900", "1000"),
  networks = c("7", "17"),
  resolution = c("1", "2"),
  outspace = NULL,
  smooth = FALSE,
  use_cache = TRUE
)
```

```
  sy_100_7(
    resolution = "2",
    outspace = NULL,
    smooth = FALSE,
    use_cache = TRUE,
    ...
  )

  sy_100_17(
    resolution = "2",
    outspace = NULL,
    smooth = FALSE,
    use_cache = TRUE,
    ...
  )

  sy_200_7(
    resolution = "2",
    outspace = NULL,
    smooth = FALSE,
    use_cache = TRUE,
    ...
  )

  sy_200_17(
    resolution = "2",
    outspace = NULL,
    smooth = FALSE,
    use_cache = TRUE,
    ...
  )

  sy_300_7(
    resolution = "2",
    outspace = NULL,
    smooth = FALSE,
    use_cache = TRUE,
    ...
  )

  sy_300_17(
    resolution = "2",
    outspace = NULL,
    smooth = FALSE,
    use_cache = TRUE,
    ...
  )
```

```
sy_400_7(
  resolution = "2",
  outspace = NULL,
  smooth = FALSE,
  use_cache = TRUE,
  ...
)

sy_400_17(
  resolution = "2",
  outspace = NULL,
  smooth = FALSE,
  use_cache = TRUE,
  ...
)

sy_500_7(
  resolution = "2",
  outspace = NULL,
  smooth = FALSE,
  use_cache = TRUE,
  ...
)

sy_500_17(
  resolution = "2",
  outspace = NULL,
  smooth = FALSE,
  use_cache = TRUE,
  ...
)

sy_600_7(
  resolution = "2",
  outspace = NULL,
  smooth = FALSE,
  use_cache = TRUE,
  ...
)

sy_600_17(
  resolution = "2",
  outspace = NULL,
  smooth = FALSE,
  use_cache = TRUE,
  ...
)
```

```
  sy_700_7(
    resolution = "2",
    outspace = NULL,
    smooth = FALSE,
    use_cache = TRUE,
    ...
  )

  sy_700_17(
    resolution = "2",
    outspace = NULL,
    smooth = FALSE,
    use_cache = TRUE,
    ...
  )

  sy_800_7(
    resolution = "2",
    outspace = NULL,
    smooth = FALSE,
    use_cache = TRUE,
    ...
  )

  sy_800_17(
    resolution = "2",
    outspace = NULL,
    smooth = FALSE,
    use_cache = TRUE,
    ...
  )

  sy_900_7(
    resolution = "2",
    outspace = NULL,
    smooth = FALSE,
    use_cache = TRUE,
    ...
  )

  sy_900_17(
    resolution = "2",
    outspace = NULL,
    smooth = FALSE,
    use_cache = TRUE,
    ...
  )
```

```
sy_1000_7(
  resolution = "2",
  outspace = NULL,
  smooth = FALSE,
  use_cache = TRUE,
  ...
)

sy_1000_17(
  resolution = "2",
  outspace = NULL,
  smooth = FALSE,
  use_cache = TRUE,
  ...
)
```

Arguments

parcels	Character string specifying number of parcels. Options: "100", "200", "300", "400", "500", "600", "800", "1000"
networks	Character string specifying network count. Options: "7", "17"
resolution	Character string specifying MNI space resolution in mm. Options: "1", "2"
outspace	Optional NeuroSpace object for resampling the atlas
smooth	Logical. Whether to smooth parcel boundaries after resampling. Default: FALSE
use_cache	Logical. Whether to cache downloaded files. Default: TRUE
...	Additional arguments (currently unused, included for consistency with convenience functions)

Details

The Schaefer atlas offers multiple resolutions of cortical parcellation (100-1000 parcels) and two network versions (7 or 17 networks). The atlas is based on resting-state functional connectivity from 1489 subjects. Features include:

- Multiple granularity levels (100-1000 parcels)
- Network assignments (7 or 17 networks)
- Bilateral parcellation
- Available in different resolutions (1mm or 2mm)

Value

A list with classes c("schaefer", "volatlas", "atlas") containing:

name Character string identifying atlas version

atlas ClusteredNeuroVol object containing the parcellation

cmap Data frame with RGB colors for visualization

ids Integer vector of region IDs

labels Character vector of region names

orig_labels Original region labels from source data

network Network assignment for each region

hemi Hemisphere designation for each region

Convenience Functions

Shorthand functions are provided for common Schaefer atlas configurations. These functions call `get_schaefer_atlas` with the `parcels` and `networks` arguments pre-set. They all accept `resolution` (default "2"), `outspace`, `smooth`, `use_cache`, and `...` arguments.

- `sy_100_7()`: 100 parcels, 7 networks.
- `sy_100_17()`: 100 parcels, 17 networks.
- `sy_200_7()`: 200 parcels, 7 networks.
- `sy_200_17()`: 200 parcels, 17 networks.
- `sy_300_7()`: 300 parcels, 7 networks.
- `sy_300_17()`: 300 parcels, 17 networks.
- `sy_400_7()`: 400 parcels, 7 networks.
- `sy_400_17()`: 400 parcels, 17 networks.
- `sy_500_7()`: 500 parcels, 7 networks.
- `sy_500_17()`: 500 parcels, 17 networks.
- `sy_600_7()`: 600 parcels, 7 networks.
- `sy_600_17()`: 600 parcels, 17 networks.
- `sy_800_7()`: 800 parcels, 7 networks.
- `sy_800_17()`: 800 parcels, 17 networks.
- `sy_1000_7()`: 1000 parcels, 7 networks.
- `sy_1000_17()`: 1000 parcels, 17 networks.

Source

https://github.com/ThomasYeoLab/CBIG/

References

Schaefer, A., et al. (2018). Local-Global Parcellation of the Human Cerebral Cortex from Intrinsic Functional Connectivity MRI. Cerebral Cortex, 28(9), 3095-3114.

See Also

`get_schaefer_surfatlas` for surface-based version

Examples

```
## Not run:
# Load 300-parcel atlas with 7 networks
atlas <- get_schaefer_atlas(parcels = "300", networks = "7")

# Load high-resolution version
atlas_hires <- get_schaefer_atlas(parcels = "400",
                                  networks = "17",
                                  resolution = "1")

# Resample to a different space
new_space <- neuroim2::NeuroSpace(dim = c(91,109,91),
                                  spacing = c(2,2,2))
atlas_resampled <- get_schaefer_atlas(parcels = "300",
                                      outspace = new_space)

## End(Not run)
```

---

get_schaefer_surfatlas

Load Surface-Based Schaefer Atlas

Description

Loads the surface-based version of the Schaefer parcellation atlas, compatible with FreeSurfer surface representations.

Usage

```
get_schaefer_surfatlas(
  parcels = c("100", "200", "300", "400", "500", "600", "800", "1000"),
  networks = c("7", "17"),
  surf = c("inflated", "white", "pial"),
  use_cache = TRUE
)
```

Arguments

parcels	Character string specifying number of parcels. Options: "100", "200", "300", "400", "500", "600", "800", "1000"
networks	Character string specifying network count. Options: "7", "17"
surf	Character string specifying surface type. Options: "inflated", "white", "pial"
use_cache	Logical. Whether to cache downloaded files. Default: TRUE

Details

Provides the Schaefer parcellation mapped to FreeSurfer surface meshes. The atlas can be loaded onto different surface representations (inflated, white, or pial) and maintains the same parcellation scheme as the volumetric version.

Value

A list with classes c("schaefer", "surfatlas", "atlas") containing:

surf_type Surface type used

lh_atlas Left hemisphere surface atlas

rh_atlas Right hemisphere surface atlas

name Atlas identifier

cmap RGB color specifications

ids Region IDs

labels Region names

orig_labels Original region labels

network Network assignments

hemi Hemisphere designations

See Also

get_schaefer_atlas for volumetric version

Examples

```
## Not run:
# Load inflated surface atlas
surf_atlas <- get_schaefer_surfatlas(parcels = "300",
                                     networks = "7",
                                     surf = "inflated")

# Load pial surface version
pial_atlas <- get_schaefer_surfatlas(parcels = "400",
                                     networks = "17",
                                     surf = "pial")

## End(Not run)
```

`get_space_transform` *Get Transform Matrix Between Coordinate Spaces*

Description

Retrieve the affine transformation matrix for converting coordinates between two standard neuroimaging coordinate spaces.

Usage

```
get_space_transform(from, to)
```

Arguments

`from`	Character string specifying the source coordinate space. One of "MNI305", "MNI152".
`to`	Character string specifying the target coordinate space. One of "MNI305", "MNI152".

Value

A 4x4 numeric matrix representing the affine transform. Returns the identity matrix if `from == to`. Returns `NULL` with a warning if the transform is not available.

See Also

`transform_coords`, `MNI305_to_MNI152`

Examples

```
# Get the MNI305 to MNI152 transform
xfm <- get_space_transform("MNI305", "MNI152")

# Identity for same-space
id <- get_space_transform("MNI152", "MNI152")
all.equal(id, diag(4))
```

`get_subcortical_atlas` *Load harmonized subcortical atlases via TemplateFlow*

Description

Fetches one of the AtlasPack-derived subcortical atlases (CIT168, HCP thalamus, MDTB10 cerebellum, or HCP hippocampus/amygdala) using the existing TemplateFlow integration. The returned object includes the source template space and resolution so downstream workflows can track provenance.

Usage

```
get_subcortical_atlas(
  name,
  template_space = NULL,
  resolution = NULL,
  desc = NULL,
  outspace = NULL,
  use_cache = TRUE,
  path_only = FALSE,
  ...
)
```

Arguments

`name`	Atlas identifier; see `subcortical_atlas_options()`. Aliases such as "thalamus_hcp" or "cerebellum" are accepted.
`template_space`	TemplateFlow space to download (e.g., `"MNI152NLin6Asym"` or `"MNI152NLin2009cAsym"`). Defaults to the atlas-specific recommended space.
`resolution`	TemplateFlow resolution (e.g., "01"). Defaults to the atlas recommendation; validated against known options for the atlas.
`desc`	Optional TemplateFlow `desc` to override the atlas default (e.g., "LRSplit" for CIT168 hemispheric split).
`outspace`	Optional `NeuroSpace` or TemplateFlow query (string/list) to resample the atlas into a different space.
`use_cache`	Logical; pass through to `get_template`.
`path_only`	Logical; if TRUE, return paths to the atlas/label files plus metadata instead of loading into R objects.
`...`	Additional arguments forwarded to `get_template`.

Value

When `path_only=FALSE` (default), a list with classes `c("subcortical", "atlas")` containing:

- `name`: human-friendly atlas name

- atlas: ClusteredNeuroVol with parcellation labels
- ids: integer vector of region IDs
- labels: character vector of region labels
- orig_labels: same as labels
- hemi: inferred hemisphere labels when available
- cmap: optional RGB mapping if provided by the label file
- space, resolution, desc: TemplateFlow metadata
- template_atlas: TemplateFlow atlas parameter used

When path_only=TRUE, a list with class c("subcortical_paths","list") containing atlas_path, label_path, and the same metadata fields above.

Examples

```
## Not run:
# Load CIT168 in NLin6Asym space
cit <- get_subcortical_atlas("cit168", template_space = "MNI152NLin6Asym")

# Get only the file paths without loading volumes
paths <- get_subcortical_atlas("mdtb10", path_only = TRUE)

## End(Not run)
```

---

get_surface_coordinate_space

Get Coordinate Space for a Surface Template

Description

Determine which standard coordinate space a surface template's vertices are defined in.

Usage

```
get_surface_coordinate_space(template_id)
```

Arguments

template_id	Character string identifying the surface template. Examples: "fsaverage", "fsaverage5", "fsaverage6", "fsLR".

Details

FreeSurfer's fsaverage surfaces were created using Talairach registration, which targets MNI305 space. All fsaverage density variants (fsaverage, fsaverage5, fsaverage6) share this coordinate system.

The HCP's fsLR template was designed to align with MNI152 space, so no transform is needed when working with MNI152 volumetric data.

Value

Character string indicating the coordinate space:

- "MNI305" for fsaverage variants (fsaverage, fsaverage5, fsaverage6)
- "MNI152" for fsLR (HCP template)
- "Unknown" for unrecognized templates

See Also

transform_coords, get_space_transform

Examples

