

Package: neuroatlas (via r-universe)

October 7, 2026

Title Neuroimaging Atlases and Parcellations

Version 0.2.0.9000

Description Provides a unified interface to access and work with various neuroimaging atlases and parcellations including Schaefer, Brainnetome, Glasser, HCPex, 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, digest, dplyr, ggiraph, ggplot2, grid, lifecycle, magrittr, memoise, methods, neuroim2 (>= 0.19.0), rlang, Rnanoflann, scales, scico, stringr, templateflow, tibble, tools, utils

Suggests albersdown, corpcor, DT, echarts4r, FNN, future.apply, geojsonio, geojsonsf, ggseg, ggsegSchaefer, gifti, hdf5r, htmltools, igraph, jsonlite, knitr, mockery, neurosurf (>= 0.1.0), neurotransform (>= 0.2.0), patchwork, png, ragg, reticulate, RNifti, rmarkdown, shiny, sf, systemfonts, testthat (>= 3.0.0), tidyr, uwot, withr, xml2

VignetteBuilder knitr

Remotes bbuchsbaum/neuroim2,
bbuchsbaum/neurosurf@b8bb05d01dcf7360141850ad56b173d7df2f8079,
bbuchsbaum/neurotransform@933edddd462593941e167726e8aaa7168ff103a,
bbuchsbaum/albersdown, bbuchsbaum/templateflow,
ggsegverse/ggsegSchaefer

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_catalog.R'
 'atlas_colors.R' 'atlas_connectivity.R' 'atlas_constructor.R'
 'atlas_download.R' 'atlas_graph.R' 'atlas_hierarchy.R'
 'atlas_metadata.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' 'cifti.R' 'cluster_explorer.R'
 'coordinate_spaces.R' 'data-schaefer.R' 'dilate_parcel.R'
 'fsaverage.R' 'fsl_atlas.R' 'ggschaefer.R' 'glasser.R'
 'hcpex.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' 'render_parcel_surface.R'
 'render_stat_surface.R' 'roi_colors.R' 'roi_metadata.R'
 'schaefer.R' 'spin_test.R' 'subcortical_atlases.R'
 'surface_assets.R' 'surface_domain.R' 'surface_projection.R'
 'surface_transform.R' 'template_flow.R' 'template_metadata.R'
 'template_transform.R' 'transform_cache.R'
 'transform_registry.R' 'visfatlas.R' 'visual_atlas.R'
 'wang_atlas.R' 'zzz.R'

Config/pak/sysreqs libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev make libicu-dev libpng-dev libuv1-dev libssl-dev

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

Date/Publication 2026-10-07 20:58:22 UTC

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

RemoteRef HEAD

RemoteSha 384308eaecb502fe6b23680b54625c4a7904ebf4

Contents

%where%	5
align_parcel_values	6
apply_cifti_transform	8
apply_surface_projection	9
apply_surface_transform	10
apply_template_transform	11
as_igraph	13

as_parcel_data	13
atlas_alignment	15
atlas_artifacts	16
atlas_citations	17
atlas_connectivity	17
atlas_coord_space	19
atlas_family	19
atlas_graph	20
atlas_hierarchy	21
atlas_history	22
atlas_metadata	23
atlas_overlap	24
atlas_provenance	26
atlas_ref	27
atlas_roi_colors	28
atlas_space	29
atlas_transform_manifest	30
atlas_transform_plan	30
batch_reduce	31
brainnetome_labels	33
build_brain_polygon_data	34
build_cluster_explorer_data	35
build_conflict_edges	37
build_surface_polygon_data	38
check_templateflow	40
clear_templateflow_cache	40
clear_transform_cache	41
cluster_explorer	41
coord_spaces	44
coordinate_spaces	45
create_templateflow	46
dilate_atlas	46
filter_atlas	48
fsaverage	49
get_aseg_atlas	50
get_atlas	51
get_brainnetome_atlas	52
get_cifti_transform	53
get_fsl_atlas	54
get_ggseg_atlas	55
get_glasser_atlas	56
get_harvard_oxford_atlas	57
get_hcpex_atlas	58
get_hipp_atlas	60
get_julich_brain_atlas	61
get_olsen_mtl	62
get_roi	62
get_roi.surfatlas	63

get_schaefer_atlas	64
get_schaefer_surfatlas	70
get_space_transform	72
get_subcortical_atlas	73
get_surface_coordinate_space	74
get_surface_geometry	75
get_surface_projection	76
get_surface_transform	77
get_template	79
get_template_transform	81
get_visfatlas	82
get_visual_atlas	84
get_wang_atlas	85
get_wang_prob_atlas	86
ggseg_schaefer	89
glasser_surf	90
infer_design_var_type	91
install_templateflow	92
launch_cluster_explorer	92
list_atlases	93
load_surface_template	94
map_atlas	95
map_to_schaefer	96
merge_atlases	97
MNI152_to_MNI305	98
MNI305_to_MNI152	99
needs_coord_transform	100
needs_template_warp	101
needs_transform	102
network_anchor_hues	102
new_atlas_ref	103
olsen_mtl	104
parcel_data	106
parcel_values	107
parcel_volume	108
plot.atlas	109
plot_brain	111
plot_brain_grid	118
plot_glasser	121
print.atlas	121
print.atlas_alignment	122
print.atlas_ref	123
print.atlas_transform_plan	124
print.templateflow	124
project_cluster_overlay	125
project_surface_view	127
projection_diagnostics	128
query_coord	129

query_point	130
read_cifti	132
read_parcel_data	133
reduce_atlas	133
reduce_atlas_vec	135
replace_cifti_values	137
resample	137
ribbon_projection	139
roi_attributes	140
roi_colors_embedding	141
roi_colors_maximin_view	142
roi_colors_network_harmony	143
roi_colors_rule_hcl	145
roi_metadata	146
schaefer_surf	148
schaefer_surf_options	149
show_templateflow_cache_path	150
space_transform_manifest	150
spin_test	151
sub_atlas	153
subcortical_atlas_options	154
surface_anatomy	155
surface_data	156
surface_domain	157
surface_geometry	159
template_to_coord_space	160
tflow_files	161
tflow_spaces	161
transform_atlas	162
transform_cache_path	163
transform_coords	164
transform_vertices_to_volume	165
validate_atlas_ref	166
validate_parcel_data	167
write_cifti	167
write_parcel_data	168

Index**170**

%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)
```

align_parcel_values *Align a Parcel Value Column to an Atlas*

Description

Match a data-frame column to atlas parcels and return a numeric vector in exact ‘atlas\$ids’ order. Matching is strict by default: atlas and data keys must be unique, data keys must all exist in the atlas, and every atlas parcel must be represented unless ‘allow_partial = TRUE’.

Usage

```
align_parcel_values(atlas, data, value, by = NULL, allow_partial = FALSE)
```

Arguments

<code>atlas</code>	An atlas object.
<code>data</code>	A data frame, tibble, or ‘ <code>parcel_data</code> ’ object containing parcel keys and values.
<code>value</code>	A numeric value column, supplied as a bare name or string.
<code>by</code>	Parcel-key specification. Use a shared column name such as “ <code>id</code> ”, or a named character vector to map an atlas metadata column to a differently named data column, for example ‘ <code>c(id = "roi_index")</code> ’. Composite keys are supported. When ‘ <code>NULL</code> ’, a safe unique key is inferred.
<code>allow_partial</code>	Logical. If ‘ <code>FALSE</code> ’ (default), ‘ <code>data</code> ’ must contain every atlas parcel. If ‘ <code>TRUE</code> ’, unmatched atlas parcels receive ‘ <code>NA</code> ’. Unknown and duplicate data keys always error.

Details

With ‘`by = NULL`’, ‘`id`’ is preferred, followed by ‘`label_full`’. Composite label keys are considered only when they are unique in both the atlas and data. Ambiguous short labels therefore fail instead of being recycled across hemispheres or networks.

Atlas metadata columns supplied in ‘`data`’ are treated as consistency checks. They are never allowed to overwrite canonical metadata.

Glasser volume and surface IDs number opposite hemispheres first. Any Glasser join using `id` (including renamed or composite keys) therefore requires a matching `id_convention` column, or a `parcel_data` object with a matching `atlas$id_convention`. The convention is “`hcp_R_first`” for volumes and “`surfatlas_L_first`” for surfaces. Only declare a convention when the source of the IDs is known; missing or conflicting conventions cause an error. Reload older Glasser atlas objects that lack this metadata.

To move Glasser values between representations, supply only the shared `label_full` key and value columns, then use `by = "label_full"`; alternatively use `by = c("area", "hemi")`. For example, `select source$parcels[c("label_full", "value")]` from a `parcel_data` object. Representation-specific IDs, labels, colours and provenance in a full source table remain consistency checks and must be omitted for such a join. A complete `parcel_data` object remains bound to its source representation.

Value

A numeric vector named by parcel ID and ordered exactly like ‘`atlas$ids`’.

Examples

```
atlas <- structure(
  list(
    name = "toy",
    atlas = array(1:3, c(3, 1, 1)),
    ids = 1:3,
    labels = c("A", "B", "C"),
    orig_labels = c("lh_A", "lh_B", "rh_C"),
    hemi = c("left", "left", "right")
  ),
)
```

```

    class = "atlas"
  )
  results <- data.frame(
    roi_index = c(3L, 1L, 2L),
    estimate = c(0.3, 0.1, 0.2)
  )
  align_parcel_values(
    atlas,
    results,
    value = estimate,
    by = c(id = "roi_index")
  )

```

apply_cifti_transform *Apply Cortical Resampling While Preserving CIFTI Subcortex*

Description

Applies the declared qualified surface operator independently to each map. Noncortical values and their availability are copied by structure/index. Source map metadata and label tables are retained. Missing scalar values stay missing. Missing label values require explicit existing label keys, and their unavailable brainordinate indices are retained in per-map XML metadata and available; assigning a key does not establish sampled support.

Usage

```

apply_cifti_transform(
  x,
  transform,
  na_policy = c("propagate", "omit", "error"),
  label_method = c("aggregate", "largest"),
  missing_labels = NULL
)

```

Arguments

<code>x</code>	Source CiftiData with the bound brain-model layout.
<code>transform</code>	A CiftiTransform from get_cifti_transform() .
<code>na_policy</code>	Missing-value policy for cortical interpolation, passed to apply_surface_transform() .
<code>label_method</code>	Categorical interpolation policy, passed to apply_surface_transform() .
<code>missing_labels</code>	NULL rejects unsupported label samples. Otherwise, supply one existing label key per map, or a single key shared by every map.

Value

A CiftiData on the target layout with original map metadata, per-brainordinate availability and cortical execution provenance.

See Also

[write_cifti\(\)](#), [apply_template_transform\(\)](#)

Examples

```
## Not run:
mapped <- apply_cifti_transform(source, op, missing_labels = 0)
write_cifti(mapped, "mapped.dlabel.nii")

## End(Not run)
```

apply_surface_projection

Sample a Volume onto an Explicit Cortical Projection

Description

Scalar and probability data use trilinear interpolation; labels use nearest voxel centers with lower-index half-voxel ties. Sampling is restricted to the closed voxel-center box; unsupported values are NA, and supported zero is preserved. Probability channels retain partial mass and are not normalized across maps. Missingness considers strictly positive contributors only. fsLR composition resolves sampling missingness on fsaverage first, then resolves missing surface contributors with the native operator. Coverage for each stage is retained separately. Ribbon omission combines voxel and node weights within the sampling stage.

Usage

```
apply_surface_projection(
  x,
  projection,
  source_space = NULL,
  affine = NULL,
  data_type = c("auto", "continuous", "label", "probability"),
  na_policy = c("propagate", "omit", "error"),
  label_table = NULL
)
```

Arguments

x	A 3D/4D numeric array, NeuroVol, NeuroVec, or volumetric atlas.
projection	A SurfaceProjection from get_surface_projection() or ribbon_projection() .
source_space	Exact frame identifier. Required for unannotated input; an explicit value must agree with attached metadata and the projection.
affine	For arrays, a finite nonsingular 4-by-4 matrix mapping zero-based voxel indices to RAS millimetres. Neuroimaging objects supply their own affine.

data_type	"auto", "continuous", "label", or "probability". Unannotated input requires an explicit type.
na_policy	"propagate", "omit" with finite-weight normalization, or "error" for a missing positive contributor in included cortical support. Outside-grid samples remain unsupported under every policy.
label_table	Optional key/name/color table preserved on output. An optional hemisphere column declares "L", "R", "both", or NA (unknown); left/right and LH/RH aliases are accepted. Hemisphere checks use these declarations, never label names or numeric key ranges.

Value

SurfaceData with values, exact target domain, per-map availability, finite source-weight mass, geometric sampling coverage, lost labels and directed projection provenance. Ribbon omission normalizes all finite voxel/node weights together; categorical nodes vote with smallest-key ties. [projection_diagnostics\(\)](#) exposes per-map coverage and per-key counts at the source, sampled surface and final surface. These are representation counts, not anatomical accuracy or conservation across dimensions.

Examples

```
## Not run:
result <- apply_surface_projection(volume, projection,
  source_space = "MNI152Nlin6Asym", data_type = "probability"
)

## End(Not run)
```

apply_surface_transform

Apply a Domain-Bound Surface Transform

Description

Continuous maps use row-normalized interpolation. Label keys use categorical voting and are never averaged. Unsupported and target-masked outputs are 'NA'; supported zero, including label key zero, remains a valid value. Missingness and source weight mass are returned separately for every map. Probability output roundoff within '1e-12' of [0,1] is clipped to that interval; larger excursions are errors. Input bounds remain strict and probability channels are never renormalized across maps.

Usage

