Package: ggseg.formats (via r-universe)

November 16, 2026

Title Brain Atlas Data Structures for the 'ggsegverse' Ecosystem

Version 0.0.4.9027

Description Provides the 'ggseg_atlas' S3 class used across the 'ggsegverse' ecosystem for 2D and 3D brain visualisation. Ships four bundled atlases ('Desikan-Killiany', 'FreeSurfer' 'aseg', 'TRACULA', 'SUIT') and functions for querying, subsetting, renaming, and enriching atlas objects. Also includes readers for 'FreeSurfer' statistics files.

License MIT + file LICENSE

Depends R (>= 4.1.0)

Imports cli, lifecycle, rlang, sfheaders

Suggests covr, devtools, here, knitr, rmarkdown, sf, spelling, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

Config/Needs/website ggsegverse/ggseg.docs

Config/Needs/lint dplyr, ggplot2, hexSticker

Config/Needs/gp dplyr, ggplot2, hexSticker

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)

URL https://github.com/ggsegverse/ggseg.formats, https://ggsegverse.github.io/ggseg.formats/

BugReports https://github.com/ggsegverse/ggseg.formats/issues

Config/roxygen2/version 8.1.1

LazyData true

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

Date/Publication 2026-11-16 19:16:19 UTC

RemoteUrl https://github.com/ggsegverse/ggseg.formats

RemoteRef HEAD

RemoteSha f0725b095712ab9d94b3b7727e7368b719a13500

Contents

as_ggseg_atlas . . . 3
as_polygon_atlas . . . 4
as_sf_atlas . . . 5
aseg . . . 5
assert_ggseg_atlas . . . 7
atlas_centerlines . . . 8
atlas_context . . . 9
atlas_core_add . . . 10
atlas_display . . . 10
atlas_geom . . . 11
atlas_geometry_type . . . 12
atlas_hemi . . . 13
atlas_labels . . . 14
atlas_manipulation . . . 15
atlas_meshes . . . 15
atlas_palette . . . 16
atlas_plot_palette . . . 17
atlas_polygons . . . 18
atlas_provenance . . . 18
atlas_region_semantics . . . 19
atlas_regions . . . 20
atlas_relabel . . . 21
atlas_sf . . . 22
atlas_structure_reorder . . . 23
atlas_type . . . 24
atlas_undrawn . . . 25
atlas_vertices . . . 26
atlas_views . . . 26
atlas_views_empty . . . 27
attach_sfc . . . 28
convert_legacy_brain_atlas . . . 29
core_contract . . . 30
dk . . . 31
drawing_manipulation . . . 32
get_brain_mesh . . . 36
get_cerebellar_mesh . . . 37
ggseg_atlas . . . 37
ggseg_data_cerebellar . . . 38
ggseg_data_cortical . . . 40
ggseg_data_subcortical . . . 41
ggseg_data_tract . . . 42
ggseg_data_wm . . . 43

is_ggseg_atlas 44
is_ggseg3d_atlas 45
legacy_region_map 45
mesh_space 46
migrate_atlas_display 48
migrate_atlas_files 49
migrate_atlas_hemi 50
read_atlas_files 51
read_freesurfer_stats 51
read_freesurfer_table 52
region_manipulation 53
reorder_sfc 55
selectors 56
set_atlas_palette 57
set_atlas_provenance 58
set_atlas_type 58
set_mesh_space 59
suit 60
tracula 61
view_manipulation 63

Index **65**

`as_ggseg_atlas` *Coerce to ggseg atlas*

Description

Coerce to ggseg atlas

Usage

```
as_ggseg_atlas(x)

as_brain_atlas(x)
```

Arguments

`x`	object to make into a ggseg_atlas

Value

an object of class 'ggseg_atlas'

Examples

```
atlas <- as_ggseg_atlas(dk())
is_ggseg_atlas(atlas)
```

`as_polygon_atlas` *Convert a ggseg atlas to the sf-optional polygon representation*

Description

Sets the single `geom` slot to the `brain_polygons` representation, converting from sf if needed. The result renders identically via the `geom_polygon`-based path in ggseg, but no longer depends on the sf class machinery in `$data` — useful for wasm builds and air-gapped installs.