```
get_surface_coordinate_space("fsaverage")   # "MNI305"
get_surface_coordinate_space("fsaverage6")  # "MNI305"
get_surface_coordinate_space("fsLR")        # "MNI152"
```

---

get_template *Fetch a Template from TemplateFlow*

Description

Unified function to retrieve neuroimaging templates and related files from the TemplateFlow repository via the pure R templateflow package.

Usage

```
get_template(
  space = "MNI152NLin2009cAsym",
  variant = "brain",
  modality = "T1w",
  resolution = 1,
  cohort = NULL,
  desc = "brain",
  label = NULL,
  atlas = NULL,
  suffix = NULL,
  extension = ".nii.gz",
  path_only = FALSE,
  use_cache = TRUE,
  api_handle = NULL,
  ...
)

get_surface_template(
  template_id,
```

```
  surface_type,
  hemi,
  density = NULL,
  resolution = NULL,
  ...,
  load_as_path = TRUE
)
```

Arguments

space	Character string. The primary TemplateFlow identifier for the template space (e.g., "MNI152NLin2009cAsym"). Default: "MNI152NLin2009cAsym".
variant	Character string. A high-level descriptor for common template types. Supported: "brain" (default), "head", "mask", "probseg", "dseg". This is used to infer desc and sometimes suffix if they are not explicitly provided.
modality	Character string. The imaging modality or primary suffix for the template file. Supported: "T1w" (default), "T2w", "mask". This is used to infer suffix if not explicitly provided.
resolution	(Optional) Character string specifying the resolution, primarily for fsaverage variants (e.g., "06" for fsaverage6).
cohort	Character string. Optional cohort identifier.
desc	Character string. Specific TemplateFlow desc field. Defaults to "brain".
label	Character string. Specific TemplateFlow label field (e.g., "GM", "WM", "CSF").
atlas	Character string. Specific TemplateFlow atlas field (e.g., "Schaefer2018").
suffix	Character string. Specific TemplateFlow suffix field. Overrides any suffix inferred from modality or variant.
extension	Character string. The file extension. Default: ".nii.gz".
path_only	Logical. If TRUE, returns the file path as a string instead of loading as a NeuroVol. Default: FALSE.
use_cache	Logical. If TRUE (default), uses memoised NeuroVol loading.
api_handle	Deprecated and ignored. Kept for backward compatibility.
...	Additional arguments passed to the TemplateFlow query (e.g., hemi, density).
template_id	The main TemplateFlow template identifier for the surface (e.g., "fsLR", "fsaverage"). This is passed as the ‘space‘ argument to ‘get_template‘.
surface_type	A character string indicating the type of surface to retrieve. Common values include: "pial", "white", "inflated", "midthickness", "sphere". This is passed as the ‘suffix‘ argument to ‘get_template‘.
hemi	Character string, "L" for left hemisphere or "R" for right hemisphere. Passed as ‘hemi‘ to ‘get_template‘.
density	(Optional) Character string specifying the surface density (e.g., "32k" for fsLR, "164k" for fsaverage). Forwarded to TemplateFlow as ‘density‘.
load_as_path	Logical, whether to return only the path to the file. Defaults to ‘TRUE‘.

Details

The function performs several pre-flight checks: - Validates the existence of the specified `space`. - Validates the specified `resolution` against available resolutions.

Caching behaviour: - The `templateflow` R package maintains its own disk cache. - `NeuroVol` loading is memoised at the R session level to avoid re-reading large NIfTI files.

Value

If any of `space`, `variant`, `modality`, `resolution`, or `label` are vectors of length > 1, a named list of results is returned. Otherwise a single `neuroim2::NeuroVol` or file path string.

If 'load_as_path' is 'TRUE', a character string (path). If 'load_as_path' is 'FALSE', the result of 'as_neurovol'.

Examples

```
## Not run:
  # Get default MNI T1w brain template
  mni_brain <- get_template()

  # Vectorized: Get MNI brain and mask variants
  mni_variants <- get_template(variant = c("brain", "mask"))

  # Path only
  path <- get_template(path_only = TRUE)

## End(Not run)

  # Get the pial surface for the left hemisphere of fsLR 32k template (as path)
  # fslr_pial_L_path <- get_surface_template(template_id = "fsLR", surface_type = "pial",
  #                                          hemi = "L", density = "32k")
  # print(fslr_pial_L_path)
```

---

get_visfatlas *visfAtlas Probabilistic Functional Visual Atlas (volume)*

Description

Load the Rosenke et al. (2021) probabilistic functional atlas of human occipito-temporal visual cortex ("visfAtlas") as a volumetric MNI atlas. The maximum-probability labelmap contains 33 regions spanning the early retinotopic areas (`v1d`, `v2d`, `v3d`, `v1v`, `v2v`, `v3v`), motion-selective `hMT`, and category-selective regions for faces, bodies, characters, and places.

Usage

```
get_visfatlas(outspace = NULL, smooth = FALSE, use_cache = TRUE)
```

Arguments

outspace	Optional NeuroSpace object (or TemplateFlow-style descriptor) to resample the atlas into.
smooth	Logical. Whether to smooth parcel boundaries when resampling.
use_cache	Logical. Whether to use cached downloads. Default TRUE.

Details

The atlas is distributed as a single archive (visfAtlas.zip, ~70 MB) containing FreeSurfer, BrainVoyager, and NIfTI representations. This loader downloads the archive on demand, extracts the volumetric maximum-probability map (visfAtlas_MNI152_volume.nii.gz, 1 mm), and caches both under the neuroatlas user cache directory. Region intensities (1-33) follow the distributed FSL atlas specification; region names use the lower/upper-case source labels with hemisphere prefixes lh_/rh_.

The volume is a single-subject MNI-space grid (182 x 218 x 182, 1 mm); the publication aligns it to the MNI colin27 brain.

Note that the visfAtlas defines V1-V3 (dorsal and ventral) but not hV4; for V4 see get_wang_atlas or get_visual_atlas.

Value

A list with classes c("visfatlas", "volatlas", "atlas").

Source

https://download.brainvoyager.com/data/visfAtlas.zip

References

Rosenke, M., van Hoof, R., van den Hurk, J., Grill-Spector, K., & Goebel, R. (2021). A Probabilistic Functional Atlas of Human Occipito-Temporal Visual Cortex. Cerebral Cortex, 31(1), 603-619. doi:10.1093/cercor/bhaa246

See Also

get_wang_atlas, get_visual_atlas.

Examples

```
## Not run:
visf <- get_visfatlas()
table(visf$hemi)
get_roi(visf, label = "lh_v1d_retinotopic")

## End(Not run)
```

`get_visual_atlas` *Early Visual Cortex Atlas (V1-V5, cytoarchitectonic)*

Description

A convenience loader that extracts the early visual areas from the Julich-Brain cytoarchitectonic atlas and relabels them as `V1-V5` per hemisphere. This provides a compact, probabilistically-derived early-visual parcellation in MNI volume space without the surrounding whole-brain regions.

Usage

```
get_visual_atlas(
  outspace = NULL,
  smooth = FALSE,
  resolution = NULL,
  fsl_dir = Sys.getenv("FSLDIR"),
  download = TRUE
)
```

Arguments

`outspace`	Optional `NeuroSpace` object to resample the atlas into.
`smooth`	Logical. Whether to smooth parcel boundaries when resampling.
`resolution`	Optional Julich-Brain resolution (e.g. `"1mm"` or `"2mm"`); passed to `get_julich_brain_atlas()`.
`fsl_dir`	FSL installation directory. Defaults to `Sys.getenv("FSLDIR")`; when empty and `download = TRUE` the Julich-Brain FSL cache is downloaded.
`download`	Logical. Download the Julich-Brain FSL cache when `fsl_dir` is unset.

Details

The source regions are the Julich-Brain maximum-probability labels `GM Visual cortex V1 BA17`, `V2 BA18`, `V3V`, `V4`, and `V5` (left/right), loaded via `get_julich_brain_atlas()`. Note that the Julich atlas defines only the ventral subdivision of V3 (`V3V`) and a single `V4`/`V5` region per hemisphere.

For the topographic surface atlas with dorsal/ventral subdivisions and `hV4`, see `get_wang_atlas`; for a volumetric functional atlas, see `get_visfatlas`.

Value

A list with classes `c("visual", "volatlas", "atlas")` containing the V1-V5 regions per hemisphere.

References

Amunts, K., Mohlberg, H., Bludau, S., & Zilles, K. (2020). Julich-Brain: A 3D probabilistic atlas of the human brain's cytoarchitecture. Science, 369(6506), 988-992. doi:10.1126/science.abb4588

See Also

get_wang_atlas, get_visfatlas, get_julich_brain_atlas.

Examples

```
## Not run:
v <- get_visual_atlas()
v$labels
get_roi(v, label = "V1", hemi = "left")

## End(Not run)
```

---

get_wang_atlas *Wang (2015) Probabilistic Visual Topography Atlas*

Description

Load the Wang et al. (2015) probabilistic atlas of visual topographic areas as a pair of neurosurf LabeledNeuroSurface objects on the FreeSurfer fsaverage surface. The atlas defines 25 topographic areas per hemisphere, covering the early visual areas (V1v, V1d, V2v, V2d, V3v, V3d, hV4) plus ventral, lateral, dorsal, and parietal maps (VO1/2, PHC1/2, LO1/2, TO1/2, V3A/B, IPS0-5, SPL1, FEF).

Usage

```
get_wang_atlas(
  surf = c("inflated", "pial", "white", "midthickness"),
  space = "fsaverage",
  use_cache = TRUE
)
```

Arguments

surf	Surface type. One of "inflated" (default), "pial", "white", or "midthickness".
space	Surface space / mesh template. Only "fsaverage" (164k vertices) is supported, matching the native atlas resolution.
use_cache	Logical. Whether to use cached downloads. Default TRUE.

Details

The surface labels are the maximum-probability map (MPM) derived from 53 subjects, distributed as fsaverage (164k vertices) FreeSurfer overlay files (lh/rh.wang15_mplbl.v1_0.mgz) bundled with the **neuropythy** library. Files are downloaded on demand from the neuropythy GitHub repository and cached under the neuroatlas user cache directory.

Surface geometry is obtained from TemplateFlow via get_surface_template, so a working TemplateFlow setup is required (mirroring glasser_surf). The full per-area probability maps are also available from the original Princeton distribution and may be added in a future release.

Value

A list with classes `c("wang", "surfatlas", "atlas")` containing `lh_atlas`/`rh_atlas` (`LabeledNeuroSurface` objects), `ids`, `labels`, `hemi`, `cmap`, and the standard atlas provenance metadata.

Source

https://github.com/noahbenson/neuropythy (bundled Wang 2015 atlas); original distribution at https://scholar.princeton.edu/napl/resources.

References

Wang, L., Mruczek, R. E. B., Arcaro, M. J., & Kastner, S. (2015). Probabilistic Maps of Visual Topography in Human Cortex. Cerebral Cortex, 25(10), 3911-3931. doi:10.1093/cercor/bhu277

See Also

`get_wang_prob_atlas` for the full per-area probability volumes, `get_visfatlas` for a volumetric visual-cortex atlas, `get_visual_atlas` for cytoarchitectonic V1-V5.

Examples

```
## Not run:
# Wang 2015 visual topography atlas on the fsaverage inflated surface
wang <- get_wang_atlas(surf = "inflated")
wang$labels
get_roi(wang, label = "hV4")

## End(Not run)
```

`get_wang_prob_atlas` *Wang (2015) Full Per-Area Probability Volumes (Princeton ProbAtlas_v4)*

Description

Resolve (and optionally load) the full per-area probability maps and maximum-probability volumes from the original Princeton ProbAtlas_v4 distribution of the Wang et al. (2015) visual topography atlas. These are the volumetric counterparts to the surface labels returned by `get_wang_atlas()`: for each of the 25 topographic areas there is a continuous probability map (`perc_VTPM_vol_roi<n>_<hemi>.nii.gz`) in MNI volume space, plus a maximum-probability summary (`maxprob_vol_<hemi>.nii.gz`).

Usage

```
get_wang_prob_atlas(
  prob_dir = NULL,
  image = c("probability", "maxprob"),
  hemi = c("both", "lh", "rh"),
  rois = NULL,
  path_only = TRUE,
  use_cache = TRUE
)

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

Arguments

prob_dir	Optional path to a locally-extracted ProbAtlas_v4 directory (the folder containing subj_vol_all, or that subfolder itself). When supplied, file paths are resolved and existence-checked.
image	One of "probability" (per-area perc_VTPM maps, the default) or "maxprob" (the maximum-probability summary volume).
hemi	One of "both" (default), "lh", or "rh".
rois	Optional subset of areas, given as labels (e.g. c("V1v", "hV4")) or integer ids (1-25). NULL selects all 25.
path_only	Logical. When TRUE (default), return a manifest of paths/metadata without reading image data. When FALSE, read the resolved volumes (requires a valid prob_dir).
use_cache	Logical. Also look for files under the neuroatlas Wang cache directory when prob_dir is not supplied.
x	A wang_prob_paths object.
...	Unused.