```
apply_surface_transform(
  x,
  transform,
  na_policy = c("propagate", "omit", "error"),
  label_method = c("aggregate", "largest")
)
```

Arguments

x	A ‘SurfaceData’ object bound to the transform’s exact source domain.
transform	A verified ‘SurfaceTransform’.
na_policy	“propagate”, “omit” with finite-weight renormalization, or “error”. Omission changes the effective operator per map.
label_method	“aggregate” votes by total weight per key; ties select the smallest key. “largest” selects the largest-weight source vertex; ties select the smallest source index. Used only for categorical data.

Value

‘SurfaceData’ on the target domain, with ‘coverage’ diagnostics and provenance including input identity, operator identity and policies.

Examples

```
sphere <- rbind(
  c(1, 0, 0), c(-1, 0, 0), c(0, 1, 0),
  c(0, -1, 0), c(0, 0, 1), c(0, 0, -1)
)
faces <- rbind(
  c(0, 2, 4), c(2, 1, 4), c(1, 3, 4), c(3, 0, 4),
  c(2, 0, 5), c(1, 2, 5), c(3, 1, 5), c(0, 3, 5)
)
cortex <- rep(TRUE, 6)
domain <- surface_domain(
  "toy", "L", "6v", sphere, faces, cortex,
  "analytic", "v1"
)
if (requireNamespace("neurotransform", quietly = TRUE) &&
    utils::packageVersion("neurotransform") >= "0.2.0") {
  geometry <- surface_geometry(domain, sphere, faces, cortex)
  operator <- get_surface_transform(geometry, geometry, cache_dir = NULL)
  values <- surface_data(seq(0, 0.5, length.out = 6), domain)
  apply_surface_transform(values, operator)
}
```

apply_template_transform

Apply a Verified Template or Surface Transform

Description

Composes the complete route in physical coordinates and samples each input channel once. Labels use nearest neighbour; scalar and probability values use linear interpolation. For image-to-image transforms, out-of-field samples are zero and probability channels are interpolated independently. Surface transforms dispatch to [apply_surface_transform\(\)](#) or [apply_surface_projection\(\)](#), whose unsupported samples are NA and whose coverage and missingness semantics are documented separately.

Usage

```

apply_template_transform(
  x,
  transform,
  target = NULL,
  data_type = c("auto", "continuous", "label", "probability"),
  interpolation = NULL,
  missing_labels = NULL
)

```

Arguments

<code>x</code>	A volumetric atlas, <code>NeuroVol</code> , or <code>NeuroVec</code> (channels in dimension 4), or <code>SurfaceData</code> for a native surface operator. A mixed cortical adapter requires <code>CiftiData</code> with its bound source layout.
<code>transform</code>	A verified <code>get_template_transform()</code> result.
<code>target</code>	Target atlas, <code>NeuroVol</code> , or explicit <code>NeuroSpace</code> . A bare grid is an assertion by the caller that it is in the transform's target space. Attached source/target metadata must agree with the route. Surface operators already bind their exact target domain: <code>NULL</code> uses that domain, or supply the matching <code>SurfaceGeometry</code> or <code>SurfaceDomain</code> .
<code>data_type</code>	"auto" uses declared metadata, or label semantics for atlas objects. Unannotated volumes require an explicit type; integer-valued samples alone do not imply labels.
<code>interpolation</code>	<code>NULL</code> selects the type-specific method. Only "nearest" for labels and "linear" for continuous/probability data are supported.
<code>missing_labels</code>	Explicit unassigned keys for unsupported CIFTI label samples, passed to <code>apply_cifti_transform()</code> . Other representations require <code>NULL</code> .

Value

The transformed atlas or volume. The `neuroatlas_transform` attribute records the route, artifact hashes, interpolation, grids and lost labels. Atlas objects retain semantic IDs, labels and source provenance, including regions that disappear on the target grid. A surface destination returns `SurfaceData` with exact domain, availability and projection provenance. A CIFTI destination returns `CiftiData` and preserves noncortical support.

Examples

```

if (requireNamespace("neurotransform", quietly = TRUE) &&
    requireNamespace("hdf5r", quietly = TRUE)) {
  grid <- neuroim2::NeuroSpace(c(2, 2, 2))
  volume <- neuroim2::NeuroVol(array(0.25, c(2, 2, 2)), grid)
  transform <- get_template_transform("MNI152Nlin6Asym", "MNI152Nlin6Asym")
  apply_template_transform(volume, transform, grid, data_type = "continuous")
}

```

as_igraph	<i>Convert Atlas Connectivity to igraph</i>
-----------	---

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.

Examples

```
if (requireNamespace("igraph", quietly = TRUE)) {
  connectivity <- structure(matrix(c(0, 0.5, 0.5, 0), 2),
    class = c("atlas_connectivity", "matrix", "array")
  )
  as_igraph(connectivity)
}
```

as_parcel_data	<i>Convert an Object to 'parcel_data'</i>
----------------	---

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",
  by = NULL,
  allow_partial = FALSE,
  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\$sids)' - data frame/tibble with a stable parcel key and one or more value columns. Use 'by' for renamed or composite keys.
value_col	Column name used when 'values' is a vector.
by	Parcel-key specification used when 'values' is a data frame. Unnamed values join columns with the same name, for example "id" or c("label", "hemi", "network"). A named character vector maps atlas metadata columns to columns in 'values', following dplyr join syntax, for example c(id = "roi_index"). When 'NULL', a safe unique key is inferred.
allow_partial	Logical. If 'FALSE' (default), a data-frame input must contain exactly one row for every atlas parcel. If 'TRUE', missing parcels are retained with 'NA' values. Rows whose keys do not occur in the atlas always produce an error.
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.

Details

Glasser ID joins require a matching `id_convention` in the input table or `parcel_data` atlas meta-data. See [align_parcel_values\(\)](#) for safe joins between Glasser volume and surface representations. Numeric vectors are positional: they must already follow the target atlas's parcel order.

Value

An object of class `"parcel_data"`.

Examples

```
x <- parcel_data(data.frame(
  id = 1:2, label = c("A", "B"),
  hemi = c("left", "right"), value = c(0.2, 0.7)
), atlas_id = "toy")
as_parcel_data(x)
```

atlas_alignment	<i>Atlas Alignment Lookup</i>
-----------------	-------------------------------

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	<i>Get Atlas Artifact Metadata</i>
-----------------	------------------------------------

Description

Get Atlas Artifact Metadata

Usage

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

Arguments

<code>x</code>	An atlas object.
<code>...</code>	Additional arguments passed to methods.

Value

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

Examples

```

grid <- neuroim2::NeuroSpace(c(2, 1, 1))
volume <- neuroim2::NeuroVol(array(c(1, 2), c(2, 1, 1)), grid)
atlas <- structure(
  list(
    atlas = volume, ids = 1:2,
    labels = c("A", "B"), orig_labels = c("A", "B"),
    hemi = c("left", "right"), name = "toy",
    cmap = rbind(c(1, 0, 0), c(0, 0, 1)),
    atlas_ref = new_atlas_ref("toy", "two-regions",
      template_space = "MNI152NLin6Asym", coord_space = "MNI152"
    )
  ),
  class = "atlas"
)
atlas_artifacts(atlas)

```

atlas_citations	<i>Bibliographic References for an Atlas or Template</i>
-----------------	--

Description

Convert the stored citation table to R bibliography entries. These accessors never resolve DOIs or contact a service. References without complete article metadata remain ‘Misc’ entries with their recorded title and URL.

Usage

```
atlas_citations(x, role = NULL)
```

```
template_citations(x, role = NULL)
```

Arguments

x	An atlas or loaded template object.
role	Optional citation role(s), such as “atlas”, “distribution”, “template”, or “software”. ‘NULL’ includes all roles.

Value

A ‘bibentry’ vector, suitable for [utils::toBibtex()]. Each entry’s note identifies its role, and its key is derived from the stored DOI, URL, or title. The table with separate role rows is available through [atlas_metadata()] or [template_metadata()].

Examples

```
refs <- atlas_citations(get_aseg_atlas())
refs
utils::toBibtex(refs)
```

atlas_connectivity	<i>Compute Connectivity Matrix from Atlas Parcellations</i>
--------------------	---

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

<code>data_vol</code>	A NeuroVec (4D) containing the time-series data.
<code>atlas</code>	An atlas object defining the parcellation.
<code>method</code>	Character string specifying the correlation method: "pearson" (default), "spearman", or "partial". Partial correlation requires the corpcor package.
<code>threshold</code>	Optional numeric value. If supplied, entries with $\text{abs}(r) < \text{threshold}$ are set to zero.
<code>stat_func</code>	Function used to summarise voxel values within each parcel (default: mean).
<code>...</code>	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	<i>Atlas Coordinate-Space Convenience Accessor</i>
-------------------	--

Description

Atlas Coordinate-Space Convenience Accessor

Usage

```
atlas_coord_space(x)
```

Arguments

`x` An atlas object.

Value

Character scalar (or 'NA_character_').

Examples

```
grid <- neuroim2::NeuroSpace(c(2, 1, 1))
volume <- neuroim2::NeuroVol(array(c(1, 2), c(2, 1, 1)), grid)
atlas <- structure(
  list(
    atlas = volume, ids = 1:2,
    labels = c("A", "B"), orig_labels = c("A", "B"),
    hemi = c("left", "right"), name = "toy",
    cmap = rbind(c(1, 0, 0), c(0, 0, 1)),
    atlas_ref = new_atlas_ref("toy", "two-regions",
      template_space = "MNI152Nlin6Asym", coord_space = "MNI152"
    )
  ),
  class = "atlas"
)
atlas_coord_space(atlas)
```

atlas_family	<i>Atlas Family Convenience Accessor</i>
--------------	--

Description

Atlas Family Convenience Accessor

Usage

```
atlas_family(x)
```

Arguments

`x` An atlas object.

Value

Character scalar.

Examples

```
grid <- neuroim2::NeuroSpace(c(2, 1, 1))
volume <- neuroim2::NeuroVol(array(c(1, 2), c(2, 1, 1)), grid)
atlas <- structure(
  list(
    atlas = volume, ids = 1:2,
    labels = c("A", "B"), orig_labels = c("A", "B"),
    hemi = c("left", "right"), name = "toy",
    cmap = rbind(c(1, 0, 0), c(0, 0, 1)),
    atlas_ref = new_atlas_ref("toy", "two-regions",
      template_space = "MNI152NLin6Asym", coord_space = "MNI152"
    )
  ),
  class = "atlas"
)
atlas_family(atlas)
```

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	<i>Atlas Hierarchy</i>
-----------------	------------------------

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(parcel = "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.

Examples

```
grid <- neuroim2::NeuroSpace(c(2, 1, 1))
volume <- neuroim2::NeuroVol(array(c(1, 2), c(2, 1, 1)), grid)
atlas <- structure(
  list(
    atlas = volume, ids = 1:2,
    labels = c("A", "B"), orig_labels = c("A", "B"),
    hemi = c("left", "right"), name = "toy",
    cmap = rbind(c(1, 0, 0), c(0, 0, 1)),
    atlas_ref = new_atlas_ref("toy", "two-regions",
      template_space = "MNI152NLin6Asym", coord_space = "MNI152"
    )
  ),
  class = "atlas"
)
atlas_history(atlas)
```

atlas_metadata	<i>Atlas and Template Metadata</i>
----------------	------------------------------------

Description

A versioned, portable description of a loaded resource. Metadata are stored with the object and can be inspected without downloading files or resolving citations. Atlas objects store the record in 'x\$metadata'; template volumes and surfaces use the 'neuroatlas_metadata' attribute, preserving their class.

Usage

```
atlas_metadata(x, ...)

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

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

## S3 method for class 'wang_prob_volumes'
atlas_metadata(x, ...)

template_metadata(x, ...)

validate_resource_metadata(x)

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

Arguments

<code>x</code>	An atlas, or a template returned by <code>[get_template()]</code> or <code>[load_surface_template()]</code> . For validation, a <code>'NeuroResourceMetadata'</code> record.
<code>...</code>	Reserved for methods.

Details

`'identity'` describes the resource and its published version (unknown versions are `'NA'`). `'content'` describes the loaded representation, value type, selected parameters and current region count. `'spatial'` describes the current sampling geometry; `'artifacts'` retain source-space and resolution descriptors. A `TemplateFlow` resolution key is a query parameter, not a voxel size in millimetres.

`'spatial$basis'` records whether template identity is source-declared, inferred, user-supplied, or unknown. Geometry alone does not identify an anatomical template. Grid resampling does not establish registration to a different anatomical template.

`'citations'` is a table with reference roles; `[atlas_citations()]` and `[template_citations()]` convert it to R `'bibentry'` objects. `'history'` contains structured parameters and software versions. `'parents'` retains metadata snapshots for composite resources. Missing historical metadata in legacy objects is reported conservatively; it is not reconstructed from the network.

The older `[atlas_ref()]`, `[atlas_artifacts()]` and `[atlas_history()]` accessors read this authoritative record. Legacy list fields are compatibility copies; do not edit them to change metadata. Metadata preservation is guaranteed for neuroatlas loaders and supported atlas operations, not arbitrary operations in other packages or direct replacement of an object's data.

Value

`'atlas_metadata()'` and `'template_metadata()'` return a `'NeuroResourceMetadata'` list with `'schema_version'`, `'kind'`, `'identity'`, `'content'`, `'spatial'`, `'provenance'`, `'citations'`, `'artifacts'`, `'history'`, and `'parents'`. `'validate_resource_metadata()'` returns its input invisibly.

Examples

```
a <- get_aseg_atlas()
atlas_metadata(a)
atlas_metadata(a)$spatial$voxel_size
atlas_citations(a)
```

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	<i>Atlas Provenance Accessors</i>
------------------	-----------------------------------

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

<code>x</code>	An atlas object.
<code>...</code>	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.

Examples

```

grid <- neuroim2::NeuroSpace(c(2, 1, 1))
volume <- neuroim2::NeuroVol(array(c(1, 2), c(2, 1, 1)), grid)
atlas <- structure(
  list(
    atlas = volume, ids = 1:2,
    labels = c("A", "B"), orig_labels = c("A", "B"),
    hemi = c("left", "right"), name = "toy",
    cmap = rbind(c(1, 0, 0), c(0, 0, 1)),
    atlas_ref = new_atlas_ref("toy", "two-regions",
      template_space = "MNI152NLin6Asym", coord_space = "MNI152"
    )
  ),
  class = "atlas"
)
atlas_provenance(atlas)

```

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"`.