Usage

```
as_polygon_atlas(atlas)
```

Arguments

`atlas`	A `ggseg_atlas` (or legacy `brain_atlas`) object.

Details

A single representation is kept (no redundant sf alongside polygons). To rehydrate sf for geometric operations later, use `as_sf_atlas()`. Extra geometry columns are carried over; only columns named `x`, `y`, `group` or `subgroup` cannot be, since the nested coordinate table reserves those names, and they are dropped with a warning.

This doubles as the backward-compatible path for sf-optional installs: a lite-only ggseg that meets a still-sf-backed atlas converts it on the fly when sf is installed. When sf is not installed the geometry cannot be read, so the call aborts with a pointer to `migrate_atlas_files()` — which the atlas maintainer runs once (on a machine with sf) to ship the package in the polygon format.

Value

A `ggseg_atlas` whose `$data$geom` is a `brain_polygons` object.

Examples

```
poly <- as_polygon_atlas(dk())
is_atlas_polygon(poly) # TRUE
```

as_sf_atlas *Rehydrate a ggseg atlas into sf-backed form*

Description

Inverse of `as_polygon_atlas()`. Sets the single `geom` slot to an sf-class geometry table, converting from polygons via `sfheaders::sf_multipolygon()` if needed — sfheaders is pure Rcpp with no GDAL/GEOS/PROJ dependencies, so the conversion itself does not require a full sf installation. Use this when you want to run sf operations (buffers, intersections, CRS transforms) on atlas geometry; those sf operations themselves still require sf.

Usage

```
as_sf_atlas(atlas)
```

Arguments

`atlas`	A `ggseg_atlas` (or legacy `brain_atlas`) object.

Details

A single representation is kept (no redundant polygons alongside sf). Extra geometry columns are carried over; because an sf row covers a whole label×view feature, a column that varies within one feature keeps only its first value there.

Value

A `ggseg_atlas` whose `$data$geom` is an sf object.

Examples

```
atlas <- as_sf_atlas(as_polygon_atlas(dk()))
sf::st_buffer(atlas_geom(atlas)$geometry[[1]], dist = 2)
```

aseg *FreeSurfer Automatic Subcortical Segmentation Atlas*

Description

Returns the FreeSurfer automatic subcortical segmentation (aseg) atlas containing deep brain structures including the thalamus, caudate, putamen, pallidum, hippocampus, amygdala, accumbens, and ventricles.

Usage

```
aseg()
```

Details

This atlas is derived from FreeSurfer's `aseg.mgz` volumetric segmentation. It works with both ggseg (2D slice views) and ggseg3d (3D mesh visualizations) from a single object.

Value

A `ggseg_atlas` object with components:

atlas Character. Atlas name ("aseg")

type Character. Atlas type ("subcortical")

palette Named character vector of colours for each region

data A `ggseg_data_subcortical` object containing:

- **meshes** Data frame with `label` and `mesh` columns
- **sf** Simple features data frame for 2D rendering

Core columns

`core` carries one row per atlas region with the columns `hemi`, `region`, `label` and `names`, plus the atlas's own grouping column. `label` is the stable atlas identifier and the key the palette and the geometry are keyed on; `region` is a short, hemisphere-free key derived from it; `names` is the curated long-form display name, spelled out to read well in a figure legend.

`region` held the long-form name until version 0.1.0. `names` is not a record of those old values – use `legacy_region_map()` to translate them.