Details

The Princeton server (scholar.princeton.edu / napl) blocks automated downloads of the binary archive (it returns HTTP 403 to scripted requests), so this function does **not** download the data. Instead it defaults to path_only = TRUE and returns a manifest describing the expected files, the manual download URL, and the canonical ROI labels.

To work with the actual volumes, download ProbAtlas_v4.zip once in a browser from the Princeton resources page, unzip it, and pass the resulting directory (or its subj_vol_all subfolder) via prob_dir. With a valid prob_dir you can set path_only = FALSE to read the requested volumes as NeuroVol objects.

Value

When path_only = TRUE, an object of class wang_prob_paths: a list with the requested image/hemi, the manual download zip_url, a files tibble (id, label, hemi, member, path, exists), and the

canonical `labels`. When `path_only = FALSE`, a `wang_prob_volumes` list whose `volumes` element holds the loaded `NeuroVol` objects.

Source

https://napl.scholar.princeton.edu/resources

References

Wang, L., Mruczek, R. E. B., Arcaro, M. J., & Kastner, S. (2015). Probabilistic Maps of Visual Topography in Human Cortex. Cerebral Cortex, 25(10), 3911-3931. doi:10.1093/cercor/bhu277

See Also

`get_wang_atlas` for the fsaverage surface atlas.

Examples

```
## Not run:
# Manifest only (no download): see which files are needed and where to get them
manifest <- get_wang_prob_atlas()
manifest
head(manifest$files)

# After manually downloading + unzipping ProbAtlas_v4.zip:
paths <- get_wang_prob_atlas(prob_dir = "~/atlases/ProbAtlas_v4")
v1v_lh <- get_wang_prob_atlas(prob_dir = "~/atlases/ProbAtlas_v4",
                              rois = "V1v", hemi = "lh", path_only = FALSE)

## End(Not run)
```

---

ggseg_schaefer *Create Interactive Schaefer Atlas Visualization*

Description

‘r lifecycle::badge("deprecated")‘

This function has been deprecated in favour of `plot_brain()`. Use `plot_brain(schaefer_surf(...))` for interactive cortical surface visualisation.

Usage

```
ggseg_schaefer(
  atlas,
  vals,
  thresh = NULL,
  pos = FALSE,
  palette = "Spectral",
```

```
  interactive = TRUE,
  lim = range(vals)
)
```

Arguments

atlas	An atlas object containing Schaefer parcellation information
vals	Numeric vector of values to visualize on the atlas
thresh	Numeric vector of length 2 specifying (min, max) thresholds.
pos	Logical. If TRUE, uses raw values for thresholding.
palette	Character string specifying the color palette. Default: "Spectral"
interactive	Logical. If TRUE, creates an interactive plot. Default: TRUE
lim	Numeric vector of length 2 specifying the range for color mapping.

Value

A ggplot2 or ggiraph object.

Examples

```
## Not run:
# Deprecated — use plot_brain(schaefer_surf(200, 17), vals = ...) instead

## End(Not run)
```

---

glasser_surf *Glasser Surface Atlas (fsaverage)*

Description

Load the Glasser HCP-MMP1.0 cortical parcellation projected to the FreeSurfer `fsaverage` surface, as distributed by Kathryn Mills (see Figshare dataset "HCP-MMP1.0 projected on fsaverage"). The result is a pair of neurosurf `LabeledNeuroSurface` objects plus atlas metadata.

Usage

```
glasser_surf(
  space = "fsaverage",
  surf = c("pial", "white", "inflated", "midthickness"),
  use_cache = TRUE
)
```

Arguments

space	Surface space / mesh template. Only "fsaverage" is supported at present.
surf	Surface type. One of "pial", "white", "inflated", or "midthickness".
use_cache	Logical. Whether to cache downloaded annotation files in the neuroatlas cache directory. Default: TRUE.

Details

This function uses:

- fsaverage surface geometry from TemplateFlow via get_surface_template
- fsaverage .annot files from the Mills Figshare distribution (lh.HCP-MMP1.annot, rh.HCP-MMP1.annot)

Currently only the "fsaverage" surface space is supported.

Value

A list with classes c("glasser_surf","surfatlas","atlas") containing:

- lh_atlas, rh_atlas: LabeledNeuroSurface objects for left and right hemispheres.
- surf_type: requested surface type.
- surface_space: surface template space ("fsaverage").
- ids, labels, orig_labels, hemi, cmap: atlas metadata.

Examples

```
## Not run:
# Glasser MMP1.0 on fsaverage pial surface
atl <- glasser_surf(space = "fsaverage", surf = "pial")

## End(Not run)
```

---

infer_design_var_type *Infer Design Variable Type*

Description

Classify a vector as "continuous" or "categorical" for plot defaults.

Usage

```
infer_design_var_type(x)
```

Arguments

x	A vector.

Value

A character scalar.

`install_templateflow` *Install Templateflow (DEPRECATED)*

Description

DEPRECATED: TemplateFlow is now accessed via the pure R 'templateflow' package. Python is no longer required.

Install the R package instead: `remotes::install_github("bbuchsbaum/templateflow")`

Usage

```
install_templateflow(
  method = "auto",
  conda = "auto",
  envname = NULL,
  force_reinstall = FALSE
)
```

Arguments

`method`	Ignored.
`conda`	Ignored.
`envname`	Ignored.
`force_reinstall`	Ignored.

Value

Invisible `NULL`.

Examples

```
# Deprecated. Install the R templateflow package instead:
# remotes::install_github("bbuchsbaum/templateflow")
```

launch_cluster_explorer *Launch Cluster Explorer in Interactive Session*

Description

Convenience wrapper around `cluster_explorer()` that launches the app in an interactive session.

Usage

```
launch_cluster_explorer(...)
```

Arguments

`...`	Passed to `cluster_explorer()`.

Value

Invisibly returns the running app object.

list_atlases *List Registered Atlases*

Description

Enumerate the atlases currently registered in the neuroatlas dispatch registry. Useful for discovery from scripts, vignettes, and Shiny apps.

Usage

```
list_atlases()
```

Value

A tibble with one row per registered atlas and columns ‘id‘, ‘label‘, ‘family‘, ‘representation‘, ‘default_space‘, and a comma-joined ‘aliases‘ string.

See Also

[get_atlas()] to load an atlas by id.

Examples

```
list_atlases()
```

load_surface_template *Load a surface template as a neurosurf geometry*

Description

Convenience wrapper around get_surface_template that downloads (via TemplateFlow) the requested surface geometry and returns it as a neurosurf::SurfaceGeometry object (or a left/right list).

Usage

```
load_surface_template(
  template_id,
  surface_type,
  hemi = c("L", "R", "both"),
  density = NULL,
  resolution = NULL,
  ...
)
```

Arguments

template_id	Surface template identifier passed to TemplateFlow (e.g., "fsaverage", "fsaverage6", "fsLR").
surface_type	Surface type (e.g., "white", "pial", "inflated", "midthickness").
hemi	Hemisphere to load. One of "L", "R", or "both". If "both", a named list with elements L and R is returned.
density	Optional surface density (TemplateFlow density argument).
resolution	Optional resolution string (TemplateFlow res argument), e.g., "06" for fsaverage6.
...	Additional arguments forwarded to get_surface_template.

Value

A neurosurf::SurfaceGeometry object when hemi is "L" or "R"; a named list of two SurfaceGeometry objects when hemi is "both".

Examples

```
## Not run:
  # fsaverage6 pial surface as NeuroSurface
  lh <- load_surface_template("fsaverage", "pial", hemi = "L",
                              density = "41k", resolution = "06")

  # Both hemispheres of fsLR 32k inflated surface
  both <- load_surface_template("fsLR", "inflated", hemi = "both",
```

```
                                density = "32k")

## End(Not run)
```

`map_atlas` *Map values to an atlas*

Description

Maps a set of values to regions/parcels in an atlas object. This can be used to visualize data (like statistics or measurements) across atlas regions.

Usage

```
map_atlas(x, vals, thresh, ...)

## S3 method for class 'atlas'
map_atlas(x, vals, thresh = NULL, pos = FALSE, ...)

## S3 method for class 'schaefer'
map_atlas(x, vals, thresh = NULL, pos = FALSE, ...)

## S3 method for class 'glasser'
map_atlas(x, vals, thresh = NULL, pos = FALSE, ...)
```

Arguments

`x`	An atlas object to map values onto
`vals`	Numeric vector of values to map to atlas regions. Length should match the number of regions in the atlas
`thresh`	Optional numeric vector of length 2 specifying (min, max) thresholds for the mapped values. Values outside this range will be clamped.
`...`	Additional arguments passed to methods
`pos`	Logical. If 'TRUE', values are thresholded using raw values; otherwise the absolute values are used.

Value

Returns the atlas object with mapped values

Examples

```
## Not run:
# Load the aseg atlas
atlas <- get_aseg_atlas()
vals <- rnorm(length(atlas$orig_labels))
```

```
# Map values with a threshold of -2 to 2
mapped <- map_atlas(atlas, vals, thresh = c(-2, 2))

## End(Not run)
```

`map_to_schaefer` *Map Values to Schaefer Atlas Format*

Description

‘r lifecycle::badge("deprecated")‘

This function has been deprecated. Use `map_atlas()` directly or `plot_brain()` for visualisation.

Usage

```
map_to_schaefer(atlas, vals, thresh = NULL, pos = FALSE)
```

Arguments

`atlas`	An atlas object containing Schaefer parcellation information
`vals`	Numeric vector of values to map to atlas regions
`thresh`	Numeric vector of length 2 specifying (min, max) thresholds.
`pos`	Logical. If TRUE, uses raw values; if FALSE, uses absolute values for thresholding. Default: FALSE

Value

A tibble with mapped values.

Examples

```
# Deprecated — use map_atlas() or plot_brain() instead
```

`merge_atlases` *Merge Two Brain Atlases*

Description

Combines two brain atlases into a single unified atlas object, preserving all region information and adjusting region IDs to prevent conflicts. This is useful for creating composite atlases that combine different parcellation schemes.

Usage

```
merge_atlases(atlas1, atlas2)
```

Arguments

`atlas1`	The first atlas object to merge
`atlas2`	The second atlas object to merge

Details

The merging process:

- Verifies that both atlases have the same dimensions
- Adjusts region IDs in the second atlas to avoid overlap
- Combines color maps, labels, and hemisphere information
- Creates a new ClusteredNeuroVol object for the merged atlas

Value

A new atlas object containing:

name Combined names of input atlases (atlas1::atlas2)

atlas Combined `ClusteredNeuroVol` object

cmap Combined colormap for all regions

ids Adjusted vector of all region IDs

labels Combined vector of region labels

orig_labels Original labels from both atlases

hemi Combined hemisphere designations

See Also

`get_aseg_atlas`, `get_roi`

Examples

```
## Not run:
# Load two atlases
atlas1 <- get_aseg_atlas()
atlas2 <- get_aseg_atlas()

# Merge the atlases
merged <- merge_atlases(atlas1, atlas2)

# Check the combined regions
print(merged)

## End(Not run)
```

MNI152_to_MNI305 *MNI152 to MNI305 Affine Transform Matrix*

Description

The inverse of MNI305_to_MNI152, for converting RAS coordinates from MNI152 space back to MNI305 (fsaverage) space.

Usage

```
MNI152_to_MNI305
```

Format

A 4x4 numeric matrix (affine transform in homogeneous coordinates).

Details

This matrix is derived from FreeSurfer's `mni152.register.dat` file, located at `$FREESURFER_HOME/average/mni152.regi`

The transform accounts for the approximately 4mm difference between MNI305 and MNI152 coordinate systems. It includes small rotation, scaling, and translation components.

To apply: for a point `p = c(R, A, S)`, compute `MNI305_to_MNI152 %*% c(p, 1)` and take the first 3 elements.