Examples

```
grid <- neuroim2::NeuroSpace(c(2, 1, 1))
volume <- neuroim2::NeuroVol(array(c(1, 2), c(2, 1, 1)), grid)
atlas <- structure(
  list(
    atlas = volume, ids = 1:2,
    labels = c("A", "B"), orig_labels = c("A", "B"),
    hemi = c("left", "right"), name = "toy",
    cmap = rbind(c(1, 0, 0), c(0, 0, 1)),
    atlas_ref = new_atlas_ref("toy", "two-regions",
      template_space = "MNI152Nlin6Asym", coord_space = "MNI152"
    )
  ),
  class = "atlas"
)
atlas_ref(atlas)
```

atlas_roi_colors	<i>Assign optimal colours to atlas regions</i>
------------------	--

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

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

Examples

```
grid <- neuroim2::NeuroSpace(c(2, 1, 1))
volume <- neuroim2::NeuroVol(array(c(1, 2), c(2, 1, 1)), grid)
atlas <- structure(
  list(
    atlas = volume, ids = 1:2,
    labels = c("A", "B"), orig_labels = c("A", "B"),
    hemi = c("left", "right"), name = "toy",
    cmap = rbind(c(1, 0, 0), c(0, 0, 1)),
    atlas_ref = new_atlas_ref("toy", "two-regions",
      template_space = "MNI152Nlin6Asym", coord_space = "MNI152"
    )
  ),
  class = "atlas"
)
atlas_space(atlas)
```

```
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 transform route 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"),
  available_only = FALSE,
  provider = c("auto", "neuroatlas", "templateflow")
)
```

Arguments

from_space	Source space identifier or a [surface_domain()] descriptor.
to_space	Target space identifier or a [surface_domain()] descriptor. Supply descriptors for both endpoints for native surface resampling, or an exact volume-template identifier and target descriptor for projection. Only identical domain descriptors establish surface identity. Named surface routes remain advisory: template names do not bind exact meshes or methods.
data_type	Data type being transformed ("parcel", "vertex", "voxel"). Voxel routes stay in volumes; vertex routes stay on surfaces. Parcel planning can also describe a single directed projection. Projection steps are not automatically composed with other routes.
mode	Planning mode. "auto" returns 'NULL' if no route exists, "strict" errors.
available_only	Restrict routing to available edges. The default also shows planned routes for diagnostic compatibility. Execution always uses available, executable edges only; retired edges are never selected.
provider	Provider filter, "auto", "neuroatlas", or "templateflow".

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

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

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>

Examples

```
## Not run:  
labels <- brainnetome_labels()  
head(labels)  
  
## End(Not run)
```

 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,
  depth_cull = TRUE
)
```

Arguments

surfatlas	A surfatlas object (e.g. from schaefer_surf()).
views	Character vector of views to render (e.g. <code>c("lateral", "medial")</code>).
surface	Character: surface type ("inflated", "pial", "white").
projection_smooth	Integer: number of Laplacian smoothing iterations on projected coordinates (default 0L).
depth_cull	Logical. If TRUE (default), remove faces hidden behind nearer cortical surface faces in the projected view.

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`.

Examples

```
## Not run:
atlas <- get_schaefer_surfatlas(100)
polygons <- build_brain_polygon_data(atlas,
  views = "lateral",
  surface = "inflated"
)

## End(Not run)
```

```
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 <code>neuroim2::series(data_source, i)</code> where <code>i</code> is voxel coordinates ($n \times 3$ 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 <code>.sample_index</code> 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 <code>neuroim2::conn_comp()</code> .
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_parcel 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.

Examples

```
grid <- neuroim2::NeuroSpace(c(2, 1, 1))
volume <- neuroim2::NeuroVol(array(c(1, 2), c(2, 1, 1)), grid)
atlas <- structure(
  list(
    atlas = volume, ids = 1:2,
    labels = c("A", "B"), orig_labels = c("A", "B"),
    hemi = c("left", "right"), name = "toy",
    cmap = rbind(c(1, 0, 0), c(0, 0, 1)),
    atlas_ref = new_atlas_ref("toy", "two-regions",
      template_space = "MNI152NLin6Asym", coord_space = "MNI152"
    )
  ),
  class = "atlas"
)
samples <- array(rep(c(0.2, 0.7), 3), c(2, 1, 1, 3))
data_source <- neuroim2::NeuroVec(
  samples, neuroim2::NeuroSpace(dim(samples))
)
```

```

)
stat_map <- neuroim2::NeuroVol(array(c(4, 5), c(2, 1, 1)), grid)
clusters <- build_cluster_explorer_data(data_source, atlas, stat_map,
  threshold = 3, min_cluster_size = 1, tail = "positive"
)
clusters$cluster_table

```

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 $\setminus[0, 1]$.

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,
  depth_cull = 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.
depth_cull	Logical. If TRUE (default), remove faces hidden behind nearer cortical surface faces in the projected view. This prevents far-side folds from shining through medial/lateral panels.

Value

A list with components:

polygons A tibble of projected polygon vertices.

boundaries A tibble of projected boundary segments.

Examples

```
## Not run:
atlas <- get_schaefer_surfatlas(100)
polygons <- build_surface_polygon_data(atlas, views = "lateral")
head(polygons$polygons)

## End(Not run)
```

check_templateflow	<i>Check TemplateFlow Installation Status</i>
--------------------	---

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	<i>Clear neuroatlas TemplateFlow Cache</i>
--------------------------	--

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)
```

clear_transform_cache	<i>Clear Verified Transform Artifacts</i>
-----------------------	---

Description

Removes only transform artifact files owned by neuroatlas. The cache root is retained, and a cache containing an active artifact lock is refused.

Usage

```
clear_transform_cache(  
  artifact_version = NULL,  
  cache_dir = transform_cache_path()  
)
```

Arguments

- artifact_version Optional immutable artifact version to remove.
- cache_dir Transform cache directory, normally [transform_cache_path()].

Value

The number of artifact files removed, invisibly.

Examples

```
cache <- tempfile("neuroatlas-example-cache-")  
dir.create(cache)  
clear_transform_cache(cache_dir = cache)  
unlink(cache, recursive = TRUE)
```

cluster_explorer	<i>Launch Cluster Explorer Shiny App</i>
------------------	--

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

<code>data_source</code>	A sample-wise data source supporting <code>neuroim2::series(data_source, i)</code> where <code>i</code> is voxel coordinates ($n \times 3$ matrix) or indices. Rows of the returned matrix correspond to samples/design rows.
<code>atlas</code>	A volumetric atlas object used for parcel annotation. When atlas and <code>stat_map</code> dimensions differ, the atlas is automatically resampled to <code>stat_map</code> space (nearest-neighbor labels) before cluster annotation.
<code>stat_map</code>	A NeuroVol statistic image used for thresholding and clustering.
<code>surfatlas</code>	A surface atlas object used by <code>plot_brain()</code> . 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`.

Examples

```
## Not run:
app <- cluster_explorer()
# Start the interactive viewer with shiny::runApp(app).

## End(Not run)
```

coord_spaces	<i>Standard Coordinate Space Identifiers</i>
--------------	--

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** Broad MNI152 coordinate family, without template identity
 - SCANNER** Native scanner space (subject-specific)
 - UNKNOWN** Unknown or unspecified space

Examples

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

coordinate_spaces	<i>Coordinate Space Transforms for Neuroimaging Templates</i>
-------------------	---

Description

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

Details

These legacy helpers classify broad coordinate families and apply a fixed FreeSurfer affine. They do not establish template-specific registration or cortical correspondence. Use exact MNI template identifiers and qualified transforms for image resampling or population cortical projection.

Value

This documentation topic describes legacy coordinate-family constants and affine helpers. Use exact template identifiers for qualified registration.

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 A broad coordinate-family label. MNI152NLin6Asym and MNI152NLin2009cAsym are distinct templates; this shorthand does not identify either variant or a registration between them.

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

Examples

```
transform_coords(matrix(c(0, 0, 0), nrow = 1),  
  from = "MNI305", to = "MNI152"  
)
```

create_templateflow	<i>Create a TemplateFlow Interface Object</i>
---------------------	---

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	<i>Dilate Atlas Parcellation Boundaries</i>
--------------	---

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 that lie within the mask
- For each unassigned voxel, finds assigned voxels within radius (Euclidean distance, in voxel units)
- Assigns the voxel to a parcel by inverse-distance-weighted voting over those neighbours (up to maxn of them)
- Leaves a voxel unassigned when no parcel lies within radius, so distant in-mask voxels (e.g. cerebellar grey matter for a cortical atlas) are not absorbed
- 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	<i>Filter Atlas by ROI Attributes</i>
--------------	---------------------------------------

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

<code>x</code>	An atlas object
<code>...</code>	Filter expressions using column names from <code>roi_metadata(x)</code> . Multiple conditions are combined with AND.
<code>.dots</code>	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. Surface-atlas subsetting is not implemented; use `get_roi()` to extract labelled surface regions.

Value

A new volume 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 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(parcel = "200", network = "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(parcel = "100", networks = "7")
schaefer <- 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.

Examples

```
## Not run:
atlas <- get_brainnetome_atlas()
atlas_ref(atlas)

## End(Not run)
```

get_cifti_transform *Bind Qualified Cortical Operators to a CIFTI Layout*

Description

Creates a mixed-layout adapter with explicit hemisphere operators. The file cannot prove a registration identity: supplying an operator declares that its exact source domain matches the CIFTI vertex ordering. Counts are checked but never used to infer that binding. Noncortical models must retain the same indices and volume geometry; their rows may be reordered. Changed voxel grids or support require a separate volumetric operation and are rejected here.

Usage

```
get_cifti_transform(from, to, cortex, volume_space = NULL)
```

Arguments

from	Source CiftiData, used for its brain-model layout.
to	Target CiftiData, used for its brain-model layout only. Source map names, metadata and label tables are retained during application.
cortex	Named list of qualified SurfaceTransform operators, L and/or R, covering every cortical hemisphere present. Obtain these with get_template_transform() and explicitly verified surface geometries.
volume_space	Caller-declared exact volume template shared by source and target, required when voxel models are present. Supply one common identifier, or a named c(from = ..., to = ...) pair that must agree. This first adapter accepts the qualified MNI6/MNI2009c frames only. Affines cannot establish template identity. Declaring a frame does not establish anatomical accuracy.

Value

A CiftiTransform binding source/target layouts and cortical operators.

See Also

[apply_cifti_transform\(\)](#), [read_cifti\(\)](#)

Examples

```
## Not run:
op <- get_cifti_transform(source, reference, list(L = left, R = right),
  volume_space = "MNI152Nlin6Asym"
)

## End(Not run)
```

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/atlas/’; 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'.

Examples

```
## Not run:
# Requires an FSL installation configured through FSLDIR.
atlas <- get_fsl_atlas("harvard_oxford_cortical", resolution = 2)
atlas_ref(atlas)

## End(Not run)
```

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	<i>Load Glasser Atlas</i>
-------------------	---------------------------

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.

Parcel IDs are representation-specific: volume IDs 1–180 refer to the right hemisphere ("hcp_R_first"), whereas [glasser_surf\(\)](#) uses left-first IDs ("surfatlas_L_first"). For cross-representation tables, join on `roi_metadata(atlas)$label_full` ("L_V1_ROI", "R_V1_ROI", etc.), or on `c("area", "hemi")`. Both loaders expose these shared keys. Original source labels remain in `orig_labels`. ID-keyed tables must declare a matching `id_convention` column; see [align_parcel_values\(\)](#).

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.
...	Arguments passed by the convenience wrappers to 'get_harvard_oxford_atlas()'.

Value

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

Examples

```

## Not run:
atlas <- get_harvard_oxford_atlas("cortical", threshold = 25)
atlas_ref(atlas)

## End(Not run)

```

get_hcpex_atlas	<i>Load the Extended HCP Atlas (HCPex)</i>
-----------------	--

Description

Load HCPex v1.1, a volumetric atlas with 360 cortical and 66 subcortical regions in the source-declared MNI152NLin2009cAsym template space.

Usage

```
get_hcpex_atlas(resolution = c("1", "2"), outspace = NULL, use_cache = TRUE)
```

Arguments

resolution	Native voxel size in millimetres: 1 (default) or 2.
outspace	Optional three-dimensional NeuroSpace. Resampling uses nearest neighbours and changes the sampling grid only; it does not register the atlas to a different anatomical template. Small regions may disappear.
use_cache	Logical. Reuse checksum-verified files in the neuroatlas user cache. If FALSE, download temporary files and remove them after loading.

Details

Files are downloaded on demand from a pinned revision of the upstream v1.1 distribution. Both native resolutions retain the upstream voxel geometry and HCPex IDs (1:426, with zero as background). HCPex cortical ordering differs from the original HCP-MMP1.0 ordering; matrices in that ordering must be reordered before use with this atlas.

Labels combine upstream abbreviations with hemisphere suffixes (e.g., V1_L, V1_R). `orig_labels` preserves the full lookup-table names. `roi_metadata()` also includes region (the unsuffixed abbreviation) and division ("cortical" for IDs 1:360, "subcortical" for 361:426). The complete region catalogue is retained after resampling, even when some regions no longer have voxels. No surface representation is supplied.

The upstream data are distributed under GPL-3.0; their license, source revision, file receipts, and publication references are attached to the atlas.

Value

An object with classes `hcpex` and `atlas`, containing a `NeuroVol`, region IDs, labels, hemisphere assignments, RGB colors, and resource metadata.

Source

<https://github.com/wayalan/HCPex>, including the v1.1 user guide.

References

Huang CC, Rolls ET, Feng J, Lin CP (2022). An extended Human Connectome Project multimodal parcellation atlas of the human cortex and subcortical areas. *Brain Structure and Function*, 227, 763–778. doi:10.1007/s00429021024216

Huang CC, Rolls ET, Hsu CH, Feng J, Lin CP (2021). Extensive Cortical Connectivity of the Human Hippocampal Memory System: Beyond the "What" and "Where" Dual Stream Model. *Cerebral Cortex*. doi:10.1093/cercor/bhab113

See Also

`get_atlas()`, `get_glasser_atlas()`, `atlas_citations()`

Examples