Structures

The atlas contains bilateral structures:

- Thalamus
- Caudate
- Putamen
- Pallidum (globus pallidus)
- Hippocampus
- Amygdala
- Accumbens (nucleus accumbens)
- Ventral diencephalon

Plus midline and ventricular structures:

- Lateral ventricles
- Third ventricle
- Fourth ventricle
- Brain stem
- Cerebellar cortex
- Cerebellar white matter

References

Fischl B, Salat DH, Busa E, et al. (2002). Whole brain segmentation: automated labeling of neuroanatomical structures in the human brain. Neuron, 33(3):341-355. doi:10.1016/S08966273(02)00569-X

See Also

`dk()` for cortical parcellation, `ggseg_atlas()` for the atlas class constructor

Other ggseg_atlases: `dk()`, `suit()`, `tracula()`

Examples

```
aseg()
plot(aseg())
atlas_regions(aseg())
```

---

assert_ggseg_atlas *Require a valid ggseg atlas, reporting why it is not*

Description

The companion to `is_ggseg_atlas()` for code that cannot continue without a valid atlas. The predicate answers a yes/no question and so has nothing to say about *why* an object failed; callers that turned that `FALSE` into an error could only report the class, which is the one thing that was usually fine. `assert_ggseg_atlas()` separates the two failure kinds:

Usage

```
assert_ggseg_atlas(x, arg = rlang::caller_arg(x), call = rlang::caller_env())
```

Arguments

`x`	an object
`arg`	argument name to report, for use inside another function
`call`	environment the error is attributed to

Details

- A wrong class is reported as a wrong class.
- A `ggseg_atlas` whose parts do not satisfy the schema aborts with the `ggseg_atlas()` constructor's own diagnostic, chained as the parent error, so the reader sees the missing column, the duplicated label or the mismatched data class instead of a message about the class tag.

Prefer this wherever a bad atlas would otherwise propagate; keep `is_ggseg_atlas()` for control flow, where signalling is wrong.

Value

x, invisibly, when it is a valid atlas. Aborts otherwise.

Examples

```
atlas <- assert_ggseg_atlas(dk())

broken <- dk()
broken$core$label <- NULL
try(assert_ggseg_atlas(broken))
```

---

atlas_centerlines *Get atlas tract centerlines*

Description

Returns the centerline of each tract, joined with core region info and palette colours. Tract atlases represent each pathway as a curve through the bundle rather than as a surface, and render it by sweeping a tube along that curve, so the centerline — not a mesh — is the geometry that defines them.

Usage

```
atlas_centerlines(atlas)
```

Arguments

atlas	a ggseg_atlas object

Value

A ggseg_centerlines data.frame with one row per tract, a points list-column of n x 3 coordinate matrices and a matching tangents list-column

See Also

Other atlas accessors: atlas_context(), atlas_display(), atlas_geom(), atlas_geometry_type(), atlas_hemi(), atlas_labels(), atlas_meshes(), atlas_palette(), atlas_plot_palette(), atlas_polygons(), atlas_provenance(), atlas_region_semantics(), atlas_regions(), atlas_sf(), atlas_type(), atlas_undrawn(), atlas_vertices(), atlas_views(), atlas_views_empty(), legacy_region_map()

Examples

```
centerlines <- atlas_centerlines(tracula())
nrow(centerlines)
dim(centerlines$points[[1]])
```

`atlas_context` *The labels an atlas draws but does not treat as regions*

Description

Anatomical backdrop – a cortical silhouette behind subcortical structures, an unlabelled medial wall – is geometry without a `core` row. It is drawn in grey, it is not in the legend, and it is not something user data joins to.

Usage

```
atlas_context(atlas)
```

Arguments

`atlas`	A `ggseg_atlas`.

Details

Returns the labels rather than leaving every caller to compute the set difference or, worse, match on a name. Build scripts had been writing `labels = "^cortex"`, which silently matched nothing for as long as a pipeline bug left the backdrop undrawn.

Value

Character vector of labels, empty when the atlas draws no backdrop.