See Also

MNI305_to_MNI152, transform_coords

Examples

```
# Transform a point from MNI152 to MNI305
point_152 <- c(10.695, -18.409, 36.137)
point_305 <- (MNI152_to_MNI305 %*% c(point_152, 1))[1:3]
print(point_305)  # approximately c(10, -20, 35)
```

MNI305_to_MNI152 *MNI305 to MNI152 Affine Transform Matrix*

Description

The canonical 4x4 affine transformation matrix for converting RAS coordinates from MNI305 (fsaverage) space to MNI152 space.

Usage

```
MNI305_to_MNI152
```

Format

A 4x4 numeric matrix (affine transform in homogeneous coordinates).

Details

This matrix is derived from FreeSurfer's `mni152.register.dat` file, located at `$FREESURFER_HOME/average/mni152.regi`

The transform accounts for the approximately 4mm difference between MNI305 and MNI152 coordinate systems. It includes small rotation, scaling, and translation components.

To apply: for a point `p = c(R, A, S)`, compute `MNI305_to_MNI152 %*% c(p, 1)` and take the first 3 elements.

Source

FreeSurfer: `$FREESURFER_HOME/average/mni152.register.dat`

See Also

https://surfer.nmr.mgh.harvard.edu/fswiki/CoordinateSystems `MNI152_to_MNI305`, `transform_coords`

Examples

```
# Transform a single point from MNI305 to MNI152
point_305 <- c(10, -20, 35)
point_152 <- (MNI305_to_MNI152 %*% c(point_305, 1))[1:3]
print(point_152)  # approximately c(10.695, -18.409, 36.137)
```

needs_coord_transform *Check if a Coordinate Transform is Needed*

Description

Determines whether two template spaces require an affine coordinate transform (e.g., MNI305 to MNI152).

Usage

```
needs_coord_transform(from, to)
```

Arguments

from	Source template space identifier.
to	Target template space identifier.

Details

This checks whether the *coordinate systems* differ (MNI305 vs MNI152), which requires an affine transform. It does NOT check whether the template grids differ within the same coordinate system (use [needs_template_warp()] for that).

Value

Logical. 'TRUE' if the coordinate spaces differ, 'FALSE' if they match, 'NA' if either space is unknown.

See Also

[needs_template_warp()] for checking template-grid differences, [template_to_coord_space()] for the underlying lookup.

Examples

```
needs_coord_transform("fsaverage", "MNI152NLin6Asym")  # TRUE
needs_coord_transform("fsaverage", "MNI305")            # FALSE
needs_coord_transform("MNI152NLin6Asym", "MNI152NLin2009cAsym")  # FALSE
```

needs_template_warp *Check if a Nonlinear Template Warp is Needed*

Description

Determines whether two template spaces are in the same coordinate system but on different template grids, requiring a nonlinear warp.

Usage

```
needs_template_warp(from, to)
```

Arguments

from	Source template space identifier.
to	Target template space identifier.

Details

This function identifies cases where an affine transform is NOT needed but a nonlinear warp IS needed. For example, MNI152NLin6Asym and MNI152NLin2009cAsym are both in MNI152 coordinate space but use different nonlinear registration targets, so voxel grids don't align exactly (~2mm difference).

Value

Logical. 'TRUE' if the templates share a coordinate space but differ in grid/registration, 'FALSE' otherwise, 'NA' if either space is unknown.

See Also

[needs_coord_transform()] for checking coordinate-space differences, [atlas_transform_plan()] for planning multi-step transforms.

Examples

```
needs_template_warp("MNI152NLin6Asym", "MNI152NLin2009cAsym")  # TRUE
needs_template_warp("fsaverage", "fsaverage6")                   # TRUE
needs_template_warp("fsaverage", "MNI152")               # FALSE (different coord spaces)
needs_template_warp("MNI152NLin6Asym", "MNI152NLin6Asym")       # FALSE (identical)
```

needs_transform *Check if Transform is Needed Between Spaces*

Description

Convenience function to check whether a coordinate transform is required when working with a surface template and volumetric data in a given space.

Usage

```
needs_transform(surface_template, volume_space)
```

Arguments

surface_template	Character string identifying the surface template (e.g., "fsaverage", "fsLR").
volume_space	Character string identifying the volumetric data's coordinate space (e.g., "MNI152", "MNI305").

Value

Logical. TRUE if a transform is needed, FALSE if the spaces match.

Examples

```
# fsaverage + MNI152 volume: transform needed
needs_transform("fsaverage", "MNI152")  # TRUE

# fsLR + MNI152 volume: no transform needed
needs_transform("fsLR", "MNI152")  # FALSE

# fsaverage + MNI305 volume: no transform needed
needs_transform("fsaverage", "MNI305")  # FALSE
```

network_anchor_hues *Assign harmonic anchor hues to networks*

Description

Assign harmonic anchor hues to networks

Usage

```
network_anchor_hues(
  network_levels,
  scheme = c("even", "triadic", "tetradic", "complementary"),
  start_hue = 15
)
```

Arguments

network_levels	Character vector of unique network names.
scheme	Hue spacing scheme: ‘"even"‘, ‘"triadic"‘, ‘"tetradic"‘, or ‘"complementary"‘.
start_hue	Starting hue in degrees.

Value

Named numeric vector of anchor hues.

Examples

```
hues <- network_anchor_hues(c("Visual", "Default", "DorsalAttn"))
hues
```

---

new_atlas_ref *Atlas Reference Metadata*

Description

Structured provenance and space metadata for atlas objects.

Usage

```
new_atlas_ref(
  family,
  model,
  representation = c("volume", "surface", "derived"),
  template_space = NA_character_,
  coord_space = NA_character_,
  resolution = NA_character_,
  density = NA_character_,
  provenance = NA_character_,
  source = NA_character_,
  lineage = NA_character_,
  confidence = c("exact", "high", "approximate", "uncertain"),
  notes = NA_character_
)
```

Arguments

family	Atlas family identifier (e.g., ‘"schaefer"‘, ‘"glasser"‘).
model	Atlas model identifier (e.g., ‘"Schaefer2018"‘).
representation	Atlas representation (‘"volume"‘, ‘"surface"‘, or ‘"derived"‘).
template_space	Template/grid identifier (e.g., ‘"MNI152NLin2009cAsym"‘, ‘"fsaverage6"‘).
coord_space	Coordinate-space identifier (e.g., ‘"MNI152"‘, ‘"MNI305"‘).
resolution	Optional resolution descriptor (e.g., ‘"1mm"‘).
density	Optional surface density descriptor (e.g., ‘"41k"‘).
provenance	URL, DOI, or short source identifier.
source	Source key used by the loader.
lineage	Optional derivation/projection note.
confidence	Confidence tier: ‘"exact"‘, ‘"high"‘, ‘"approximate"‘, or ‘"uncertain"‘.
notes	Optional free-text notes.

Value

An object of class ‘"atlas_ref"‘.

---

olsen_mtl *Olsen Medial Temporal Lobe Atlas*

Description

A detailed parcellation atlas of the medial temporal lobe (MTL) regions, including hippocampus and surrounding cortical areas, based on the work of Rosanna Olsen and colleagues.

Usage

```
olsen_mtl
```

Format

A list with class 'atlas' containing:

name Character string identifying the atlas

atlas NeuroVol object containing the parcellation in 1mm MNI space

labels Character vector of anatomical region labels

orig_labels Full region labels including hemisphere information

ids Integer vector of region IDs (1:16)

hemi Character vector indicating hemisphere ('left' or 'right')

Details

The atlas provides a detailed segmentation of MTL structures in MNI space at 1mm resolution. It includes bilateral parcellation of:

- Hippocampal subfields
- Perirhinal cortex
- Entorhinal cortex
- Parahippocampal cortex

Source

Olsen, R. K., et al. (2013). The role of relational binding in item memory: Evidence from face recognition in a case of developmental amnesia. Journal of Neuroscience, 33(36), 14107-14111.

Examples

```
# Load the atlas data
data(olsen_mtl)

# View available regions
olsen_mtl$labels

# Check distribution across hemispheres
table(olsen_mtl$hemi)
```

---

parcel_data *Parcel-Level Data Container*

Description

Create a validated, serializable parcel-level data object.

Usage

```
parcel_data(
  parcels,
  atlas_id,
  atlas_name = atlas_id,
  atlas_version = NULL,
  atlas_space = NULL,
  schema_version = "1.0.0"
)
```

Arguments

parcels	A data frame or tibble with one row per parcel.
atlas_id	Canonical parcellation identifier.
atlas_name	Human-readable parcellation name. Defaults to ‘atlas_id‘.
atlas_version	Optional atlas version string.
atlas_space	Optional template/space identifier.
schema_version	Schema version string. Default: ‘"1.0.0"‘.

Details

‘parcel_data‘ formalizes reduced parcel representations as two components: - atlas identity metadata (‘atlas‘) - a parcel table (‘parcels‘) with required columns ‘id‘, ‘label‘, and ‘hemi‘

Additional columns in ‘parcels‘ are interpreted as value or feature columns.

Value

An object of class ‘"parcel_data"‘.

See Also

[as_parcel_data()], [write_parcel_data()], [read_parcel_data()]

Examples

```
tbl <- tibble::tibble(
  id = c(1L, 2L),
  label = c("A", "B"),
  hemi = c("left", "right"),
  statistic = c(0.4, -0.2)
)
x <- parcel_data(tbl, atlas_id = "toy_atlas")
x
```

---

parcel_values *Extract Parcel Values Aligned to an Atlas*

Description

Returns a vector aligned to ‘atlas$ids‘, suitable for ‘map_atlas()‘ or ‘plot_brain()‘.

Usage

```
parcel_values(x, atlas, column = "value")
```

Arguments

x	A ‘parcel_data‘ object.
atlas	An atlas object.
column	Value column in ‘x$parcels‘ to extract.

Value

A vector with ‘length(atlas$ids)‘ elements ordered to ‘atlas$ids‘.

---

plot.atlas *Plot Glasser Atlas*

Description

Visualise a Glasser atlas object. For surface atlases (glasser_surf), renders via plot_brain(). For volumetric Glasser atlases, falls back to the base plot.atlas() method.

Volumetric slice visualisation for any atlas object. By default renders a multi-slice montage with each parcel coloured by the atlas_roi_colors() system. An orthogonal three-plane view is also available.

Usage

```
## S3 method for class 'atlas'
plot(
  x,
  y,
  view = c("montage", "ortho"),
  method = "rule_hcl",
  colors = NULL,
  nslices = 12L,
  ...
)

## S3 method for class 'glasser'
plot(
  x,
  y,
  vals = NULL,
  thresh = c(0, 0),
  pos = FALSE,
  palette = "cork",
  lim = NULL,
  ...
)

## S3 method for class 'surfatlas'
plot(x, y, vals = NULL, views = c("lateral", "medial"), ...)
```

Arguments

x	An atlas object (atlas, glasser, schaefer, etc.)
y	Ignored (required for S3 consistency)
view	Character; "montage" (default) for a multi-slice montage or "ortho" for three orthogonal planes.
method	Colour algorithm passed to atlas_roi_colors(). One of "rule_hcl" (default), "network_harmony", "maximin_view", or "embedding".
colors	Optional pre-computed colour specification: a tibble from atlas_roi_colors(), or a named character vector of hex colours keyed by region ID.
nslices	Number of slices for montage view (default 12).
...	Additional arguments passed to neuroim2::plot_montage() or neuroim2::plot_ortho().
vals	Numeric vector of values to visualize. If NULL (default), all regions will be assigned a value of 1, creating a uniform visualization
thresh	Numeric vector of length 2 for thresholding values
pos	Logical. If TRUE, uses raw values for thresholding
palette	Character. Name of scico color palette
lim	Numeric vector of length 2 for color scale limits. If NULL, will be set to range of vals
views	Character vector of views to render for surface atlases. Any combination of "lateral", "medial", "dorsal", "ventral". Default: c("lateral", "medial").

Details

plot.surfatlas renders a cortical surface projection via plot_brain().

Value

A ggplot2 or ggiraph object

For view = "montage", a ggplot2 object (invisibly). For view = "ortho", a patchwork composite (if available) or a list of three ggplot2 objects.

See Also

atlas_roi_colors, ggseg_schaefer

Examples

```
atlas <- get_aseg_atlas()
plot(atlas)
plot(atlas, view = "ortho")
plot(atlas, method = "maximin_view", nslices = 6)
```

plot_brain *Plot Brain Surface Atlas*

Description

Renders a triangle-mesh projection of cortical surface parcellations with configurable views and optional ggiraph interactivity. This function replaces the legacy ggseg-based visualisation pipeline.