```
## Not run:
hcp <- get_hcpex_atlas(resolution = 2)
# Also available through get_atlas("hcpex", resolution = 2)
get_roi(hcp, label = "V1_L")
subcortex <- filter_atlas(hcp, division == "subcortical")
atlas_metadata(hcp)
atlas_citations(hcp)

## End(Not run)
```

get_hipp_atlas	<i>Extract Hippocampal Parcellation</i>
----------------	---

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/atlasses’.

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’.
download	Download the Julich-Brain atlas archive into the neuroatlas cache when ‘fsl_dir’ is unset.
...	Additional arguments passed to ‘get_fsl_atlas()’.

Value

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

Examples

```
## Not run:
atlas <- get_julich_brain_atlas()
atlas_ref(atlas)

## End(Not run)
```

get_olsen_mtl	<i>Load Olsen MTL Atlas</i>
---------------	-----------------------------

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	<i>Extract a region of interest (ROI) from an atlas</i>
---------	---

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_roi.surfatlas	<i>Extract regions of interest from a surface atlas</i>
-------------------	---

Description

get_roi() method for surface atlases (class surfatlas, e.g. from [get_wang_atlas](#), [glasser_surf](#), or [schaefer_surf](#)). It returns one [ROISurface](#) per matched region, carrying the mesh vertices of that area on the appropriate hemisphere.

Because a surface atlas stores each hemisphere on its own mesh, a label present in both hemispheres yields one ROI per hemisphere, named "<label>_left" / "<label>_right"; pass hemi to restrict. When selecting by id, results are named by id (ids are unique across hemispheres: 1..K left, K+1..2K right).

Usage

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

Arguments

x	A surface atlas (class surfatlas).
label, id	Character labels or integer ids identifying regions; supply exactly one.
hemi	Optional "left" / "right" filter.

Value

A named list of [ROI](#)[Surface](#) objects.

See Also

[get_roi](#)

Examples

```
## Not run:
atlas <- get_schaefer_surfatlas(100)
roi <- get_roi(atlas, id = atlas$ids[1])

## End(Not run)
```

get_schaefer_atlas	<i>Load Schaefer Brain Parcellation Atlas</i>
--------------------	---

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,
  cache_dir = NULL
)

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

```
    ...  
)  
  
sy_100_17(  
    resolution = "2",  
    outspace = NULL,  
    smooth = FALSE,  
    use_cache = TRUE,  
    cache_dir = NULL,  
    ...  
)  
  
sy_200_7(  
    resolution = "2",  
    outspace = NULL,  
    smooth = FALSE,  
    use_cache = TRUE,  
    cache_dir = NULL,  
    ...  
)  
  
sy_200_17(  
    resolution = "2",  
    outspace = NULL,  
    smooth = FALSE,  
    use_cache = TRUE,  
    cache_dir = NULL,  
    ...  
)  
  
sy_300_7(  
    resolution = "2",  
    outspace = NULL,  
    smooth = FALSE,  
    use_cache = TRUE,  
    cache_dir = NULL,  
    ...  
)  
  
sy_300_17(  
    resolution = "2",  
    outspace = NULL,  
    smooth = FALSE,  
    use_cache = TRUE,  
    cache_dir = NULL,  
    ...  
)
```

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

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

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

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

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

```
sy_600_17(  
  resolution = "2",  
  outspace = NULL,
```

```
    smooth = FALSE,  
    use_cache = TRUE,  
    cache_dir = NULL,  
    ...  
)
```

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

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

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

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

```
sy_900_7(  
  resolution = "2",  
  outspace = NULL,  
  smooth = FALSE,  
  use_cache = TRUE,  
  cache_dir = NULL,
```

```

    ...
)

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

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

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

```

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
cache_dir	Optional directory for Schaefer volume and label assets. When 'NULL', uses the 'neuroatlas.schaefer.cache_dir' option, then the 'neuroatlas.cache_dir' option, and finally the platform cache directory. Ignored when 'use_cache = FALSE', which uses transient files only.
...	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
cmmap 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`, `cache_dir`, 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

<code>parcels</code>	Character string specifying number of parcels. Options: "100", "200", "300", "400", "500", "600", "800", "1000"
<code>networks</code>	Character string specifying network count. Options: "7", "17"
<code>surf</code>	Character string specifying surface type. Options: "inflated", "white", "pial"
<code>use_cache</code>	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	<i>Get Transform Matrix Between Coordinate Spaces</i>
---------------------	---

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_surface_geometry` *Fetch Exact Pinned Surface Registration Geometry*

Description

Downloads checksum-locked registration spheres, cortical masks and vertex areas from their original providers. Pinned fsaverage 164k, 41k (fsaverage6), 10k (fsaverage5), and fsLR 32k domains are supported. The fsLR sphere is in fsaverage correspondence. Inputs retain their upstream licenses, independently of neuroatlas's license; see 'extdata/surface-assets-LICENSES.md' in the installed package.

Usage

```
get_surface_geometry(
  template,
  density,
  hemisphere,
  cache_dir = transform_cache_path(),
  download = TRUE,
  offline = FALSE
)
```

Arguments

template	Exact family, "fsaverage" or "fsLR".
density	"164k", "41k", or "10k" for fsaverage; "32k" for fsLR.
hemisphere	"L" or "R".
cache_dir	Dedicated transform cache directory.
download	Allow downloads from pinned upstream URLs.
offline	Use only checksum-verified local files.

Value

A 'SurfaceGeometry' with exact domain identity, pinned input hashes, cortical mask, and upstream file provenance.

Examples

```
## Not run:
left <- get_surface_geometry("fsaverage", "164k", "L")
target <- get_surface_geometry("fsLR", "32k", "L")
transform <- get_template_transform(left, target)
x <- surface_data(rep(0.25, left$domain$n_vertices), left$domain)
y <- apply_template_transform(x, transform)

## End(Not run)
```

get_surface_projection

Resolve a Qualified Population Volume-to-Cortex Projection

Description

Uses pinned CBIG RF-ANTs population registration-fusion sampling coordinates on the exact fsaverage 164k domain. MNI2009c uses the qualified MNI6-to-2009c image pullback at those coordinates, without an intermediate resampled image. fsLR 32k output additionally uses the admitted native surface operator. Population correspondence does not substitute for subject registration. Numerical qualification covers the locked 1 mm and 2 mm source grids and the four pinned target

domains. Other grids in the declared frame can be sampled, but have no retained grid-specific qualification. The original CBIG reference and MNI6 images agree on sampled cortical support, but differ elsewhere.

Usage

```
get_surface_projection(
  from,
  to,
  cache_dir = transform_cache_path(),
  download = TRUE,
  offline = FALSE
)
```

Arguments

from	Exact "MNI152NLin6Asym" or "MNI152NLin2009cAsym" identifier. Generic "MNI152" is insufficient.
to	Verified pinned 'SurfaceGeometry' for one hemisphere.
cache_dir	Dedicated transform cache directory.
download	Allow downloads of checksum-locked upstream inputs.
offline	Require verified cached inputs and operators.

Value

A directed 'SurfaceProjection', with resident sampling coordinates, exact domains, input hashes, engine identity and composition provenance.

Examples

```
## Not run:
target <- get_surface_geometry("fsaverage", "164k", "L")
projection <- get_surface_projection("MNI152NLin6Asym", target)
volume <- get_template("MNI152NLin6Asym", resolution = 2)
cortex <- apply_surface_projection(volume, projection,
  source_space = "MNI152NLin6Asym", data_type = "continuous"
)

## End(Not run)
```

get_surface_transform *Build or Load a Directed Surface Transform*

Description

Constructs ordinary native closest-point barycentric weights between explicit registered spheres. Source mask exclusion precedes row normalization; target masking follows it. No registration is fitted. Native geometric correctness and exact Workbench parity are distinct qualification claims. Arbitrary supplied geometries are always marked "unqualified".

Usage

```
get_surface_transform(
  from,
  to,
  cache_dir = transform_cache_path(),
  offline = FALSE
)
```

Arguments

from, to	Verified ‘SurfaceGeometry’ objects in the same declared sphere correspondence frame and hemisphere.
cache_dir	Transform cache directory, or ‘NULL’ to disable caching.
offline	Require an existing verified operator cache entry.

Value

A ‘SurfaceTransform’ with sparse weights, exact domains, engine fingerprint, integrity digest and directed provenance. Reverse resampling requires calling this function with swapped geometries; it is not an inverse.

See Also

[surface_geometry()], [surface_data()], [apply_surface_transform()]

Examples

```
sphere <- rbind(
  c(1, 0, 0), c(-1, 0, 0), c(0, 1, 0),
  c(0, -1, 0), c(0, 0, 1), c(0, 0, -1)
)
faces <- rbind(
  c(0, 2, 4), c(2, 1, 4), c(1, 3, 4), c(3, 0, 4),
  c(2, 0, 5), c(1, 2, 5), c(3, 1, 5), c(0, 3, 5)
)
cortex <- rep(TRUE, 6)
domain <- surface_domain(
  "toy", "L", "6v", sphere, faces, cortex,
  "analytic", "v1"
)
if (requireNamespace("neurotransform", quietly = TRUE) &&
  utils::packageVersion("neurotransform") >= "0.2.0") {
  geometry <- surface_geometry(domain, sphere, faces, cortex)
  operator <- get_surface_transform(geometry, geometry, cache_dir = NULL)
  values <- surface_data(seq(0, 0.5, length.out = 6), domain)
  apply_surface_transform(values, operator)
}
```

get_template	<i>Fetch a Template from TemplateFlow</i>
--------------	---

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', a 'SurfaceGeometry' with attached [template_metadata()].

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_template_transform

Resolve and Load a Verified Template Transform

Description

Resolves an available image or exact-domain surface route and loads its verified artifacts. Two SurfaceGeometry endpoints select an admitted native surface operator. An exact MNI identifier and a pinned surface target select a directed population cortical projection. Equal domains require a verified diagonal operator with unit weights before admission as an identity. Planning and fitting are separate: this function never estimates a registration. Nonlinear routes must have a published checksum and a passing qualification record in [space_transform_manifest\(\)](#).

Usage

```
get_template_transform(
  from,
  to,
  provider = c("auto", "neuroatlas", "templateflow"),
  download = TRUE,
  verify = TRUE,
  cache_dir = transform_cache_path(),
  offline = FALSE,
  cortex = NULL,
  volume_space = NULL
)
```

Arguments

from, to	Exact template identifiers, verified SurfaceGeometry objects or CiftiData layouts with explicitly supplied operators and volume frame. Surface routes require exact geometries; broad names cannot establish vertex ordering or admit execution.
provider	Artifact provider, or "auto" for registry selection.
download	Allow missing artifacts to be downloaded.
verify	Must be TRUE; artifact integrity cannot be disabled.
cache_dir	Dedicated transform cache directory.
offline	Use only verified local artifacts.
cortex	For two CiftiData layouts, explicitly bound qualified cortical operators passed to get_cifti_transform() . Other representations require NULL.
volume_space	For CIFTI layouts with voxels, the caller-declared common exact frame, passed to get_cifti_transform() . Other representations require NULL.

Value

A `template_transform` containing the plan, files, pullback morphism, and artifact provenance. Its direction describes image movement; its morphism maps target coordinates into source coordinates for sampling. Surface routes return the operator from [get_surface_transform\(\)](#) or the SurfaceProjection from [get_surface_projection\(\)](#). CIFTI layouts return a CiftiTransform from [get_cifti_transform\(\)](#), preserving noncortical geometry and support.

See Also

[apply_template_transform\(\)](#), [transform_atlas\(\)](#)

Examples

```
if (requireNamespace("neurotransform", quietly = TRUE) &&
    requireNamespace("hdf5r", quietly = TRUE)) {
  get_template_transform("MNI152NLin6Asym", "MNI152NLin6Asym")
}
```

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, Brain-Voyager, 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](https://doi.org/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	<i>Early Visual Cortex Atlas (V1-V5, cytoarchitectonic)</i>
------------------	---

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 (V01/2, PHC1/2, L01/2, T01/2, V3A/B, IPS0-5, SPL1, FEF).

Usage

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

Arguments

surf	Surface type. One of "midthickness" (default), "pial", or "white". (TemplateFlow does not provide an inflated fsaverage surface.)
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_mplb1.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 volumetric counterparts (maximum-probability and per-area probability maps in MNI space) are served by `get_wang_prob_atlas`.

TemplateFlow does not currently distribute an *inflated* fsaverage mesh, so the available surfaces are "midthickness" (default), "pial", and "white".

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 midthickness surface
wang <- get_wang_atlas(surf = "midthickness")
wang$labels
# Extract an area as ROI Surface objects (one per hemisphere)
get_roi(wang, label = "hV4")
get_roi(wang, label = "V1v", hemi = "left")

## End(Not run)
```

<code>get_wang_prob_atlas</code>	<i>Wang (2015) Full Per-Area Probability Volumes (Princeton ProbAtlas_v4)</i>
----------------------------------	---

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

<code>prob_dir</code>	Optional path to a locally-extracted ProbAtlas_v4 directory (the folder containing <code>subj_vol_all</code> , or that subfolder itself). When supplied it is used exclusively; otherwise the cached/downloaded data is used.
<code>image</code>	One of "probability" (per-area perc_VTPM maps, the default) or "maxprob" (the maximum-probability summary volume).
<code>hemi</code>	One of "both" (default), "lh", or "rh".
<code>rois</code>	Optional subset of areas, given as labels (e.g. <code>c("V1v", "hV4")</code>) or integer ids (1-25). NULL selects all 25.
<code>path_only</code>	Logical. When TRUE (default), return a manifest of paths/metadata without downloading or reading image data. When FALSE, read the resolved volumes as <code>NeuroVol</code> objects, downloading them to the cache on first use if needed.
<code>use_cache</code>	Logical. Look for (and download into) the neuroatlas Wang cache directory when <code>prob_dir</code> is not supplied. FALSE forces a fresh download on load.
<code>x</code>	A <code>wang_prob_paths</code> object.
<code>...</code>	Unused.