See Also

Other atlas accessors: `atlas_centerlines()`, `atlas_display()`, `atlas_geom()`, `atlas_geometry_type()`, `atlas_hemi()`, `atlas_labels()`, `atlas_meshes()`, `atlas_palette()`, `atlas_plot_palette()`, `atlas_polygons()`, `atlas_provenance()`, `atlas_region_semantics()`, `atlas_regions()`, `atlas_sf()`, `atlas_type()`, `atlas_undrawn()`, `atlas_vertices()`, `atlas_views()`, `atlas_views_empty()`, `legacy_region_map()`

Examples

```
atlas_context(aseg())
```

atlas_core_add *Join additional metadata columns to an atlas's core*

Description

Adds columns to `core`, never rows: `data` must hold unique values of `by`. Use it to carry a classification — a lobe, a network, a study's own grouping — alongside the atlas's own metadata.

Usage

```
atlas_core_add(atlas, data, by = "region")
```

Arguments

`atlas`	A `ggseg_atlas` object
`data`	A data.frame with metadata to join.
`by`	Column(s) to join by. Default `"region"`.

Value

Modified `ggseg_atlas` object

See Also

atlas_manipulation for the other groups

Other atlas manipulations: `atlas_manipulation`, `atlas_relabel()`, `atlas_structure_reorder()`, `drawing_manipulation`, `region_manipulation`, `view_manipulation`

Examples

```
lobes <- data.frame(region = atlas_regions(dk())[1:2], network = c("a", "b"))
atlas_core_add(dk(), lobes)
```

atlas_display *Extract the display-name column of an atlas*

Description

The `display` column of `core` holds the long-form, human-readable name of each region, as against the short key in `region` and the atlas identifier in `label`. For the atlases bundled here it is also the `region` value shipped before the 0.1.0 re-keying, which is what makes that re-keying recoverable; see `legacy_region_map()`.

Usage

```
atlas_display(x)
```

Arguments

x	brain atlas

Value

The `display` column of `core`, unchanged: one element per `core` row, in `core` row order, with repeats and `NA`s retained. It is therefore row-aligned with `atlas_regions()` and `atlas_labels()`. Use `unique()` for the distinct set. A zero-length character vector when the atlas carries no `display` column.

See Also

`atlas_regions()`, `atlas_labels()`, `legacy_region_map()`

Other atlas accessors: `atlas_centerlines()`, `atlas_context()`, `atlas_geom()`, `atlas_geometry_type()`, `atlas_hemi()`, `atlas_labels()`, `atlas_meshes()`, `atlas_palette()`, `atlas_plot_palette()`, `atlas_polygons()`, `atlas_provenance()`, `atlas_region_semantics()`, `atlas_regions()`, `atlas_sf()`, `atlas_type()`, `atlas_undrawn()`, `atlas_vertices()`, `atlas_views()`, `atlas_views_empty()`, `legacy_region_map()`

Examples

```
atlas_display(dk())
atlas_display(aseg())
```

---

atlas_geom *Get the raw 2D geometry of an atlas*

Description

Returns the single 2D geometry object stored in `atlas$data$geom`, which is either an sf-class data frame or a `brain_polygons` data.frame. Its class determines which rendering path is used downstream.

Usage

```
atlas_geom(atlas)
```

Arguments

atlas	a ggseg_atlas object

Details

For backward compatibility with released atlases built before the unified `geom` slot, this falls back to the legacy `sf` slot. Reverse dependencies should call this accessor (or `atlas_sf()` / `atlas_polygons()`) rather than reaching into `atlas$data` directly.