Usage

```
plot_brain(
  surfatlas,
  vals = NULL,
  views = c("lateral", "medial"),
  hemis = c("left", "right"),
  surface = "inflated",
  color_method = "rule_hcl",
  colors = NULL,
  palette = "cork",
  lim = NULL,
  interactive = TRUE,
  data_id_mode = c("parcel", "polygon"),
  ncol = 2L,
  panel_layout = c("native", "presentation"),
  style = c("default", "ggseg_like"),
  border = TRUE,
  border_geom = c("path", "segment"),
  boundary_smooth = 0L,
  projection_smooth = 0L,
  border_color = "grey30",
  border_size = 0.15,
  border_lineend = "round",
  border_linejoin = "round",
  silhouette = border,
  silhouette_color = border_color,
  silhouette_size = border_size,
  network_border = FALSE,
  network_border_color = border_color,
  network_border_size = border_size * 2,
  shading = FALSE,
  shading_strength = 0.22,
  shading_gamma = 1,
  shading_color = "black",
  fill_alpha = 1,
  overlay = NULL,
  overlay_threshold = NULL,
  overlay_alpha = 0.45,
```

```
  overlay_palette = "vik",
  overlay_lim = NULL,
  overlay_border = TRUE,
  overlay_border_color = "black",
  overlay_border_size = 0.25,
  overlay_fun = c("avg", "nn", "mode"),
  overlay_sampling = c("midpoint", "normal_line", "thickness"),
  colorbar = FALSE,
  colorbar_title = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  panel_labels = NULL,
  outline = FALSE,
  bg = "white",
  ...
)
```

Arguments

surfatlas	A surface atlas object of class "surfatlas" (e.g. from schaefer_surf() or glasser_surf()).
vals	Optional numeric vector of values to map onto parcels. Length must equal the number of atlas regions (length(surfatlas$ids)). When NULL (default), parcels are coloured using the ROI colour system.
views	Character vector of views to render. Any combination of "lateral", "medial", "dorsal", "ventral". Default: c("lateral", "medial").
hemis	Character vector of hemispheres to include. Default: c("left", "right").
surface	Surface type. One of "inflated", "pial", "white". Must match the surface type of surfatlas.
color_method	Colour algorithm for discrete parcel colouring (when vals is NULL). Passed to atlas_roi_colors(). Default: "rule_hcl".
colors	Optional pre-computed colours: a tibble with id and color columns, or a named character vector of hex colours keyed by region ID. Overrides color_method when vals is NULL.
palette	Character: scico palette for continuous colour scale (when vals is provided). Default: "cork".
lim	Numeric vector of length 2 for colour scale limits (continuous mode). Defaults to range of vals.
interactive	Logical. If TRUE (default), returns a ggiraph::girafe widget with hover tooltips. If FALSE, returns a static ggplot2 object.
data_id_mode	Interactive data-id granularity (when interactive = TRUE): "parcel" (default) uses parcel ids; "polygon" encodes panel + parcel + polygon/face id for click-to-surface workflows.
ncol	Integer: number of columns in the facet layout. Default: 2.

`panel_layout` Panel coordinate layout strategy: `"native"` (default) preserves raw projected units; `"presentation"` recentres each panel, rotates dorsal/ventral views to horizontal, and normalises per-panel scale for a cleaner ggseg-like grid.

`style` Visual preset. `"default"` keeps existing behaviour. `"ggseg_like"` enables a cleaner publication style and, unless explicitly overridden, switches `panel_layout` to `"presentation"` with softer border defaults and light projection smoothing.

`border` Logical. If `TRUE` (default), draw thin lines at parcel boundaries (edges between different parcels). Gives a clean ggseg-like appearance.

`border_geom` Boundary rendering method. `"path"` (default) chains boundary edges into longer paths for smoother lines; `"segment"` draws each boundary edge independently.

`boundary_smooth`
Non-negative integer controlling Chaikin smoothing iterations applied to boundary paths when `border_geom = "path"`. `0` (default) keeps original mesh-aligned boundaries; `1` or `2` yields cleaner curves in close-up figures.

`projection_smooth`
Non-negative integer controlling Laplacian-like smoothing iterations applied to projected vertex coordinates before parcel polygons are constructed. This smooths filled parcel edges while preserving shared boundaries across parcels. `0` (default) keeps raw projected coordinates.

`border_color` Colour for parcel boundary lines. Default: `"grey30"`.

`border_size` Line width for parcel boundaries. Default: `0.15`.

`border_lineend` Line end style for boundary lines (passed to `geom_path` / `geom_segment`). One of `"butt"`, `"round"`, `"square"`. Default: `"round"`.

`border_linejoin`
Line join style for boundary lines (passed to `geom_path` / `geom_segment`). One of `"round"`, `"mitre"`, `"bevel"`. Default: `"round"`.

`silhouette` Logical. If `TRUE`, draw the mesh silhouette (edges between visible and culled faces) as a separate boundary layer. Defaults to `border`.

`silhouette_color`
Colour for silhouette lines. Default: `border_color`.

`silhouette_size`
Line width for silhouette lines. Default: `border_size`.

`network_border` Logical. If `TRUE`, highlight boundaries between different networks (requires `surfatlas$network`). Default: `FALSE`.

`network_border_color`
Colour for network boundary lines. Default: `border_color`.

`network_border_size`
Line width for network boundary lines. Default: `border_size * 2`.

`shading` Logical. If `TRUE`, overlay a subtle normal-based shading layer to enhance depth cues (recommended for static figures).

`shading_strength`
Numeric in `[0, 1]`. Maximum opacity of the shading overlay. Default: `0.22`.

`shading_gamma` Positive numeric scalar controlling the shadow falloff. Higher values concentrate shadows in more oblique regions. Default: `1`.

shading_color Colour of the shading overlay. Default: "black".

fill_alpha Numeric in [0, 1]. Opacity of parcel fills. Lower values can help the shading read more clearly. Default: 1.

overlay Vertex-wise overlay or a NeuroVol. If a NeuroVol, it is automatically projected onto the surface using neurosurf::vol_to_surf(). Otherwise, a list with lh and rh components (numeric vectors matching the vertex count of each hemisphere mesh).

overlay_threshold
Optional absolute threshold for overlay values before rendering.

overlay_alpha Numeric in [0, 1]. Opacity of overlay polygons. Default: 0.45.

overlay_palette
scico palette for overlay colour mapping. Default: "vik".

overlay_lim Optional numeric length-2 limits for overlay colour mapping.

overlay_border Logical. If TRUE, draw cluster overlay boundaries. Default: TRUE.

overlay_border_color
Colour for overlay boundaries. Default: "black".

overlay_border_size
Line width for overlay boundaries. Default: 0.25.

overlay_fun Character: interpolation function passed to neurosurf::vol_to_surf() when overlay is a NeuroVol. One of "avg", "nn", or "mode". Default: "avg".

overlay_sampling
Character: sampling strategy passed to neurosurf::vol_to_surf() when overlay is a NeuroVol. One of "midpoint", "normal_line", or "thickness". Default: "midpoint".

colorbar Logical or character. When vals is non-NULL and interactive = FALSE, controls whether and where to add a standalone colorbar panel. Use TRUE or "right" for a vertical colorbar, "bottom" for a horizontal colorbar, or FALSE / "none" to omit it. Default: FALSE.

colorbar_title Optional character label for the colorbar.

title, subtitle, caption
Optional plot-level annotations for static output. When a colorbar is present these are applied to the composed figure; otherwise they are added directly to the returned ggplot.

panel_labels Optional panel label override. Use either an unnamed character vector matching the number of panels, a named character vector keyed by default panel names such as "Left Lateral", or a function that takes the default panel name and returns a new label.

outline Logical. If TRUE, draw every triangle edge (mesh wireframe). Default: FALSE. Typically border is preferred.

bg Character: background colour for the plot. Default: "white".

... Additional arguments (currently unused).

Value

A ggplot2 object (when interactive = FALSE and no standalone colorbar is requested), a patchwork object (when a standalone static colorbar is composed), or a ggiraph::girafe widget (when interactive = TRUE).

Examples

```
## Not run:
atl <- schaefer_surf(200, 17)
plot_brain(atl)
plot_brain(atl, vals = rnorm(200), palette = "vik")
plot_brain(atl, views = "lateral", interactive = FALSE)

# Styling: rounded white parcel borders + thicker silhouette + network edges
plot_brain(
  atl,
  interactive = FALSE,
  border_color = "white",
  border_size = 0.25,
  border_lineend = "round",
  silhouette_size = 0.6,
  network_border = TRUE,
  network_border_color = "grey10",
  network_border_size = 0.5,
  shading = TRUE,
  fill_alpha = 0.98,
  bg = "#f7f7f7"
)

## End(Not run)
```

---

plot_brain_grid *Multi-panel Brain Plot Grid*

Description

Arranges multiple brain surface plots into a grid layout using **patchwork**. Each element of vals_list produces one panel rendered by plot_brain.

Usage

```
plot_brain_grid(
  surfatlas,
  vals_list,
  views = c("lateral", "medial"),
  hemis = c("left", "right"),
  ncol = NULL,
  shared_scale = TRUE,
```

```
  palette = "cork",
  lim = NULL,
  titles = NULL,
  colorbar = TRUE,
  colorbar_title = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  ...
)
```

Arguments

surfatlas	A surfatlas object (e.g., from schaefer_surf or glasser_surf).
vals_list	A named list of numeric vectors, one per panel. Each vector must have length equal to the number of atlas regions.
views	Character vector of views passed to plot_brain. Default: c("lateral", "medial").
hemis	Character vector of hemispheres passed to plot_brain. Default: c("left", "right").
ncol	Integer number of columns in the grid layout. If NULL, chosen automatically.
shared_scale	Logical. If TRUE (default), all panels share the same colour scale computed from the range of all values.
palette	Character: scico palette name. Default: "cork".
lim	Optional numeric length-2 colour limits. Overrides automatic limits when provided.
titles	Optional character vector of panel titles. If NULL, names of vals_list are used.
colorbar	Logical or character. Use TRUE or "right" (default) for a vertical shared colorbar, "bottom" for a horizontal shared colorbar, or FALSE / "none" to omit it.
colorbar_title	Optional shared colorbar title.
title, subtitle, caption	Optional overall plot annotations applied to the composed figure.
...	Additional arguments passed to plot_brain.

Value

A patchwork object.

Examples

```
## Not run:
atl <- schaefer_surf(200, 7)
vals <- list(
  Contrast_A = rnorm(200),
  Contrast_B = rnorm(200)
)
```

```
plot_brain_grid(atl, vals)

## End(Not run)
```

`plot_glasser` *Plot Glasser Atlas Values*

Description

‘r lifecycle::badge("deprecated")‘

This function has been deprecated. Use `plot_brain(glasser_surf(), vals = ...)` instead for interactive cortical surface visualisation.

Usage

```
plot_glasser(vals = NULL, value_col = "value", position = "dispersed")
```

Arguments

`vals`	A data frame containing values to plot, must include columns: • label: character, matching ggseg Glasser atlas labels • value: numeric, values to visualize for each region If NULL (default), all regions will be assigned a value of 1
`value_col`	Character string specifying the name of the column in vals containing the values to plot. Defaults to "value"
`position`	Character string specifying layout type.

Value

An error; this function has been removed.

Examples

```
## Not run:
# Deprecated — use plot_brain(glasser_surf(), vals = ...) instead

## End(Not run)
```

print.atlas *Print Methods for neuroatlas Objects*

Description

Print methods for various atlas objects in the neuroatlas package

Usage

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

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

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

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

Arguments

x	An atlas object (atlas, glasser, schaefer, etc.)
...	Additional arguments passed to print

Value

Invisibly returns x.

The object is returned invisibly

Examples

```
atlas <- get_aseg_atlas()
print(atlas)
```

print.atlas_alignment *Print Method for Atlas Alignment Results*

Description

Print Method for Atlas Alignment Results

Usage

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

Arguments

x	An ‘atlas_alignment‘ object.
...	Unused.

Value

Invisibly returns ‘x‘.

Examples

```
ref <- new_atlas_ref(
  family = "schaefer",
  model = "Schaefer2018",
  representation = "volume",
  template_space = "MNI152NLin6Asym",
  coord_space = "MNI152",
  confidence = "high"
)
a <- structure(list(atlas_ref = ref), class = c("schaefer", "atlas"))
print(atlas_alignment(a, a))
```

---

print.atlas_ref *Print Method for Atlas References*

Description

Print Method for Atlas References

Usage

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

Arguments

x	An ‘atlas_ref‘ object.
...	Unused.