Details

On first load the volumes (~0.7 MB) are downloaded from the neuroatlas GitHub release and cached under the neuroatlas Wang cache directory, after which they resolve offline. They originate from the Princeton ProbAtlas_v4 distribution (Wang et al. 2015), whose original host (`napl.scholar.princeton.edu`) blocks scripted downloads and is no longer reliably available; the archive carried no licence and the maps are widely redistributed as an open-science resource, so they are re-hosted on the package's own release for programmatic access.

To use a local copy instead (e.g. an updated release), download/unzip ProbAtlas_v4 and pass the resulting directory (or its `subj_vol_all` subfolder) via `prob_dir`; it takes precedence and is never mixed with the cache or download. Set `path_only = FALSE` to read the requested volumes as `NeuroVol` objects.

The resolution order on a load is `prob_dir` -> cache -> download. A read-only manifest (`path_only = TRUE`) resolves from `prob_dir`/cache only and never downloads or writes to the cache. When

path_only = FALSE and use_cache = TRUE, volumes read from a user-supplied prob_dir are also copied (best-effort, atomically) into the cache.

Note on naming: the volume coding (ROIfiles_Labeling.txt) labels areas 12/13 as MST/hMT, which correspond to T02/T01 in the surface (**neuropythy**) naming used by `get_wang_atlas()`; the numeric ids are identical.

Value

When path_only = TRUE, an object of class wang_prob_paths: a list with the requested image/hemi, the resources_url and download_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): what is available and where it comes from
manifest <- get_wang_prob_atlas()
manifest
head(manifest$files)

# Load the maximum-probability volumes (downloaded + cached on first use)
wp <- get_wang_prob_atlas(image = "maxprob", path_only = FALSE)

# Load a single area's probability map
v1v_lh <- get_wang_prob_atlas(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

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

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", "midthickness"),
  use_cache = TRUE
)
```

Arguments

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

Details

This function uses:

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

Annotation downloads are checked against the file sizes and MD5 checksums published by Figshare before they enter or leave the neuroatlas cache. Currently only the "fsaverage" surface space is supported. TemplateFlow provides "pial", "white", and "midthickness" geometry at the required 164k density; it does not provide an "inflated" fsaverage surface.

Surface IDs 1–180 are left hemisphere and 181–360 are right hemisphere ("surfatlas_L_first"), opposite to [get_glasser_atlas\(\)](#). Use the shared label_full or c("area", "hemi") keys from [roi_metadata\(\)](#) when moving parcel values between representations. ID-keyed tables must declare a matching id_convention column; see [align_parcel_values\(\)](#).

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.

Examples

```
infer_design_var_type(c(0.1, 0.2, 0.3))  
infer_design_var_type(c("control", "patient"))
```

```
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

<code>method</code>	Ignored.
<code>conda</code>	Ignored.
<code>envname</code>	Ignored.
<code>force_reinstall</code>	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.

Examples

```
## Not run:  
launch_cluster_explorer()  
  
## End(Not run)
```

list_atlases	<i>List Registered Atlases</i>
--------------	--------------------------------

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, hemisphere information, and other per-region metadata such as Schaefer network assignments
- 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

network Combined network assignments, when either parent has them (e.g. Schaefer); regions from a parent without networks are NA

Other per-region attributes reported by [roi_metadata](#) for either parent are carried over in the same way, so `roi_metadata()` and [query_point](#) on the merged atlas report them for the regions that had them.

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`.

It includes rotation, scaling and translation components. Anatomical error depends on position and the source/target registrations; it is not a uniform displacement or a qualified volume-to-cortex mapping.

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	<i>MNI305 to MNI152 Affine Transform Matrix</i>
------------------	---

Description

A legacy 4x4 FreeSurfer affine for approximate coordinate conversion from MNI305 to the broad MNI152 family. It does not encode an exact MNI variant.

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.register.dat`.

It includes rotation, scaling and translation components. Anatomical error depends on position and the source/target registrations; it is not a uniform displacement or a qualified volume-to-cortex mapping.

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	<i>Check if a Nonlinear Template Warp is Needed</i>
---------------------	---

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	<i>Check if Transform is Needed Between Spaces</i>
-----------------	--

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	<i>Assign harmonic anchor hues to networks</i>
---------------------	--

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	<i>Atlas Reference Metadata</i>
---------------	---------------------------------

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("uncertain", "exact", "high", "approximate"),
  notes = NA_character_,
  id_convention = 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.
id_convention	Optional parcel ID convention. Glasser uses "hcp_R_first" for volumes and "surfatlas_L_first" for surfaces.

Value

An object of class "atlas_ref".

Examples

```
ref <- new_atlas_ref("toy", "two-regions",
  template_space = "MNI152NLin6Asym", coord_space = "MNI152"
)
ref
```

olsen_mtl	<i>Olsen Medial Temporal Lobe Atlas</i>
-----------	---

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

Bundled 'data/olsen_mtl.rda', attributed to Rosanna Olsen and colleagues. The original atlas publication and redistribution terms have not been verified. Use [get_olsen_mtl()] to attach the available provenance record.

Examples

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

# View available regions
olsen_mtl$labels

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

parcel_data	<i>Parcel-Level Data Container</i>
-------------	------------------------------------

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 meta-data ('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’.

Examples

```
grid <- neuroim2::NeuroSpace(c(2, 1, 1))
volume <- neuroim2::NeuroVol(array(c(1, 2), c(2, 1, 1)), grid)
atlas <- structure(
  list(
    atlas = volume, ids = 1:2,
    labels = c("A", "B"), orig_labels = c("A", "B"),
    hemi = c("left", "right"), name = "toy",
    cmap = rbind(c(1, 0, 0), c(0, 0, 1)),
    atlas_ref = new_atlas_ref("toy", "two-regions",
      template_space = "MNI152NLin6Asym", coord_space = "MNI152"
    )
  ),
  class = "atlas"
)
```

```
x <- parcel_data(data.frame(
  id = 1:2, label = c("A", "B"),
  hemi = c("left", "right"), value = c(0.2, 0.7)
), atlas_id = "toy")
parcel_values(x, atlas)
```

parcel_volume

Expand Parcel Values to a Volumetric Atlas

Description

Align one parcel-level metric to a volumetric atlas and expand it into a dense `neuroim2::NeuroVol` in the atlas's native grid and spatial reference.

Usage

```
parcel_volume(
  atlas,
  data,
  value,
  by = NULL,
  allow_partial = FALSE,
  background = NA_real_
)
```

Arguments

atlas	A volumetric atlas object. Surface atlases are not supported.
data	A data frame, tibble, or <code>parcel_data</code> object containing parcel keys and values.
value	A numeric value column, supplied as a bare name or string.
by	Parcel-key specification passed to align_parcel_values() .
allow_partial	Logical. If FALSE (default), data must contain every atlas parcel. If TRUE, missing parcels receive NA. Unknown and duplicate parcel keys always error.
background	Numeric scalar used outside the atlas. The default is <code>NA_real_</code> , which keeps background distinct from valid parcel values such as zero.

Details

`parcel_volume()` uses the same strict key matching as [align_parcel_values\(\)](#). The input table remains the authoritative result; the returned volume is a deterministic rendering representation. Atlas voxels whose non-zero labels are absent from `atlas$ids` cause an error rather than being silently treated as background.

Value

A dense `neuroim2::NeuroVol` containing the selected parcel value at every voxel belonging to that parcel. Atlas space and geometry are preserved.

Examples

```
## Not run:
atlas <- get_aseg_atlas()
results <- data.frame(
  id = atlas$ids,
  z_stat = stats::rnorm(length(atlas$ids))
)
z_map <- parcel_volume(atlas, results, z_stat)

## End(Not run)
```

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,
  legend = FALSE,
  legend_max = 30L,
  ...
)

## 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).
legend	Logical. If TRUE, draw a region colour legend (label -> colour) below the "montage" view. Default FALSE. A legend is only useful for small atlases, so it is suppressed (with a warning) when the atlas has more than legend_max regions; it is also omitted for view = "ortho" (each plane shows a different subset of regions). Labels occurring in both hemispheres are disambiguated with (L)/(R).
legend_max	Integer. Maximum number of regions for which a legend is drawn when legend = TRUE. Default 30.
...	Additional arguments passed to <code>neuroim2::plot_montage()</code> or <code>neuroim2::plot_ortho()</code> .
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: <code>c("lateral", "medial")</code> .

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,
  static_backend = c("ggplot", "cpu"),
  data_id_mode = c("parcel", "polygon"),
  ncol = 2L,
  panel_layout = c("native", "presentation"),
  style = c("default", "ggseg_like", "stat_publication"),
  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,
outer_contour = FALSE,
outer_contour_color = "grey35",
outer_contour_size = 0.3,
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_alpha_mode = c("constant", "threshold"),
overlay_alpha_ramp = NULL,
overlay_palette = "vik",
overlay_lim = NULL,
overlay_border = FALSE,
overlay_border_color = "black",
overlay_border_size = 0.25,
overlay_fun = c("avg", "nn", "mode"),
overlay_sampling = c("midpoint", "normal_line", "thickness"),
overlay_interpolation = c("legacy", "nearest", "linear"),
overlay_aggregate = NULL,
overlay_n_samples = NULL,
overlay_depth = NULL,
overlay_surface_smooth_fwhm = 0,
colorbar = FALSE,
colorbar_source = c("auto", "base", "overlay", "none"),
colorbar_title = NULL,
overlay_title = colorbar_title,
title = NULL,
subtitle = NULL,
caption = NULL,
panel_labels = NULL,
cortex_mask = NULL,
cortex_mask_source = NULL,
anatomy_metric = NULL,
anatomy_metric_source = NULL,
medial_wall = c("shade", "mask", "outline"),
camera = c("canonical", "presentation"),
orientation_labels = TRUE,
render_width = 1200L,
render_height = 750L,
render_antialias = 2L,

```