Value

an sf or `brain_polygons` object, or `NULL` if the atlas has no 2D geometry

See Also

Other atlas accessors: `atlas_centerlines()`, `atlas_context()`, `atlas_display()`, `atlas_geometry_type()`, `atlas_hemi()`, `atlas_labels()`, `atlas_meshes()`, `atlas_palette()`, `atlas_plot_palette()`, `atlas_polygons()`, `atlas_provenance()`, `atlas_region_semantics()`, `atlas_regions()`, `atlas_sf()`, `atlas_type()`, `atlas_undrawn()`, `atlas_vertices()`, `atlas_views()`, `atlas_views_empty()`, `legacy_region_map()`

Examples

```
g <- atlas_geom(dk())
atlas_geometry_type(dk())
```

---

atlas_geometry_type *Classify or test an atlas's 2D geometry*

Description

Classify or test an atlas's 2D geometry

Usage

```
atlas_geometry_type(atlas)

is_atlas_sf(atlas)

is_atlas_polygon(atlas)
```

Arguments

`atlas`	a ggseg_atlas object

Value

`atlas_geometry_type()` returns `"sf"` or `"polygon"`, and errors if the atlas has no recognised 2D geometry. `is_atlas_sf()` / `is_atlas_polygon()` return a logical scalar (`FALSE` for non-atlases).

See Also

Other atlas accessors: atlas_centerlines(), atlas_context(), atlas_display(), atlas_geom(), atlas_hemi(), atlas_labels(), atlas_meshes(), atlas_palette(), atlas_plot_palette(), atlas_polygons(), atlas_provenance(), atlas_region_semantics(), atlas_regions(), atlas_sf(), atlas_type(), atlas_undrawn(), atlas_vertices(), atlas_views(), atlas_views_empty(), legacy_region_map()

Examples

```
atlas_geometry_type(dk())
is_atlas_polygon(dk())
```

---

atlas_hemi *Extract the hemisphere column of an atlas*

Description

hemi records which side of the brain a row sits on, as one of "left", "right" or "midline", or NA where a side does not apply. It is a closed vocabulary, enforced at construction, because it is what consumers filter and facet on.

Usage

```
atlas_hemi(x)
```

Arguments

x	brain atlas

Value

The hemi column of core, unchanged: one element per core row, in core row order, with repeats and NAs retained. A zero-length character vector when the atlas carries no hemi column.

See Also

atlas_regions(), atlas_labels(), atlas_display()

Other atlas accessors: atlas_centerlines(), atlas_context(), atlas_display(), atlas_geom(), atlas_geometry_type(), atlas_labels(), atlas_meshes(), atlas_palette(), atlas_plot_palette(), atlas_polygons(), atlas_provenance(), atlas_region_semantics(), atlas_regions(), atlas_sf(), atlas_type(), atlas_undrawn(), atlas_vertices(), atlas_views(), atlas_views_empty(), legacy_region_map()

Examples

```
unique(atlas_hemi(dk()))
unique(atlas_hemi(suit()))
```

`atlas_labels` *Extract the label column of an atlas*

Description

Extract the label column of an atlas

Usage

```
atlas_labels(x)

brain_labels(x)
```

Arguments

x	brain atlas

Value

The `label` column of `core`, unchanged: one element per `core` row, in `core` row order, with repeats and `NA`s retained. It is therefore row-aligned with `atlas_regions()` and `atlas_display()`. Use `unique()` for the distinct set. A zero-length character vector when the atlas carries no `label` column.

See Also

Other atlas accessors: `atlas_centerlines()`, `atlas_context()`, `atlas_display()`, `atlas_geom()`, `atlas_geometry_type()`, `atlas_hemi()`, `atlas_meshes()`, `atlas_palette()`, `atlas_plot_palette()`, `atlas_polygons()`, `atlas_provenance()`, `atlas_region_semantics()`, `atlas_regions()`, `atlas_sf()`, `atlas_type()`, `atlas_undrawn()`, `atlas_vertices()`, `atlas_views()`, `atlas_views_empty()`, `legacy_region_map()`

Examples

```
atlas_labels(dk())
atlas_labels(aseg())
```

atlas_manipulation *Manipulate brain atlas regions, views and drawings*

Description

Four groups of pipe-friendly verbs, each acting on a different thing. Which group a verb belongs to is the first thing to know about it, because two of the groups read alike and do not behave alike.