Value

Invisibly returns ‘x‘.

print.atlas_transform_plan *Print Method for Transform Plans*

Description

Print Method for Transform Plans

Usage

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

Arguments

x	An ‘atlas_transform_plan‘ object.
...	Unused.

Value

Invisibly returns ‘x‘.

Examples

```
p <- atlas_transform_plan("MNI305", "MNI152")
print(p)
```

print.templateflow *Print a TemplateFlow Object*

Description

Provides a brief summary of the TemplateFlow interface object.

Usage

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

Arguments

x	An object of class `templateflow`.
...	Additional arguments (unused).

Value

The input object `x`, returned invisibly.

project_surface_view *Project Surface Vertices to a Canonical 2D View*

Description

Public wrapper for the core surface-view projection used by `plot_brain()`.

Usage

```
project_surface_view(
  verts,
  view = c("lateral", "medial", "dorsal", "ventral"),
  hemi = c("left", "right")
)
```

Arguments

`verts`	Numeric matrix ($N \times 3$) of vertex coordinates.
`view`	Character scalar: one of `"lateral"`, `"medial"`, `"dorsal"`, `"ventral"`.
`hemi`	Character scalar: `"left"` or `"right"`.

Value

A list with elements:

`xy` Numeric matrix ($N \times 2$) of projected coordinates.

`view_dir` Numeric length-3 view direction vector used for backface culling.

query_coord *Query Atlas Labels by Coordinate*

Description

Atlas-first convenience wrappers for label lookup. ‘query_coord()‘ queries world/mm coordinates, while ‘query_vox()‘ queries R-style 1-based voxel grid indices and converts them to world coordinates before dispatching to ‘query_point()‘.

Usage

```
query_coord(x, coords, ...)

query_vox(x, ijk, ...)
```

Arguments

x	A single atlas object or a named list of atlas objects.
coords	Numeric vector of length 3 or an N x 3 matrix of world/mm coordinates.
...	Additional arguments passed to ‘query_point()‘, such as ‘radius‘ or ‘from_space‘.
ijk	Numeric/integer vector of length 3 or an N x 3 matrix of R-style 1-based voxel grid indices.

Value

A tibble with atlas labels at the requested locations.

---

query_point *Look Up Atlas Regions at MNI Coordinates*

Description

Given one or more 3D coordinates, look up the atlas region(s) at each location. Supports exact voxel lookup (‘radius = 0‘) or fuzzy search (‘radius > 0‘) that returns all regions within a sphere of given radius in millimetres. Multiple atlases can be queried simultaneously.

Usage

```
query_point(coords, atlas, radius = 0, from_space = "MNI152")
```

Arguments

coords	Numeric vector of length 3 (single point) or an N x 3 matrix of world coordinates.
atlas	A single atlas object (class ‘"atlas"‘) or a named list of atlas objects.
radius	Numeric scalar giving the search radius in millimetres. ‘0‘ (default) performs an exact voxel lookup; values greater than zero return every region whose labelled voxels fall within ‘radius‘ mm of the query coordinate.
from_space	Character string identifying the coordinate space of the input coordinates (default ‘"MNI152"‘).

Details

World coordinates are converted to voxel grid positions via coord_to_grid and rounded to the nearest integer. When ‘radius > 0‘, candidate voxel centres in a local grid neighbourhood are tested in world coordinates to find all labelled atlas voxels within ‘radius‘ mm.

If the atlas carries a coordinate-space annotation (‘atlas$atlas_ref$coord_space‘) that differs from ‘from_space‘, the input coordinates are transformed automatically via transform_coords.

Value

A tibble with columns:

point Integer index of the query coordinate (row number).

x, y, z The input world coordinates.

atlas_name Name of the atlas.

id Integer region ID (‘NA‘ for background or out-of-bounds).

label Region label string (‘NA‘ for background/OOB).

hemi Hemisphere designation (‘NA‘ if unavailable).

network Network label (‘NA‘ if unavailable).

Examples

```
## Not run:
atlas <- get_schaefer_atlas(parcels = "200", networks = "7")
query_point(c(24, -10, 5), atlas)

# Multiple points
pts <- rbind(c(24, -10, 5), c(-30, 20, 50))
query_point(pts, atlas)

# Fuzzy search within 5 mm
query_point(c(24, -10, 5), atlas, radius = 5)

# Query multiple atlases at once
atlases <- list(schaefer = atlas, aseg = get_aseg_atlas())
query_point(c(24, -10, 5), atlases)

## End(Not run)
```

---

read_parcel_data *Read a ‘parcel_data‘ Object from Disk*

Description

Read a ‘parcel_data‘ Object from Disk

Usage

```
read_parcel_data(file, format = c("auto", "rds", "json"), validate = TRUE)
```

Arguments

file	Input file path.
format	Serialization format: ‘"auto"‘, ‘"rds"‘, or ‘"json"‘.
validate	Logical. If ‘TRUE‘ (default), validate after reading.

Value

A ‘parcel_data‘ object.

`reduce_atlas` *Reduce a NeuroVol or NeuroVec by an Atlas*

Description

Applies a summary function to data within each ROI defined by an atlas. This is an S3 generic function.

Usage

```
reduce_atlas(atlas, data_vol, stat_func, ..., format = NULL)

## S3 method for class 'atlas'
reduce_atlas(
  atlas,
  data_vol,
  stat_func,
  ...,
  format = NULL,
  level = c("parcel", "network", "hemisphere"),
  by = NULL
)
```

Arguments

`atlas`	An atlas object or another object for which a method is defined.
`data_vol`	A `NeuroVol` (3D) or `NeuroVec` (4D) with the data to be summarized. When a 3D volume is supplied the result contains a single row; 4D inputs yield one row per time point.
`stat_func`	The function to apply to the data within each ROI (e.g., `mean`, `sd`, `sum`).
`...`	Additional arguments passed to `stat_func`.
`format`	Character string specifying output format: "wide" or "long". If NULL (default), uses "long" for NeuroVol and "wide" for NeuroVec.

Details

When `data_vol` is a 3D `NeuroVol`, the returned tibble contains a single row with one column per ROI. If a 4D `NeuroVec` is supplied, each time point is summarised separately and a `time` column is added to the tibble.

Value

A `tibble` with format depending on the `format` parameter:

- `"wide"`: Regions as columns. For NeuroVec, rows are time points.
- `"long"`: Columns are `region` and `value` (NeuroVol) or `time`, `region`, and `value` (NeuroVec).

See Also

`map_atlas` for mapping values to atlas regions, `get_roi` for extracting specific regions

Examples

```
## Not run:
# Load an atlas
atlas <- get_schaefer_atlas(parcels = "200", networks = "7")

# Create example data (random values in brain space)
brain_data <- neuroim2::NeuroVol(rnorm(prod(dim(atlas$atlas))),
                                 space = space(atlas$atlas))

# Compute mean values within each atlas region
region_means <- reduce_atlas(atlas, brain_data, mean)

# Compute standard deviation within each region
region_sds <- reduce_atlas(atlas, brain_data, sd, na.rm = TRUE)

## End(Not run)
```

---

reduce_atlas_vec *Reduce a NeuroVec by an Atlas to a ClusteredNeuroVec*

Description

Averages (or otherwise summarises) a 4D image within each atlas parcel, returning a `ClusteredNeuroVec` whose voxels share one time-series per region.

Usage

```
reduce_atlas_vec(atlas, data_vol, mask, ...)

## S3 method for class 'atlas'
reduce_atlas_vec(
  atlas,
  data_vol,
  mask,
  stat_func = mean,
  dilate = FALSE,
```

```
  radius = 4,
  maxn = 50,
  ...
)
```

Arguments

atlas	An atlas object.
data_vol	A NeuroVec (4D) with the data to be summarised.
mask	A NeuroVol or LogicalNeuroVol brain mask. Non-zero values are treated as TRUE.
...	Additional arguments passed to methods.
stat_func	Function used to aggregate voxel values within each parcel (default: mean).
dilate	Logical. If TRUE, atlas parcels are dilated into the mask before averaging (via dilate_atlas).
radius	Numeric dilation radius in voxels (passed to dilate_atlas). Default: 4.
maxn	Integer maximum neighbours for dilation (passed to dilate_atlas). Default: 50.

Details

The atlas volume is intersected with the brain mask so that only voxels inside the mask contribute to each parcel average.

Cluster IDs are remapped to contiguous 1:K before constructing the ClusteredNeuroVec. This is required because the [,,,t] accessor in **neuroim2** uses cluster IDs as direct column indices into the internal time-series matrix.

Parcels that have no voxels inside the mask receive NA time-series in the output.

Value

A ClusteredNeuroVec.

See Also

reduce_atlas for a tibble-based summary, dilate_atlas for expanding parcels into a mask.

Examples

```
## Not run:
atlas <- get_schaefer_atlas(parcels = "200", networks = "7")
# data_vol: a NeuroVec with matching spatial dimensions
# mask:     a brain mask NeuroVol
cvec <- reduce_atlas_vec(atlas, data_vol, mask)
ts_mat <- as.matrix(cvec)   # T x K cluster time-series

## End(Not run)
```

`resample` *Resample Volume to New Space*

Description

Resamples a volume to a new space with optional smoothing of parcel boundaries. This is particularly useful for atlas parcellations where maintaining discrete labels is important.

Usage

```
resample(
  vol,
  outspace,
  smooth = FALSE,
  interp = 0,
  radius = NULL,
  min_neighbors = 3
)
```

Arguments

`vol`	A NeuroVol object to be resampled
`outspace`	A NeuroSpace object specifying the target space
`smooth`	Logical. Whether to apply boundary smoothing after resampling. Default: FALSE
`interp`	Integer. Interpolation method (0=nearest neighbor, 1=linear). Default: 0
`radius`	Numeric. Radius for smoothing neighborhood in voxels. If NULL, uses max(spacing)+0.5. Default: NULL
`min_neighbors`	Integer. Minimum number of neighbors required for smoothing. Default: 3

Details

The resampling process:

- First performs nearest-neighbor interpolation to the new space
- Optionally smooths boundaries using a local majority voting scheme
- Preserves zeros in the mask (background)

Value

A resampled NeuroVol object in the new space

Examples

```
atlas <- get_aseg_atlas()
vol <- atlas$atlas
new_space <- neuroim2::NeuroSpace(
  dim = dim(vol), spacing = neuroim2::spacing(vol)
)
resampled <- resample(vol, new_space)
```

`roi_attributes` *List Available ROI Attributes*

Description

Returns the names of attributes available for ROIs in an atlas. This is useful for discovering what metadata is available for filtering or analysis.

Usage

```
roi_attributes(x, ...)

## S3 method for class 'atlas'
roi_attributes(x, ...)
```

Arguments

`x`	An atlas object
`...`	Additional arguments passed to methods

Value

A character vector of attribute names that contain meaningful (non-NA) values. Excludes internal fields like color values and the id column.

See Also

`roi_metadata` for getting the full metadata tibble

Examples

```
## Not run:
# Discover available attributes
atlas <- get_schaefer_atlas(parcels = "200", networks = "7")
roi_attributes(atlas)
#> [1] "label" "label_full" "hemi" "network"

# Compare with ASEG atlas (no network attribute)
```

```
aseg <- get_aseg_atlas()
roi_attributes(aseg)
#> [1] "label" "label_full" "hemi"

## End(Not run)
```

`roi_colors_embedding` *Embedding-based palette with smooth gradients*

Description

Project ROI features to two dimensions (UMAP or PCA) and map polar angle to hue, yielding palettes that look globally structured.

Usage

```
roi_colors_embedding(
  rois,
  id_col = "roi",
  feature_cols = c("x", "y", "z"),
  hemi_col = NULL,
  method = c("umap", "pca"),
  C_range = c(40, 90),
  L = 65,
  seed = 1
)
```

Arguments

`rois`	Tibble with one ROI per row containing ‘roi‘, ‘x‘, ‘y‘, and ‘z‘.
`id_col`	Column containing ROI IDs.
`feature_cols`	Character vector of numeric or categorical columns that will be embedded (passed to ‘stats::model.matrix()‘).
`hemi_col`	Optional hemisphere column used for the final light/dark adjustment.
`method`	Either ‘"umap"‘ (requires the ‘uwot‘ package) or ‘"pca"‘.
`C_range`	Chroma range mapped from radial distance in the embedding.
`L`	Lightness value used for all ROIs before hemispheric adjustments.
`seed`	Random seed for reproducibility.