```

outline = FALSE,
background = FALSE,
background_color = "grey80",
depth_cull = TRUE,
bg = "white",
data = NULL,
value = NULL,
by = NULL,
allow_partial = FALSE,
anatomy_style = "publication",
anatomy_midpoint = NULL,
anatomy_invert = FALSE,
anatomy_range = c(0.72, 0.9),
vals_threshold = NULL,
parcel_style = NULL,
...
)

```

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", or "midthickness". When omitted, it is read from surfatlas\$surf_type; an explicitly requested type must match the geometry carried by 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.
static_backend	Static renderer: the existing "ggplot" polygon path or deterministic "cpu" rasterization. With a vertex overlay the CPU path renders continuous publication overlays; with parcel vals (and no overlay) it renders a publication parcel map through [neurosurf::render_surface_parcels()]: smooth antialiased parcel edges,

	a sulcal-depth underlay, soft lighting, a light medial wall, and an automatically arranged figure. Neither needs OpenGL or a browser.
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. Placement, scale, and panel limits use the full cortex silhouette, regardless of parcel coverage or whether background is drawn.
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. "stat_publication" treats the surface as an anatomical substrate for a continuous overlay: parcel and culling-derived silhouette lines are disabled, weak shading is drawn below the overlay, and a clean outer contour is enabled.
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.
outer_contour	Logical. If TRUE, draw the largest projected exterior loop in each panel. Unlike silhouette, this excludes internal visibility and sulcal edge fragments.

outer_contour_color	Colour for the exterior contour.
outer_contour_size	Line width for the exterior contour.
network_border	Logical. If TRUE, highlight boundaries between different networks (requires <code>surfatlas\$network</code>). Default: FALSE.
network_border_color	Colour for network boundary lines. Default: <code>border_color</code> .
network_border_size	Line width for network boundary lines. Default: <code>border_size * 2</code> .
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 <code>NeuroVol</code> . If a <code>NeuroVol</code> , it is projected onto the surface using <code>neurosurf::vol_to_surf()</code> . A raw <code>NeuroVol</code> does not carry a template-space identifier that this function can enforce, so callers must ensure it is already expressed in the coordinates of the resolved white and pial surfaces. No cross-coordinate transform is applied here. 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_alpha_mode	Character. "constant" uses <code>overlay_alpha</code> for all rendered overlay faces. "threshold" fades overlay faces from transparent at <code>overlay_threshold</code> to <code>overlay_alpha</code> , avoiding hard threshold edges for dense surface maps. Default: "constant".
overlay_alpha_ramp	Optional positive numeric scalar controlling the absolute-value distance above <code>overlay_threshold</code> over which <code>overlay_alpha_mode = "threshold"</code> reaches full opacity. If NULL, a small data-driven ramp is chosen from the rendered overlay values. Use 0 to disable ramping while keeping the threshold mode.
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: FALSE.
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 <code>neurosurf::vol_to_surf()</code> when overlay is a <code>NeuroVol</code> . One of "avg", "nn", or "mode". Default: "avg".
overlay_sampling	Character: sampling strategy passed to <code>neurosurf::vol_to_surf()</code> when overlay is a <code>NeuroVol</code> . One of "midpoint", "normal_line", or "thickness". Default: "midpoint".
overlay_interpolation	Voxel interpolation passed to <code>neurosurf::vol_to_surf()</code> : "legacy", "nearest", or "linear". The publication preset defaults to linear interpolation.
overlay_aggregate	Optional explicit aggregation across depth samples: "mean", "mode", or "closest".
overlay_n_samples	Optional number of sampling depths.
overlay_depth	Optional explicit thickness fractions or normal-line offsets. The publication preset uses five fractions from 0.1 through 0.9.
overlay_surface_smooth_fwhm	Tangential surface smoothing in mm. Defaults to zero; it is separate from voxel interpolation.
colorbar	Logical or character. When <code>interactive = FALSE</code> , 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_source	Which mapped quantity supplies the static colorbar: "auto", "base", "overlay", or "none". "auto" chooses the overlay whenever one was supplied, even if all of its values are removed by thresholding; otherwise it chooses vals.
colorbar_title	Optional character label for the colorbar.
overlay_title	Optional character label used when the colorbar source is the overlay. Defaults to colorbar_title.
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.
cortex_mask	Optional logical vertex-domain mask or lh/rh list.
cortex_mask_source	Provenance label for an explicit cortex mask.
anatomy_metric	Optional anatomy metric or lh/rh list. A declared sulcal metric is preferred; otherwise the CPU backend computes curvature on matched white geometry and verifies vertex correspondence.

anatomy_metric_source	Provenance label for an explicit metric.
medial_wall	Explicit medial-wall policy: neutral shade, mask, or independent outline.
camera	Camera for <code>static_backend = "cpu"</code> : strict canonical orthographic or slightly oblique presentation. The <code>ggplot</code> backend always uses canonical projections; use <code>panel_layout</code> to arrange its panels.
orientation_labels	Draw small anterior/posterior marks in CPU panels.
render_width, render_height	Per-panel CPU raster dimensions.
render_antialias	CPU supersampling factor.
outline	Logical. If TRUE, draw every triangle edge (mesh wireframe). Default: FALSE. Typically border is preferred.
background	Logical. If TRUE, draw the full cortical surface beneath the parcellation, so sparse atlases (e.g. the Wang visual areas, which label only part of cortex) are shown in anatomical context rather than floating on the page. The backdrop is given sulcal/gyral depth via normal-based shading (controlled by <code>shading_strength/shading_gamma</code>), so it reads as a folded surface rather than a flat silhouette. Default: FALSE.
background_color	Fill colour for the cortex backdrop when <code>background = TRUE</code> . Default: "grey80".
depth_cull	Logical. If TRUE (default), remove faces hidden behind nearer cortical surface faces in each projected view. This makes static medial/lateral panels read as opaque cortex instead of showing far-side folds through the mesh.
bg	Character: background colour for the plot. Default: "white".
data	Optional data frame, tibble, or <code>parcel_data</code> object with one row per parcel. When supplied, value is aligned to the atlas with <code>align_parcel_values()</code> before rendering. Supply either data or vals, not both.
value	Numeric column in data, supplied as a bare name or character string.
by	Parcel-key specification for data. Use a shared column name such as "id", or map an atlas key to a differently named data column with <code>c(id = "roi_index")</code> . Composite keys are supported. When NULL, a safe unique key is inferred.
allow_partial	Logical. If FALSE (default), data must contain every atlas parcel. If TRUE, missing parcels are rendered with NA values. Unknown and duplicate keys always error.
anatomy_style, anatomy_midpoint, anatomy_invert, anatomy_range	CPU underlay controls passed to <code>[neurosurf::render_surface_rgba()]</code> .
vals_threshold	Optional non-negative magnitude for the CPU parcel renderer (<code>static_backend = "cpu"</code> with vals): parcels with <code>abs(vals)</code> below it stay unfilled, and the colorbar greys out the sub-threshold band.
parcel_style	Optional <code>[neurosurf::surface_parcel_style()]</code> object or named list of its settings for the CPU parcel renderer.
...	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)

results <- data.frame(
  roi_index = rev(atl$ids),
  estimate = seq(-2, 2, length.out = length(atl$ids))
)
plot_brain(
  atl,
  data = results,
  value = estimate,
  by = c(id = "roi_index"),
  views = "medial",
  hemis = "left",
  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)
```

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 = NULL,
  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,
  data = NULL,
  values = NULL,
  by = NULL,
  allow_partial = FALSE,
  ...
)
```

Arguments

<code>surfatlas</code>	A <code>surfatlas</code> object (e.g., from <code>schaefer_surf</code> or <code>glasser_surf</code>).
<code>vals_list</code>	A list of numeric vectors, one per panel. Each vector must have length equal to the number of atlas regions. Optional when <code>data</code> and <code>values</code> are supplied. Names, when present, become the default panel titles.
<code>views</code>	Character vector of views passed to <code>plot_brain</code> . Default: <code>c("lateral", "medial")</code> .
<code>hemis</code>	Character vector of hemispheres passed to <code>plot_brain</code> . Default: <code>c("left", "right")</code> .
<code>ncol</code>	Integer number of columns in the grid layout. If <code>NULL</code> , chosen automatically.
<code>shared_scale</code>	Logical. If <code>TRUE</code> (default), all panels share the same colour scale computed from the range of all values.
<code>palette</code>	Character: <code>scico</code> palette name. Default: <code>"cork"</code> .
<code>lim</code>	Optional numeric length-2 colour limits. Overrides automatic limits when provided.
<code>titles</code>	Optional character vector of panel titles. If <code>NULL</code> , names of <code>vals_list</code> 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.
data	Optional data frame, tibble, or parcel_data object with one row per parcel. Supply either data or vals_list.
values	Character vector naming numeric columns in data; one brain map is produced for each column.
by	Parcel-key specification passed to align_parcel_values() .
allow_partial	Logical. If FALSE (default), data must contain every atlas parcel. If TRUE, missing parcels receive NA values.
...	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)

results <- data.frame(
  id = rev(atl$ids),
  Contrast_A = rnorm(200),
  Contrast_B = rnorm(200)
)
plot_brain_grid(
  atl,
  data = results,
  values = c("Contrast_A", "Contrast_B")
)

## 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	<i>Print Method for Atlas References</i>
-----------------	--

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’.

Examples

```
ref <- new_atlas_ref("toy", "two-regions",
  template_space = "MNI152NLin6Asym", coord_space = "MNI152"
)
print(ref)
```

print.atlas_transform_plan	<i>Print Method for Transform Plans</i>
----------------------------	---

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	<i>Print a TemplateFlow Object</i>
--------------------	------------------------------------

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.

Examples

```
## Not run:
interface <- create_templateflow()
print(interface)

## End(Not run)
```

project_cluster_overlay

Project a Volume onto a Surface Atlas

Description

Samples a volumetric map (for example a thresholded statistic or cluster map) onto the vertices of both hemispheres of a surface atlas, using the same volume-to-surface projection that [plot_brain](#) applies when its `overlay` argument is a `NeuroVol`. The result holds one value per atlas vertex, so it can be passed back to `plot_brain(overlay = )`, summarised, or used to build a colour key that matches the rendered surface exactly.

Usage

```
project_cluster_overlay(
  cluster_vol,
  surfatlas,
  space_override = NULL,
  density_override = NULL,
  resolution_override = NULL,
  fun = c("avg", "nn", "mode"),
  sampling = c("midpoint", "normal_line", "thickness"),
  interpolation = c("legacy", "nearest", "linear"),
  aggregate = NULL,
  n_samples = NULL,
  depth = NULL,
  surface_smooth_fwhm = 0
)
```

Arguments

<code>cluster_vol</code>	A NeuroVol in the world (mm) space of the surface meshes, typically MNI152 for fsaverage surfaces. Zero voxels are treated as data, so mask the volume first if zero should mean "no signal".
<code>surfatlas</code>	A surface atlas (class "surfatlas"), for example from schaefer_surf . Its vertex count per hemisphere sets the length of the returned vectors.

space_override	Optional surface space (e.g. "fsaverage6"), TemplateFlow density, and resolution used to look up the white and pial meshes. By default these come from the atlas (surfatlas\$surface_space, falling back to "fsaverage6").
fun	Vertex summary passed to neurosurf::vol_to_surf(): one of "avg", "nn", or "mode".
sampling	Sampling strategy between the white and pial surfaces: "midpoint", "normal_line", or "thickness".
interpolation	Voxel interpolation: "legacy", "nearest", or "linear".
aggregate	Optional aggregation across depth samples ("mean", "mode", or "closest").
n_samples	Optional number of sampling depths.
depth	Optional explicit thickness fractions or normal-line offsets.
surface_smooth_fwhm	Tangential surface smoothing in mm (default 0, no smoothing).

Details

White and pial meshes are taken from the atlas when it was built on them and otherwise loaded for the atlas's surface space, so the projection is anatomically correct even for atlases displayed on inflated surfaces.

If projection fails for a hemisphere (for example because the meshes cannot be loaded or do not match the atlas), that hemisphere is returned as all NA and a warning reports the reason; plot_brain() keeps its existing silent behaviour.

Value

A list with two elements:

overlay A list with numeric vectors lh and rh, one value per vertex of the corresponding atlas hemisphere. Vertices the projection does not reach are NA. A hemisphere missing from the atlas is NULL.

meta Projection provenance: surface_space and, per hemisphere, the vertex counts and the sampling settings that were applied (the same record plot_brain() stores).

See Also

[plot_brain](#), neurosurf::vol_to_surf()

Examples

```
## Not run:
atlas <- schaefer_surf(200, 7, space = "fsaverage6", surf = "inflated")
stat <- neuroim2::read_vol("zstat1.nii.gz")
proj <- project_cluster_overlay(stat, atlas,
  sampling = "thickness",
  interpolation = "linear"
)
range(proj$overlay$lh, na.rm = TRUE)
```

```
# Draw exactly the projected values
plot_brain(atlas, overlay = proj$overlay, overlay_threshold = 3.1)

## End(Not run)
```

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.

Examples

```
project_surface_view(rbind(c(0, 0, 0), c(1, 2, 3)),
  view = "lateral", hemi = "left"
)
```

projection_diagnostics

Inspect Coverage and Label Loss in a Cortical Projection

Description

Returns diagnostics computed by `apply_surface_projection()`. Label counts are separated by map and stage, so a key surviving another map cannot hide loss. `not_sampled` means a source key has no supported sampled vertex; `lost_resampling` means it survives sampling but disappears during surface resampling. Declared keys absent from the source have status `absent_source`. Opposite-hemisphere keys have status `other_hemisphere` and are excluded from expected parcel-loss totals; any supported output vertices carrying those keys are still counted as hemisphere mismatches. Without explicit hemisphere metadata, keys remain unchecked and no hemisphere is inferred. Key zero is counted separately and excluded from parcel-loss totals; supported zero is still a valid value. Diagnostics never mask, relabel or repair values.

Usage

```
projection_diagnostics(x)
```

Arguments

`x` A `SurfaceData` result from `apply_surface_projection()`.

Value

A list with summary, a per-map coverage/count tibble, and labels, a per-map/per-key tibble for label data (NULL for other data types). Label rows contain source voxel counts, supported sampled and target vertex counts, hemisphere declarations, loss flags and stage status. Vertex and voxel counts are not interchangeable area or volume measures. Missing and masked vertices are excluded from label counts and retained in coverage.

Examples

```
## Not run:
qa <- projection_diagnostics(projected_atlas)
qa$summary
subset(qa$labels,
  lost_at_sampling | lost_at_resampling | hemisphere_mismatch
)

## End(Not run)
```

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’, ‘from_space’, or ‘nearest’.
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.

Examples

```
grid <- neuroim2::NeuroSpace(c(2, 1, 1))
volume <- neuroim2::NeuroVol(array(c(1, 2), c(2, 1, 1)), grid)
atlas <- structure(
  list(
    atlas = volume, ids = 1:2,
    labels = c("A", "B"), orig_labels = c("A", "B"),
    hemi = c("left", "right"), name = "toy",
    cmap = rbind(c(1, 0, 0), c(0, 0, 1)),
    atlas_ref = new_atlas_ref("toy", "two-regions",
      template_space = "MNI152NLin6Asym", coord_space = "MNI152"
    )
  ),
  class = "atlas"
)
query_coord(atlas, matrix(c(1, 0, 0), nrow = 1), radius = 0)
```

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'), fuzzy search ('radius > 0') that returns all regions within a sphere of given radius in millimetres, and a nearest-label mode ('nearest = TRUE') that returns the single closest labelled region within 'radius' together with its distance. Multiple atlases can be queried simultaneously.

Usage

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

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").
nearest	Logical. If 'TRUE', return exactly one row per point and atlas: the region containing the point if there is one, otherwise the nearest labelled voxel within 'radius' mm, with its distance in a 'distance' column. Default 'FALSE' keeps the exact/fuzzy behaviour.

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.

With 'nearest = TRUE', the voxel containing the point (the voxel whose grid position is nearest to it) is checked first; if it is labelled, that region is returned with distance '0'. Otherwise every labelled voxel whose centre lies within 'radius' mm of the point is considered and the closest one wins. Distances are Euclidean, in the atlas's world coordinates. Ties (voxel centres equidistant from the point, to within 1e-6 mm) are broken deterministically in favour of the **smallest region id**, so the result does not depend on voxel scan order. With 'radius = 0' only the containing voxel is considered.

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](#).

An atlas whose 'coord_space' is 'NA', empty or '"Unknown"' has no single coordinate space to transform into. This is the case for [merge_atlases](#) composites whose parents declare different spaces, and for atlases built without a space annotation. Such atlases are queried directly in their own world coordinates: the input coordinates are assumed to be in the atlas's space, no transform is applied, and a warning of class '"neuroatlas_unknown_coord_space"' says so. Check that the input coordinates match the atlas grid before relying on the result.

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).

distance Only when 'nearest = TRUE': distance in mm from the query coordinate to the centre of the reported region's nearest voxel; '0' when the point lies in a labelled voxel, 'NA' when no labelled voxel is within 'radius'.

Any further per-region columns from [roi_metadata](#) follow.

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)

# Nearest labelled region within 4 mm, with its distance
query_point(c(24, -10, 5), atlas, radius = 4, nearest = TRUE)

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

## End(Not run)
```

read_cifti	<i>Read CIFTI-2 Dense Scalar or Label Maps</i>
------------	--

Description

Preserves both cortical and volumetric brain models, their zero-based indices, map metadata and per-map label tables. Matrix axes are normalized in memory only: brainordinates are rows and maps are columns. The original file layout is retained for writing. CIFTI metadata does not identify a registration sphere or an exact MNI template; vertex counts alone cannot bind a surface domain.

Usage

```
read_cifti(file)
```

Arguments

file	Path to a CIFTI-2 dense scalar or dense label file.
------	---

Value

A CiftiData S3 list with values, ordered brain_models, volume, map_names, label_tables, original XML/header/extensions and file identity. available distinguishes missing or explicitly unassigned samples from supported zero. Vertex counts do not prove a surface registration identity. Scalar missing values are retained. Infinite values and label values absent from their map's label table are rejected. Series/connectivity/parcellated mappings and CIFTI-1 are not supported.

See Also

[write_cifti\(\)](#), [replace_cifti_values\(\)](#)

Examples

```
# Read an existing file without guessing a surface registration:
## Not run:
read_cifti("atlas.dlabel.nii")

## End(Not run)
```

read_parcel_data	<i>Read a 'parcel_data' Object from Disk</i>
------------------	--

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.

Examples

```
x <- parcel_data(data.frame(
  id = 1:2, label = c("A", "B"),
  hemi = c("left", "right"), value = c(0.2, 0.7)
), atlas_id = "toy")
file <- tempfile(fileext = ".rds")
write_parcel_data(x, file)
read_parcel_data(file)
unlink(file)
```

reduce_atlas	<i>Reduce a NeuroVol or NeuroVec by an Atlas</i>
--------------	--

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

<code>atlas</code>	An atlas object or another object for which a method is defined.
<code>data_vol</code>	A <code>NeuroVol</code> (3D) or <code>NeuroVec</code> (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.
<code>stat_func</code>	The function to apply to the data within each ROI (e.g., <code>mean</code> , <code>sd</code> , <code>sum</code>).
<code>...</code>	Additional arguments passed to <code>stat_func</code> .
<code>format</code>	Character string specifying output format: "wide" or "long". If <code>NULL</code> (default), uses "long" for <code>NeuroVol</code> and "wide" for <code>NeuroVec</code> .
<code>level</code>	Optional grouping level used by methods.
<code>by</code>	Optional grouping variable used by methods.

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(parcel = "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(parcel = "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)
```

replace_cifti_values	<i>Replace Values Without Changing CIFTI Brain Models</i>
----------------------	---

Description

Keeps brain-model rows, map metadata and label tables bound to new values. Changing indices or map counts requires a new explicitly constructed layout. Previously unavailable samples remain unavailable, even when their replacement values are finite or use an existing label key.

Usage

```
replace_cifti_values(x, values)
```

Arguments

x	A CiftiData from <code>read_cifti()</code> .
values	Numeric matrix with the same dimensions as x\$values.

Value

A new CiftiData, with the input identity recorded in provenance.

See Also

`read_cifti()`, `write_cifti()`

Examples

```
## Not run:
x <- read_cifti("map.dscalar.nii")
x <- replace_cifti_values(x, x$values * 2)

## End(Not run)
```

resample	<i>Resample Volume to New Space</i>
----------	-------------------------------------

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

<code>vol</code>	A NeuroVol object to be resampled
<code>outspace</code>	A NeuroSpace object specifying the target space
<code>smooth</code>	Logical. Whether to apply boundary smoothing after resampling. Default: FALSE
<code>interp</code>	Integer. Interpolation method (0=nearest neighbor, 1=linear). Default: 0
<code>radius</code>	Numeric. Radius for smoothing neighborhood in voxels. If NULL, uses $\max(\text{spacing})+0.5$. Default: NULL
<code>min_neighbors</code>	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)
```

ribbon_projection	<i>Declare Aligned White/Pial Ribbon Sampling</i>
-------------------	---

Description

Declares ordered white and pial coordinates in the same physical frame as the source volume. Alignment and vertex ordering are assertions by the caller; this function does not fit or verify a registration. Inflated display surfaces must not be supplied. Equal-weight nodes include both end-points. This method differs from Workbench voxel-intersection ribbon mapping.

Usage

```
ribbon_projection(domain, white, pial, cortex, frame, n_samples = 5L)
```

Arguments

domain	Exact surface domain for the ordered anatomical coordinates.
white, pial	Numeric vertex-by-three matrices in RAS millimetres.
cortex	Logical inclusion vector matching the domain's cortical mask.
frame	Exact shared physical-frame identifier.
n_samples	Integer node count, at least two. Nodes are equally spaced along each white-to-pial segment; categorical nodes vote with smallest-key ties.

Value

A 'SurfaceProjection' with coordinate hashes and caller-declared alignment provenance. Numerical sampling qualification does not establish anatomical registration accuracy.

Examples

```
## Not run:
projection <- ribbon_projection(domain, white, pial, cortex,
  frame = "subject-01-scanner-RAS", n_samples = 5
)

## End(Not run)
```

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". PCA component signs are canonicalized from the first non-negligible loading so the resulting hues are stable across LAPACK implementations.
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

<code>rois</code>	Tibble with one ROI per row containing 'roi', 'x', 'y', and 'z'.
<code>id_col</code>	Column containing ROI IDs.
<code>xyz_cols</code>	Character vector of length three giving the coordinates to use for distance calculations.
<code>hemi_col</code>	Optional hemisphere column used for the final light/dark adjustment.
<code>network_col</code>	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

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

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	<i>ROI Metadata Functions</i>
--------------	-------------------------------

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 Full region label. Glasser uses a shared hemisphere-qualified key such as "L_V1_ROI" in both representations.

area Glasser area name without hemisphere, e.g. "V1".

id_convention Parcel ID convention, when recorded by the loader.

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	<i>Schaefer Surface Atlas</i>
---------------	-------------------------------

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'. Artifact-backed routes also record 'artifact_id', 'artifact_version', 'provider', 'url', 'sha256', 'size_bytes', 'format', 'convention', 'qualification', 'qualification_scope', 'qa_url', and 'license'. These fields are missing for routes without a downloadable artifact. 'source_representation' and 'target_representation' distinguish volumes and surfaces. 'executable' indicates support by the template-transform API, given the required optional dependencies and artifacts; it is not a local readiness check. Exact admitted surface routes also bind 'from_domain_id', 'to_domain_id', 'method', 'engine_revision', 'input_lock_sha256' and hemisphere. 'from_density' and 'to_density' record pinned surface densities; 'route_scope' distinguishes "roadmap_placeholder", "exact_surface_domains", "exact_volume_to_surface", "exact_template_pair", and legacy "coordinate_family" routes. Broad surface names remain advisory. Numerical qualification is restricted to the recorded inputs and methods; it does not establish anatomical accuracy.

Examples

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

# Executable routes, with their identity scope
reg[reg$executable, c("from_space", "to_space", "route_scope",
  "hemisphere", "from_density", "to_density", "backend")]
```

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 $(\text{sum}(\text{abs}(\text{null}) \geq \text{abs}(\text{observed})) + 1) / (n_perm + 1)$.

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 $\geq$ 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	<i>Select a Subset of Atlas Regions</i>
-----------	---

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 'surfatlas'
sub_atlas(x, ids = NULL, labels = NULL, hemi = NULL, network = NULL, ...)

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

Arguments

<code>x</code>	An atlas object.
<code>ids</code>	Integer or numeric vector of region IDs to retain (matched against <code>x\$ids</code>).
<code>labels</code>	Character vector of region labels to retain (matched against <code>x\$labels</code>).
<code>hemi</code>	Character vector of hemispheres to retain ("left" and/or "right").
<code>...</code>	Additional arguments passed to methods.
<code>network</code>	Optional network labels to retain, where supported.

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
```

surface_anatomy

Resolve an anatomical underlay for a surface atlas

Description

Uses an explicit metric, then the atlas anatomy metric, otherwise a metric computed from matching white geometry. Supplied metrics are used unchanged. The result can be shared by static and interactive renderers. No curvature is inferred from an inflated surface.

Usage

```
surface_anatomy(
  surfatlas,
  hemi = "lh",
  metric = NULL,
  source = NULL,
  type = c("curvature", "sulcal_depth")
)
```

Arguments

surfatlas	A surface atlas.
hemi	Hemisphere, lh or rh (left and right are also accepted).
metric	Optional finite per-vertex metric overriding the atlas.
source	Optional provenance label for the supplied metric.
type	Computed metric used when neither ‘metric’ nor the atlas supplies one: “curvature” (default) or “sulcal_depth”.

Details

Two computed metrics are available. "curvature" is mean curvature of the white surface with five adjacency averaging steps; it resolves individual folds but looks mottled on inflated displays. "sulcal_depth" is a FreeSurfer-style sulcal-depth proxy, the signed displacement between the white and the displayed (inflated) surface from [neurosurf::surface_sulcal_proxy()]; it gives the broad two-tone gyral/sulcal pattern used by Workbench and pycortex. When the proxy cannot be computed (for example on a white display surface) curvature is used instead and the provenance records that.

Value

A list with metric and provenance. Unavailable computed anatomy is neutral, with source recorded as neutral_fallback.

Examples

```
## Not run:
atlas <- get_schaefer_surfatlas(100)
anatomy <- surface_anatomy(atlas, hemi = "lh")

## End(Not run)
```

surface_data	<i>Bind Vertex Values to an Exact Surface Domain</i>
--------------	--

Description

Explicitly declaring the domain binds the caller's vertex ordering. GIFTI hemisphere metadata, when present, must agree. A GIFTI file alone cannot establish topology or vertex order; the caller must know its domain.

Usage

```
surface_data(
  values,
  domain,
  data_type = c("continuous", "label", "probability"),
  label_table = NULL
)
```

Arguments

- values Numeric vertex vector or vertex-by-map matrix, or a metric or label GIFTI filename. 'NA' values represent missing data; infinity is rejected.
- domain A 'SurfaceDomain' descriptor.
- data_type "continuous", "label", or "probability". Probability values must lie in [0,1]; channels are never renormalized across maps.

label_table Optional data frame with unique integer ‘key’ values and optional names/colors. Required when importing label GIFTI; read from the file when omitted. Every finite input key must appear in the table.

Value

‘SurfaceData’, preserving values, domain, label table and input identity.

Examples

```
sphere <- diag(3)
faces <- matrix(c(0L, 1L, 2L), nrow = 1)
cortex <- rep(TRUE, 3)
domain <- surface_domain(
  "toy", "L", "3v", sphere, faces, cortex,
  "analytic", "v1"
)
surface_data(c(0, 0.25, 0.5), domain)
```

surface_domain	<i>Identify an Exact Surface Vertex Domain</i>
----------------	--

Description

Constructs a descriptor from an ordered registration sphere, its triangles, and its cortical mask. Template names and vertex counts alone cannot establish correspondence. This function records identity; it does not establish that a registration is anatomically valid or qualify a resampling method. Array validation does not establish closed-manifold topology or mesh quality.

Usage

```
surface_domain(
  template,
  hemisphere,
  density,
  sphere,
  triangles,
  cortex,
  registration,
  revision,
  vertex_area = NULL,
  index_base = c("zero", "one")
)
```

Arguments

template	Exact template family identifier, such as "fsaverage" or "fsLR". Aliases are not inferred.
hemisphere	"L" or "R".
density	Density identifier, such as "164k" or "32k".
sphere	Numeric matrix of sphere coordinates, one vertex per row and three columns. Coordinates must be finite, nonzero, and approximately equidistant from the origin (relative radius tolerance 0.001).
triangles	Integer-valued matrix with three vertex indices per row.
cortex	Logical vector in vertex order; 'TRUE' includes cortical vertices, 'FALSE' excludes the medial wall. Missing values are disallowed.
registration	Exact identifier for the sphere correspondence frame. Matching strings record a declaration, not proof of correspondence.
revision	Immutable upstream sphere revision or content identifier.
vertex_area	Optional positive, finite per-vertex areas in square mm, in the same vertex order. Absence is recorded explicitly.
index_base	Triangle indexing: "zero" (GIFTI default) or "one" (R). It is never inferred from the minimum index.

Value

A 'SurfaceDomain' descriptor containing counts, metadata, SHA-256 fingerprints of ordered coordinates, topology, mask, and optional areas, and a combined 'id'. Arrays are not retained. Numeric arrays are hashed as row-major little-endian doubles, indices as zero-based 32-bit integers, and masks as 32-bit integers. Row names and R storage mode do not affect identity; reordering vertices or triangles does. This conservative identity deliberately distinguishes equivalent meshes with different encodings.

See Also

[atlas_transform_plan()]

Examples

```
sphere <- diag(3)
faces <- matrix(c(0L, 1L, 2L), nrow = 1)
cortex <- rep(TRUE, 3)
domain <- surface_domain(
  "toy", "L", "3v", sphere, faces, cortex,
  "analytic", "v1"
)
domain
```

surface_geometry	<i>Bind Registration Geometry to a Surface Domain</i>
------------------	---

Description

Checks the ordered sphere, topology and cortical mask against an exact [surface_domain()] descriptor. This does not estimate a registration.

Usage

```
surface_geometry(
  domain,
  sphere,
  triangles = NULL,
  cortex,
  index_base = c("zero", "one")
)
```

Arguments

domain	A ‘SurfaceDomain’ descriptor.
sphere	Numeric vertex-by-three matrix, or a GIFTI surface filename.
triangles	Triangle matrix; omit when reading a GIFTI surface.
cortex	Logical cortical inclusion vector, or a binary GIFTI filename.
index_base	Triangle indexing for matrices, “zero” or “one”. GIFTI triangles are always zero-based.

Value

A ‘SurfaceGeometry’ containing the verified domain and arrays.