Value

Tibble with ROI IDs and colours.

Examples

```
rois <- data.frame(
  roi = 1:10, network = rep(c("Vis", "DMN"), 5),
  x = runif(10), y = runif(10), z = runif(10)
)
pal <- roi_colors_embedding(rois, feature_cols = c("x","y","z"),
                            method = "pca")
```

`roi_colors_maximin_view` *Slice-aware maximin palette for ROIs*

Description

Assign perceptually separated colours by penalising conflicts between ROIs that are likely to appear together in axial/coronal/sagittal views. This is a strong default for general slice and surface visualisation.

Usage

```
roi_colors_maximin_view(
  rois,
  id_col = "roi",
  xyz_cols = c("x", "y", "z"),
  hemi_col = NULL,
  network_col = NULL,
  pair_col = NULL,
  k = 12,
  sigma_xy = 25,
  sigma_slice = 10,
  candidate_multiplier = 10,
  tau = 10,
  lambda_global = 0.15,
  bg_hex = "#808080",
  alpha = 0.85,
  seed = 1,
  weight_transform = NULL
)
```

Arguments

`rois`	Tibble with one ROI per row containing ‘roi‘, ‘x‘, ‘y‘, and ‘z‘.
`id_col`	Column containing ROI IDs.
`xyz_cols`	Character vector of length three giving the coordinates to use for distance calculations.

hemi_col	Optional hemisphere column used for the final light/dark adjustment.
network_col	Optional network label column to slightly boost cross-network conflicts.
pair_col	Optional column identifying homologous L/R ROI pairs. When supplied, colours are first assigned to the homolog pair and then nudged per hemisphere.
k	Number of nearest neighbours to evaluate when forming edges.
sigma_xy	In-plane decay (mm) for slice visibility.
sigma_slice	Through-slice decay (mm) for slice visibility.
candidate_multiplier	How many candidate colours to generate relative to the number of ROIs.
tau	Soft-min temperature; smaller emphasises hard minimum distance, larger smooths the objective.
lambda_global	Optional additional term that encourages global colour diversity.
bg_hex	Background grey (for contrast filtering).
alpha	Opacity of overlays when evaluating contrast.
seed	Random seed for reproducibility.
weight_transform	Optional weight adjustment callback passed to [build_conflict_edges()].

Value

Tibble with ROI IDs and assigned colours.

Examples

```
rois <- data.frame(
  roi = 1:10,
  x = runif(10, 0, 90), y = runif(10, 0, 100), z = runif(10, 0, 80)
)
pal <- roi_colors_maximin_view(rois, xyz_cols = c("x", "y", "z"))
```

---

roi_colors_network_harmony

Network-harmonic palette with neighbour separation

Description

Assign colours so that networks share analogous hue families while still maximising spatial separation between neighbouring ROIs.

Usage

```
roi_colors_network_harmony(
  rois,
  id_col = "roi",
  xyz_cols = c("x", "y", "z"),
  network_col = "network",
  hemi_col = NULL,
  k = 12,
  sigma_xy = 25,
  sigma_slice = 10,
  scheme = "even",
  start_hue = 15,
  hue_width = 28,
  candidate_multiplier = 8,
  tau = 10,
  lambda_global = 0.1,
  bg_hex = "#808080",
  alpha = 0.85,
  seed = 1,
  weight_transform = NULL
)
```

Arguments

rois	Tibble with one ROI per row containing ‘roi‘, ‘x‘, ‘y‘, and ‘z‘.
id_col	Column containing ROI IDs.
xyz_cols	Character vector of length three giving the coordinates to use for distance calculations.
network_col	Optional network label column to slightly boost cross-network conflicts.
hemi_col	Optional hemisphere column used for the final light/dark adjustment.
k	Number of nearest neighbours to evaluate when forming edges.
sigma_xy	In-plane decay (mm) for slice visibility.
sigma_slice	Through-slice decay (mm) for slice visibility.
scheme	Hue spacing scheme passed to [network_anchor_hues()].
start_hue	Starting hue in degrees (0–360).
hue_width	Half-width (degrees) of the analogous band around each anchor hue.
candidate_multiplier	How many candidate colours to generate relative to the number of ROIs.
tau	Soft-min temperature; smaller emphasises hard minimum distance, larger smooths the objective.
lambda_global	Optional additional term that encourages global colour diversity.
bg_hex	Background grey (for contrast filtering).
alpha	Opacity of overlays when evaluating contrast.

seed Random seed for reproducibility.

weight_transform
Optional weight adjustment callback passed to [build_conflict_edges()].

Value

Tibble with ROI IDs and hex colours.

Examples

```
rois <- data.frame(
  roi = 1:10, network = rep(c("Vis", "DMN"), 5),
  x = runif(10, 0, 90), y = runif(10, 0, 100), z = runif(10, 0, 80)
)
pal <- roi_colors_network_harmony(rois, xyz_cols = c("x","y","z"),
                                  network_col = "network")
```

---

roi_colors_rule_hcl *Deterministic HCL palette with harmonic variation*

Description

Generates a fast, reproducible palette that combines harmonic network hues with anterior-posterior gradients and hemisphere luminance differences.

Usage

```
roi_colors_rule_hcl(
  rois,
  id_col = "roi",
  xyz_cols = c("x", "y", "z"),
  network_col = NULL,
  hemi_col = NULL,
  scheme = "even",
  start_hue = 15,
  hue_width = 30,
  C = 70,
  L_L = 72,
  L_R = 60
)
```

Arguments

rois Tibble with one ROI per row containing ‘roi‘, ‘x‘, ‘y‘, and ‘z‘.

id_col Column containing ROI IDs.

xyz_cols	Character vector of length three giving the coordinates to use for distance calculations.
network_col	Optional network label column to slightly boost cross-network conflicts.
hemi_col	Optional hemisphere column used for the final light/dark adjustment.
scheme	Hue spacing scheme passed to [network_anchor_hues()].
start_hue	Starting hue in degrees (0–360).
hue_width	Half-width (degrees) of within-network hue modulation.
C	Fixed chroma.
L_L	Lightness for left hemisphere ROIs.
L_R	Lightness for right hemisphere ROIs.

Value

Tibble with ROI IDs and colours.

Examples

```
rois <- data.frame(
  roi = 1:10, network = rep(c("Vis", "DMN"), 5),
  hemi = rep(c("left", "right"), 5),
  x = runif(10), y = runif(10), z = runif(10)
)
pal <- roi_colors_rule_hcl(rois, network_col = "network",
                           hemi_col = "hemi", xyz_cols = c("x","y","z"))
```

---

roi_metadata *ROI Metadata Functions*

Description

Functions for accessing and filtering ROI (Region of Interest) metadata in atlas objects. These functions provide a unified, discoverable interface for working with multiple ROI attributes across different atlas types.

Returns a tibble containing metadata for all regions of interest (ROIs) in an atlas. This provides a unified, tidy interface for accessing ROI attributes across different atlas types.

Usage

```
roi_metadata(x, ...)

## S3 method for class 'atlas'
roi_metadata(x, ...)
```

Arguments

x	An atlas object
...	Additional arguments passed to methods

Value

A tibble with one row per ROI and columns for each attribute. Standard columns include:

id Numeric ROI identifier

label Simplified region name

label_full Original/full region label

hemi Hemisphere ("left", "right", or NA for bilateral/midline)

color_r, color_g, color_b RGB color values (0-255)

template_space Template space identifier (e.g., "MNI152NLin6Asym"), when atlas_ref is available

coord_space Coordinate space (e.g., "MNI152", "MNI305"), when atlas_ref is available

atlas_family Atlas family identifier (e.g., "schaefer"), when atlas_ref is available

atlas_model Atlas model identifier (e.g., "Schaefer2018"), when atlas_ref is available

atlas_representation Representation type ("volume", "surface", or "derived"), when atlas_ref is available

atlas_source Loader/source key, when atlas_ref is available

atlas_confidence Atlas provenance confidence tier, when atlas_ref is available

Additional atlas-specific columns may be present (e.g., `network` for Schaefer atlases).

See Also

`roi_attributes` for listing available attributes, `filter_atlas` for filtering atlas objects by attributes

Examples

```
## Not run:
# Get metadata for Schaefer atlas
atlas <- get_schaefer_atlas(parcels = "200", networks = "7")
meta <- roi_metadata(atlas)

# Filter using dplyr
library(dplyr)
left_visual <- meta %>% filter(hemi == "left", network == "Vis")

## End(Not run)
```

schaefer_surf *Schaefer Surface Atlas*

Description

Load the Schaefer2018 cortical parcellation as a surface atlas, returning neurosurf `LabeledNeuroSurface` objects plus standard atlas metadata. By default this uses the packaged fsaverage6 geometry; other FreeSurfer surface spaces use TemplateFlow for mesh geometry.

Usage

```
schaefer_surf(
  parcels = c(100, 200, 300, 400, 500, 600, 800, 1000),
  networks = c(7, 17),
  space = c("fsaverage6", "fsaverage", "fsaverage5"),
  surf = c("inflated", "white", "pial"),
  use_cache = TRUE
)
```

Arguments

`parcels`	Number of parcels. Can be numeric or character; valid values are 100, 200, 300, 400, 500, 600, 800, 1000.
`networks`	Number of networks. Can be numeric or character; valid values are 7 or 17.
`space`	Surface space / mesh template. One of `"fsaverage6"` (default), `"fsaverage"`, or `"fsaverage5"`. Currently only `"fsaverage6"` uses packaged geometry; other spaces require a working TemplateFlow setup.
`surf`	Surface type. One of `"inflated"`, `"white"`, or `"pial"`.
`use_cache`	Logical. Passed to `schaefer_metainfo()` for label metadata caching.

Value

A list with classes `c("schaefer","surfatlas","atlas")` containing:

- `lh_atlas`, `rh_atlas`: `LabeledNeuroSurface` objects for left and right hemispheres.
- `surf_type`: requested surface type.
- `surface_space`: Schaefer surface space (e.g. fsaverage6).
- `ids`, `labels`, `orig_labels`, `network`, `hemi`, `cmap`: atlas metadata.

Examples

```
## Not run:
# Schaefer 200 parcels, 17 networks on fsaverage6 inflated surface
atl <- schaefer_surf(parcels = 200, networks = 17,
                     space = "fsaverage6", surf = "inflated")

## End(Not run)
```

schaefer_surf_options *List supported Schaefer surface atlas variants*

Description

Return a data frame enumerating the currently supported Schaefer surface atlas variants, including their CBIG and TemplateFlow identifiers.

Usage

```
schaefer_surf_options()
```

Value

A data frame with columns:

- `space`: Schaefer surface space (fsaverage, fsaverage5, fsaverage6).
- `parcels`: Number of parcels.
- `networks`: Number of networks.
- `surf`: Surface type ("inflated", "white", "pial").
- `cbig_space`: CBIG FreeSurfer5.3 subfolder.
- `template_id`, `tf_resolution`, `tf_density`: TemplateFlow identifiers used by `get_surface_template()`.

Examples

```
opts <- schaefer_surf_options()
head(opts)
```

show_templateflow_cache_path *Show neuroatlas TemplateFlow Cache Path*

Description

Returns the path to the TemplateFlow cache directory. With the pure R `templateflow` package, this is the `TEMPLATEFLOW_HOME` directory.

Usage

```
show_templateflow_cache_path()
```

Value

A character string representing the path to the TemplateFlow cache directory.

Examples

```
cat("TemplateFlow cache is at:", show_templateflow_cache_path(), "\n")
```

`space_transform_manifest` *Space Transform Manifest*

Description

Returns known transforms between coordinate/template spaces from a static registry shipped with the package.

Usage

```
space_transform_manifest(status = NULL)
```

Arguments

`status`	Optional character vector to filter by status (e.g., ‘"available"‘, ‘"planned"‘).

Value

A data frame with one row per transform route and columns: ‘from_space‘, ‘to_space‘, ‘transform_type‘, ‘backend‘, ‘confidence‘, ‘reversible‘, ‘data_files‘, ‘status‘, and ‘notes‘.

Examples

```
# All known routes
reg <- space_transform_manifest()

# Only implemented routes
available <- space_transform_manifest(status = "available")
```

`spin_test` *Spin Test for Spatial Correlation Significance*

Description

Tests whether the spatial correlation between two parcellated brain maps is stronger than expected by chance, using the spin-test framework of Alexander-Bloch et al. (2018). Parcel centroids on a sphere surface are randomly rotated and reassigned to the nearest original centroid, generating a null distribution of correlations.