Examples

```
sphere <- diag(3)
faces <- matrix(c(0L, 1L, 2L), nrow = 1)
cortex <- rep(TRUE, 3)
domain <- surface_domain(
  "toy", "L", "3v", sphere, faces, cortex,
  "analytic", "v1"
)
surface_geometry(domain, sphere, faces, cortex)
```

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	<i>Find TemplateFlow Files Matching Metadata Criteria</i>
-------------	---

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	<i>List Available TemplateFlow Template Spaces</i>
--------------	--

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_atlas

Transform a Volumetric Atlas to a Template or Cortical Surface

Description

Uses atlas metadata to resolve the source space, then applies a verified template transform on the requested target grid. Region identities and source receipts are retained. This changes spatial coordinates, not the atlas's parcellation scheme; use [atlas_overlap()] after alignment to compare parcels.

Usage

```
transform_atlas(
  x,
  to_space,
  target = NULL,
  resolution = NULL,
  provider = "auto",
  ...
)
```

Arguments

x	A volumetric atlas with a declared template identity.
to_space	Exact target template identifier or verified 'SurfaceGeometry'.
target	Explicit target atlas, volume or grid. If 'NULL', load the requested template through [get_template()]. A 'SurfaceGeometry' supplies an exact cortical destination, matching 'to_space' when both are supplied.

resolution	Template resolution, required when 'target' is 'NULL'.
provider	Artifact provider.
...	Download/cache options passed to [get_template_transform()].

Value

For a volume destination, a transformed atlas with updated spatial metadata and history. For a cortical destination, 'SurfaceData' with the original region IDs and names, source atlas reference, coverage and lost-label reporting. Population projection does not establish subject registration.

Examples

```
## Not run:
atlas <- get_harvard_oxford_atlas("cortical", resolution = "02")
transformed <- transform_atlas(atlas, "MNI152NLin2009cAsym", resolution = 2)

## End(Not run)
```

transform_cache_path	<i>Transform Artifact Cache Path</i>
----------------------	--------------------------------------

Description

Returns the directory reserved for verified transform artifacts. This helper does not create the directory.

Usage

```
transform_cache_path(cache_dir = tools::R_user_dir("neuroatlas", "cache"))
```

Arguments

cache_dir	Base cache directory.
-----------	-----------------------

Value

A character path to the transform cache directory.

Examples

```
transform_cache_path()
```

transform_coords	<i>Transform Coordinates Between Spaces</i>
------------------	---

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	<i>Validate an Atlas Reference</i>
--------------------	------------------------------------

Description

Validate an Atlas Reference

Usage

```
validate_atlas_ref(x)
```

Arguments

x Object to validate.

Value

Invisibly returns 'x' when valid.

Examples

```
ref <- new_atlas_ref("toy", "two-regions",
  template_space = "MNI152NLin6Asym", coord_space = "MNI152"
)
validate_atlas_ref(ref)
```

validate_parcel_data	<i>Validate a Parcel-Level Data Object</i>
----------------------	--

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.

Examples

```
x <- parcel_data(data.frame(
  id = 1:2, label = c("A", "B"),
  hemi = c("left", "right"), value = c(0.2, 0.7)
), atlas_id = "toy")
validate_parcel_data(x)
```

write_cifti	<i>Write CIFTI-2 Maps With Preserved Brain Models</i>
-------------	---

Description

Writes double-precision values using the original axis layout, XML metadata, brain models, label tables and other NIfTI extensions. Label missing values are rejected; no background label is assigned implicitly.

Usage

```
write_cifti(x, file, overwrite = FALSE)
```

Arguments

x	A valid CiftiData.
file	Destination ending in .nii (uncompressed CIFTI-2).
overwrite	Allow replacement of an existing regular file.

Value

The normalized destination path, invisibly.

See Also

[read_cifti\(\)](#), [replace_cifti_values\(\)](#)

Examples

```
## Not run:
x <- read_cifti("atlas.dlabel.nii")
write_cifti(x, "copy.dlabel.nii")

## End(Not run)
```

write_parcel_data	<i>Write a ‘parcel_data’ Object to Disk</i>
-------------------	---

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.

Examples

```
x <- parcel_data(data.frame(
  id = 1:2, label = c("A", "B"),
  hemi = c("left", "right"), value = c(0.2, 0.7)
), atlas_id = "toy")
file <- tempfile(fileext = ".rds")
write_parcel_data(x, file)
read_parcel_data(file)
unlink(file)
```

Index

* datasets

- coord_spaces, [44](#)
- fsaverage, [49](#)
- MNI152_to_MNI305, [98](#)
- MNI305_to_MNI152, [99](#)
- olsen_mtl, [104](#)
- %where%, [5](#)
- align_parcel_values, [6](#), [15](#), [56](#), [90](#), [108](#), [117](#), [120](#)
- apply_cifti_transform, [8](#)
- apply_cifti_transform(), [12](#), [53](#)
- apply_surface_projection, [9](#)
- apply_surface_projection(), [11](#), [128](#)
- apply_surface_transform, [10](#)
- apply_surface_transform(), [8](#), [11](#)
- apply_template_transform, [11](#)
- apply_template_transform(), [9](#), [82](#)
- as_igraph, [13](#)
- as_igraph.atlas_connectivity, [18](#)
- as_parcel_data, [13](#)
- atlas_alignment, [15](#)
- atlas_artifacts, [16](#), [52](#)
- atlas_artifacts.atlas
 - (atlas_provenance), [26](#)
- atlas_artifacts.default
 - (atlas_provenance), [26](#)
- atlas_citations, [17](#)
- atlas_citations(), [59](#)
- atlas_connectivity, [17](#)
- atlas_coord_space, [19](#)
- atlas_family, [19](#)
- atlas_graph, [20](#)
- atlas_hierarchy, [21](#)
- atlas_history, [22](#)
- atlas_history.atlas(atlas_provenance), [26](#)
- atlas_history.default
 - (atlas_provenance), [26](#)
- atlas_metadata, [23](#)

- atlas_overlap, [24](#)
- atlas_provenance, [26](#)
- atlas_ref, [26](#), [27](#)
- atlas_roi_colors, [28](#), [109–111](#), [113](#)
- atlas_space, [29](#)
- atlas_transform_manifest, [30](#)
- atlas_transform_plan, [30](#)
- batch_reduce, [31](#)
- brainnetome_labels, [33](#)
- build_brain_polygon_data, [34](#)
- build_cluster_explorer_data, [35](#)
- build_conflict_edges, [37](#)
- build_surface_polygon_data, [38](#)
- check_templateflow, [40](#)
- clear_templateflow_cache, [40](#)
- clear_transform_cache, [41](#)
- cluster_explorer, [41](#), [92](#), [93](#)
- ClusteredNeuroVec, [135](#), [136](#)
- ClusteredNeuroVol, [153](#)
- coord_spaces, [44](#)
- coord_to_grid, [130](#)
- coordinate_spaces, [45](#)
- cor, [152](#)
- create_templateflow, [46](#)
- dilate_atlas, [46](#), [136](#)
- filter_atlas, [5](#), [6](#), [48](#), [147](#), [154](#)
- fsaverage, [49](#)
- geom_path, [114](#)
- geom_segment, [114](#)
- get_aseg_atlas, [50](#), [98](#)
- get_atlas, [51](#)
- get_atlas(), [59](#)
- get_brainnetome_atlas, [33](#), [52](#)
- get_cifti_transform, [53](#)
- get_cifti_transform(), [8](#), [82](#)
- get_fsl_atlas, [54](#)

get_ggseg_atlas, [55](#)
 get_glasser_atlas, [56, 90](#)
 get_glasser_atlas(), [59](#)
 get_harvard_oxford_atlas, [57](#)
 get_harvard_oxford_cortical_atlas
 (get_harvard_oxford_atlas), [57](#)
 get_harvard_oxford_cortical_subcortical_atlas
 (get_harvard_oxford_atlas), [57](#)
 get_harvard_oxford_subcortical_atlas
 (get_harvard_oxford_atlas), [57](#)
 get_hcpex_atlas, [58](#)
 get_hipp_atlas, [60, 62](#)
 get_julich_brain_atlas, [61, 84, 85](#)
 get_olsen_mtl, [62](#)
 get_roi, [49, 51, 62, 64, 98, 134, 154](#)
 get_roi.surfatlas, [63](#)
 get_schaefer_atlas, [64, 71](#)
 get_schaefer_surfatlas, [70, 70](#)
 get_space_transform, [72, 75, 164](#)
 get_subcortical_atlas, [73](#)
 get_surface_coordinate_space, [74, 166](#)
 get_surface_geometry, [75](#)
 get_surface_projection, [76](#)
 get_surface_projection(), [9, 82](#)
 get_surface_template, [86, 94](#)
 get_surface_template(get_template), [79](#)
 get_surface_transform, [77](#)
 get_surface_transform(), [82](#)
 get_template, [73, 74, 79](#)
 get_template_brainmask, [47](#)
 get_template_transform, [81](#)
 get_template_transform(), [12, 53](#)
 get_visfatlas, [82, 84–86](#)
 get_visual_atlas, [83, 84, 86](#)
 get_wang_atlas, [63, 83, 84, 85, 85, 86, 88](#)
 get_wang_prob_atlas, [86, 86](#)
 ggseg_schaefer, [89, 111](#)
 glasser_surf, [56, 63, 86, 90, 113, 119](#)

 infer_design_var_type, [91](#)
 install_templateflow, [92](#)

 launch_cluster_explorer, [92](#)
 list_atlases, [93](#)
 load_surface_template, [90, 94, 152](#)

 map_atlas, [51, 95, 96, 134](#)
 map_to_schaefer, [96](#)
 merge_atlases, [97, 131](#)

 MNI152_to_MNI305, [98, 99](#)
 MNI305_to_MNI152, [72, 98, 99, 164](#)

 needs_coord_transform, [100](#)
 needs_template_warp, [101](#)
 needs_transform, [102](#)
 network_anchor_hues, [102](#)
 NeuroVol, [125](#)
 new_atlas_ref, [103](#)
 nn, [152](#)

 olsen_mtl, [104](#)

 parcel_data, [106](#)
 parcel_values, [107](#)
 parcel_volume, [108](#)
 plot-methods(plot.atlas), [109](#)
 plot.atlas, [109, 109](#)
 plot.glasser(plot.atlas), [109](#)
 plot.surfatlas(plot.atlas), [109](#)
 plot_brain, [38, 42, 43, 55, 89, 96, 109, 110, 111, 119, 120, 125–127](#)
 plot_brain_grid, [118](#)
 plot_glasser, [121](#)
 print-methods(print.atlas), [121](#)
 print.atlas, [121](#)
 print.atlas_alignment, [122](#)
 print.atlas_provenance(print.atlas),
 [121](#)
 print.atlas_ref, [123](#)
 print.atlas_transform_plan, [124](#)
 print.glasser(print.atlas), [121](#)
 print.NeuroResourceMetadata
 (atlas_metadata), [23](#)
 print.schaefer(print.atlas), [121](#)
 print.templateflow, [124](#)
 print.wang_prob_paths
 (get_wang_prob_atlas), [86](#)
 project_cluster_overlay, [125](#)
 project_surface_view, [127](#)
 projection_diagnostics, [128](#)
 projection_diagnostics(), [10](#)

 query_coord, [129](#)
 query_point, [97, 130](#)
 query_vox(query_coord), [129](#)

 read_cifti, [132](#)
 read_cifti(), [53, 137, 168](#)

- read_parcel_data, 133
- reduce_atlas, 18, 31, 32, 133, 136
- reduce_atlas_vec, 135
- replace_cifti_values, 137
- replace_cifti_values(), 132, 168
- resample, 137
- ribbon_projection, 139
- ribbon_projection(), 9
- roi_attributes, 49, 140, 147
- roi_colors_embedding, 141
- roi_colors_maximin_view, 142
- roi_colors_network_harmony, 143
- roi_colors_rule_hcl, 145
- roi_metadata, 6, 49, 90, 97, 131, 140, 146
- roi_metadata(), 59
- ROISurface, 63, 64
- schaefer_surf, 34, 113, 119, 125, 148
- schaefer_surf_options, 149
- show_templateflow_cache_path, 150
- space_transform_manifest, 150
- space_transform_manifest(), 81
- spin_test, 151
- sub_atlas, 153
- sub_clusters, 153
- subcortical_atlas_options, 73, 154
- surface_anatomy, 155
- surface_data, 156
- surface_domain, 157
- surface_geometry, 159
- sy_1000_17 (get_schaefer_atlas), 64
- sy_1000_7 (get_schaefer_atlas), 64
- sy_100_17 (get_schaefer_atlas), 64
- sy_100_7 (get_schaefer_atlas), 64
- sy_200_17 (get_schaefer_atlas), 64
- sy_200_7 (get_schaefer_atlas), 64
- sy_300_17 (get_schaefer_atlas), 64
- sy_300_7 (get_schaefer_atlas), 64
- sy_400_17 (get_schaefer_atlas), 64
- sy_400_7 (get_schaefer_atlas), 64
- sy_500_17 (get_schaefer_atlas), 64
- sy_500_7 (get_schaefer_atlas), 64
- sy_600_17 (get_schaefer_atlas), 64
- sy_600_7 (get_schaefer_atlas), 64
- sy_700_17 (get_schaefer_atlas), 64
- sy_700_7 (get_schaefer_atlas), 64
- sy_800_17 (get_schaefer_atlas), 64
- sy_800_7 (get_schaefer_atlas), 64
- sy_900_17 (get_schaefer_atlas), 64
- sy_900_7 (get_schaefer_atlas), 64
- template_citations (atlas_citations), 17
- template_metadata (atlas_metadata), 23
- template_to_coord_space, 160
- tflow_files, 161
- tflow_spaces, 161
- tibble, 28, 131
- transform_atlas, 162
- transform_atlas(), 82
- transform_cache_path, 163
- transform_coords, 72, 75, 98, 99, 130, 164, 166
- transform_vertices_to_volume, 165
- validate_atlas_ref, 166
- validate_parcel_data, 167
- validate_resource_metadata (atlas_metadata), 23
- write_cifti, 167
- write_cifti(), 9, 132, 137
- write_parcel_data, 168