Usage

```
spin_test(
  map1,
  map2,
  atlas,
  n_perm = 1000L,
  cor_method = c("pearson", "spearman", "kendall"),
  sphere = NULL,
  seed = NULL
)
```

Arguments

map1	Numeric vector of parcel values (length K, aligned to atlas parcels).
map2	Numeric vector of parcel values (length K, aligned to atlas parcels).
atlas	A surface atlas object (class "surfatlas") with lh_atlas and rh_atlas fields, or any atlas whose geometry can provide sphere coordinates.
n_perm	Integer. Number of spin permutations. Default: 1000.
cor_method	Character. Correlation method passed to cor. One of "pearson", "spearman", or "kendall". Default: "pearson".
sphere	Optional list with elements lh and rh, each an N x 3 matrix of sphere vertex coordinates. If NULL (default), sphere coordinates are fetched via load_surface_template(..., "sphere").
seed	Optional integer seed for reproducibility.

Details

The algorithm:

1. Compute parcel centroids on the sphere surface by averaging the sphere coordinates of vertices belonging to each parcel.
2. For each permutation, generate a uniform random 3D rotation matrix (from the Haar measure on SO(3)), rotate all centroids, then reassign each rotated centroid to its nearest original centroid using nn.
3. Compute the correlation of map1 with the permuted map2 (values shuffled according to the reassignment).
4. The p-value is (sum(abs(null) >= abs(observed)) + 1) / (n_perm + 1).

Value

A list of class "spin_test" with:

observed The observed correlation between map1 and map2.

null_distribution Numeric vector of length n_perm containing null correlations.

p_value Two-sided p-value: proportion of null correlations with absolute value >= the observed absolute correlation.

n_perm Number of permutations performed.

References

Alexander-Bloch, A. F., et al. (2018). On testing for spatial correspondence between maps of human brain structure and function. *NeuroImage*, 178, 540-551.

Examples

```
## Not run:
atlas <- get_schaefer_surfatlas(parcels = "100", networks = "7")
K <- length(atlas$ids)
map1 <- rnorm(K)
map2 <- map1 + rnorm(K, sd = 0.5)
res  <- spin_test(map1, map2, atlas, n_perm = 500, seed = 42)
print(res)

## End(Not run)
```

---

sub_atlas *Select a Subset of Atlas Regions*

Description

Return a new atlas containing only the requested regions. Regions can be identified by integer ID (matching `x$ids`), by label (matching `x$labels`), or by hemisphere. When multiple selection criteria are given they are intersected.

Usage

```
sub_atlas(x, ids = NULL, labels = NULL, hemi = NULL, ...)

## S3 method for class 'atlas'
sub_atlas(x, ids = NULL, labels = NULL, hemi = NULL, network = NULL, ...)
```

Arguments

`x`	An atlas object.
`ids`	Integer or numeric vector of region IDs to retain (matched against `x$ids`).
`labels`	Character vector of region labels to retain (matched against `x$labels`).
`hemi`	Character vector of hemispheres to retain (`"left"` and/or `"right"`).
`...`	Additional arguments passed to methods.

Details

This is the atlas-level analogue of `sub_clusters` for `ClusteredNeuroVol` objects.

Value

An atlas object of the same class as `x` containing only the selected regions. Voxels (or vertices) not belonging to the selected regions are excluded.

See Also

`filter_atlas` for tidy-eval filtering by metadata columns, `get_roi` for extracting ROI volumes

Examples

```
## Not run:
atlas <- get_aseg_atlas()

# By ID
sub <- sub_atlas(atlas, ids = c(10, 11, 12))

# By label
sub2 <- sub_atlas(atlas, labels = c("Thalamus", "Caudate"))

# By hemisphere
left <- sub_atlas(atlas, hemi = "left")

# Combined: left-hemisphere regions matching specific labels
sub3 <- sub_atlas(atlas, labels = c("Thalamus", "Caudate"), hemi = "left")

## End(Not run)
```

---

subcortical_atlas_options

Subcortical Atlas Options (TemplateFlow-backed)

Description

Returns a tibble describing the supported subcortical atlases built and harmonized by the AtlasPack resource (HCP thalamus, MDTB10 cerebellum, CIT168 subcortex, and HCP hippocampus/amygdala). Each row includes the TemplateFlow atlas identifier, supported template spaces, and default resolution/desc parameters used to fetch the data.

Usage

```
subcortical_atlas_options()
```

Value

A tibble with columns:

- `id`: canonical neuroatlas identifier

- `atlas`: TemplateFlow `atlas` field
- `label`: human-readable name
- `default_space`: default TemplateFlow space
- `spaces`: list-column of allowed spaces
- `default_resolution`: default TemplateFlow resolution string
- `resolutions`: list-column of allowed resolutions
- `default_desc`: optional TemplateFlow `desc` value

References

1. AtlasPack resource for harmonized atlases: https://github.com/PennLINC/AtlasPack 2. Najdenovska E. et al. (2018) Scientific Data, HCP thalamus. 3. King M. et al. (2019) Nature Neuroscience, MDTB10 cerebellum. 4. Pauli W.M. et al. (2018) Scientific Data, CIT168 subcortex. 5. Glasser M.F. et al. (2013) Neuroimage, HCP subcortical structures.

Examples

```
opts <- subcortical_atlas_options()
opts
```

`template_to_coord_space` *Get Coordinate Space for Any Template*

Description

Determine which standard coordinate space a template's data are defined in. Handles both surface templates (e.g., fsaverage) and volumetric templates (e.g., MNI152NLin6Asym).

Usage

```
template_to_coord_space(template_id)
```

Arguments

`template_id`	Character string identifying the template. Examples: "fsaverage", "fsaverage6", "fsLR_32k", "MNI152NLin6Asym", "MNI152NLin2009cAsym", "MNI152", "MNI305".

Details

This function normalizes the input using the same alias resolution as the space transform registry, then maps it to a coordinate space:

MNI305 fsaverage, fsaverage5, fsaverage6, MNI305

MNI152 fsLR_32k, MNI152, MNI152NLin6Asym, MNI152NLin2009cAsym

Value

Character string indicating the coordinate space (‘"MNI305"‘, ‘"MNI152"‘, or ‘"Unknown"‘).

See Also

[get_surface_coordinate_space()] for the surface-only version, [needs_coord_transform()] and [needs_template_warp()] for transform checks.

Examples

```
template_to_coord_space("fsaverage")            # "MNI305"
template_to_coord_space("MNI152NLin6Asym")      # "MNI152"
template_to_coord_space("MNI152NLin2009cAsym")   # "MNI152"
template_to_coord_space("fsLR_32k")              # "MNI152"
```

---

tflow_files *Find TemplateFlow Files Matching Metadata Criteria*

Description

Retrieves a list of file paths from TemplateFlow that match a given template space and other optional metadata query parameters.

Usage

```
tflow_files(space, query_args = list(), api_handle = NULL)
```

Arguments

space	Character string. The TemplateFlow identifier for the template space (e.g., "MNI152NLin2009cAsym").
query_args	(Optional) A named list of additional query parameters to filter the results (e.g., ‘list(suffix = "T1w", resolution = 1, desc = "brain")‘).
api_handle	Deprecated and ignored.

Value

A character vector of file paths matching the query. Returns an empty vector if no files match.

Examples

```
# List all T1w files for MNI152NLin2009cAsym template
# mni_t1w_files <- tflow_files("MNI152NLin2009cAsym",
#                              query_args = list(suffix = "T1w"))
```

tflow_spaces *List Available TemplateFlow Template Spaces*

Description

Retrieves a list of all available template space identifiers from the TemplateFlow archive.

Usage

```
tflow_spaces(pattern = NULL, api_handle = NULL, ...)
```

Arguments

pattern	(Optional) A character string containing a regular expression to filter the template space names. If ‘NULL‘ (default), all names are returned.
api_handle	Deprecated and ignored.
...	Additional arguments passed to ‘grep‘ if ‘pattern‘ is specified (e.g., ‘ignore.case = TRUE‘).

Value

A character vector of available template space names.

Examples

```
# List all template spaces
# all_spaces <- tflow_spaces()

# List template spaces containing "MNI"
# mni_spaces <- tflow_spaces(pattern = "MNI")
```

transform_coords *Transform Coordinates Between Spaces*

Description

Apply an affine transformation to convert 3D coordinates from one neuroimaging coordinate space to another.

Usage

```
transform_coords(
  coords,
  from = NULL,
  to = NULL,
  transform = NULL,
  coords_as_cols = FALSE
)
```

Arguments

coords	Numeric matrix of 3D coordinates to transform. Can be: • An N x 3 matrix (N points, each row is x/y/z) • A 3 x N matrix (if coords_as_cols = TRUE) • A length-3 vector (single point)
from	Character string specifying the source coordinate space. One of "MNI305", "MNI152".
to	Character string specifying the target coordinate space. One of "MNI305", "MNI152".
transform	Optional 4x4 affine matrix. If provided, from and to are ignored and this matrix is applied directly.
coords_as_cols	Logical. If TRUE, coordinates are in columns (3 x N matrix). If FALSE (default), coordinates are in rows (N x 3 matrix).

Details

The transformation is applied as: p_new = M %*% [p; 1] where M is the 4x4 affine matrix and p is each 3D point in homogeneous coordinates.

Value

Transformed coordinates in the same format as the input.

See Also

get_space_transform, MNI305_to_MNI152

Examples

```
# Transform a single point
p305 <- c(10, -20, 35)
p152 <- transform_coords(p305, from = "MNI305", to = "MNI152")
print(p152)

# Transform multiple points (N x 3 matrix)
points_305 <- rbind(
  c(10, -20, 35),
  c(0, 0, 0),
```

```
  c(-30, 15, 50)
)
points_152 <- transform_coords(points_305, from = "MNI305", to = "MNI152")
print(points_152)

# Round-trip: MNI305 -> MNI152 -> MNI305
p_roundtrip <- transform_coords(
  transform_coords(p305, "MNI305", "MNI152"),
  "MNI152", "MNI305"
)
all.equal(p305, p_roundtrip, tolerance = 1e-10)
```

`transform_vertices_to_volume` *Transform Surface Vertices to Volume Space*

Description

Convenience function to transform surface vertex coordinates to match a target volumetric coordinate space.

Usage

```
transform_vertices_to_volume(
  vertices,
  surface_template,
  target_space = "MNI152"
)
```

Arguments

`vertices`	Numeric matrix of vertex coordinates (N x 3).
`surface_template`	Character string identifying the surface template the vertices come from (e.g., "fsaverage", "fsLR").
`target_space`	Character string identifying the target coordinate space (e.g., "MNI152"). Default is "MNI152".

Details

This is a convenience wrapper around `transform_coords` that automatically determines the source space from the surface template.

Value

Transformed vertex coordinates (N x 3 matrix). Returns the input unchanged if no transform is needed.

See Also

transform_coords, get_surface_coordinate_space

Examples

```
## Not run:
# Load fsaverage surface
surf <- load_surface_template("fsaverage", "white", hemi = "L")
verts <- neurosurf::vertices(surf)

# Transform vertices to MNI152 for sampling from fMRI volume
verts_mni152 <- transform_vertices_to_volume(verts, "fsaverage", "MNI152")

## End(Not run)
```

---

validate_atlas_ref *Validate an Atlas Reference*

Description

Validate an Atlas Reference

Usage

```
validate_atlas_ref(x)
```

Arguments

x	Object to validate.

Value

Invisibly returns ‘x‘ when valid.

---

validate_parcel_data *Validate a Parcel-Level Data Object*

Description

Validate structure and key invariants for ‘parcel_data‘ objects.

Usage

```
validate_parcel_data(x, strict = TRUE)
```

Arguments

x	An object expected to be ‘parcel_data‘.
strict	Logical. If ‘TRUE‘ (default), enforce strict checks on atlas metadata consistency.

Value

Invisibly returns ‘x‘ if valid; otherwise throws an error.

---

write_parcel_data *Write a ‘parcel_data‘ Object to Disk*

Description

Write a ‘parcel_data‘ Object to Disk

Usage

```
write_parcel_data(x, file, format = c("auto", "rds", "json"), pretty = TRUE)
```

Arguments

x	A ‘parcel_data‘ object.
file	Output file path.
format	Serialization format: ‘"auto"‘, ‘"rds"‘, or ‘"json"‘.
pretty	Logical; pretty-print JSON output when ‘format = "json"‘.

Value

Invisibly returns normalized output path.

Index