Details

- region_manipulation: which regions the atlas has.
- view_manipulation: which views exist, and where they sit.
- drawing_manipulation: which regions are drawn inside a view. These add no views and remove nothing from `core`: the region stays, its depiction in some view goes — though a view left drawing nothing goes with it.
- `atlas_core_add()`: metadata columns on `core`.

All of them return a new `ggseg_atlas`.

See Also

region_manipulation, view_manipulation, drawing_manipulation, `atlas_core_add()`

Other atlas manipulations: `atlas_core_add()`, `atlas_relabel()`, `atlas_structure_reorder()`, `drawing_manipulation`, `region_manipulation`, `view_manipulation`

atlas_meshes *Get atlas meshes for 3D rendering*

Description

Returns meshes data joined with core region info and palette colours. Used for subcortical and tract atlases.

Usage

```
atlas_meshes(atlas)
```

Arguments

`atlas`	a ggseg_atlas object

Value

data.frame with meshes ready for 3D rendering

See Also

Other atlas accessors: atlas_centerlines(), atlas_context(), atlas_display(), atlas_geom(), atlas_geometry_type(), atlas_hemi(), atlas_labels(), atlas_palette(), atlas_plot_palette(), atlas_polygons(), atlas_provenance(), atlas_region_semantics(), atlas_regions(), atlas_sf(), atlas_type(), atlas_undrawn(), atlas_vertices(), atlas_views(), atlas_views_empty(), legacy_region_map()

Examples

```
meshes <- atlas_meshes(aseg())
head(meshes)
```

---

atlas_palette *Get the palette of an atlas*

Description

Retrieves the colour palette from a brain atlas: a named character vector mapping region labels to colours. To change the palette, use the set_atlas_palette() setter.

Usage

```
atlas_palette(atlas, ...)
```

Arguments

atlas	a ggseg_atlas object
...	Additional arguments (unused)

Value

Named character vector of colours.

See Also

set_atlas_palette() to set the palette.

Other atlas accessors: atlas_centerlines(), atlas_context(), atlas_display(), atlas_geom(), atlas_geometry_type(), atlas_hemi(), atlas_labels(), atlas_meshes(), atlas_plot_palette(), atlas_polygons(), atlas_provenance(), atlas_region_semantics(), atlas_regions(), atlas_sf(), atlas_type(), atlas_undrawn(), atlas_vertices(), atlas_views(), atlas_views_empty(), legacy_region_map()

Examples

```
atlas_palette(aseg())
atlas_palette(dk())
```

`atlas_plot_palette` *Get a plottable palette for an atlas*

Description

Returns the atlas `palette` when it can tell the atlas's regions apart, and otherwise a palette of automatically assigned, distinguishable colours. Use this instead of reading `atlas$palette` directly whenever the palette is about to be drawn: some atlases are built from colour lookup tables that give every region the same colour (often black), and rendering those faithfully produces a solid silhouette in which no region can be told from its neighbours.

Usage

```
atlas_plot_palette(atlas)
```

Arguments

`atlas`	a `ggseg_atlas` object

Details

A palette is treated as unusable when every one of the atlas's regions resolves to a single colour. Contextual geometry – the `cortex_` silhouette and the `unknown` medial wall – is excluded from that judgement and keeps its conventional grey in the fallback, so backdrop is never mistaken for a parcel. Atlases with a single region, and palettes that are merely dark but varied, are left untouched: this replaces palettes that cannot be read, not palettes that are unusual.

Falling back warns, naming the atlas, because the fix belongs in the atlas's colour lookup table. An atlas carrying no palette at all is a supported, expected state rather than a broken table, so it falls back silently.

Value

Named character vector of colours keyed by label.

See Also

`atlas_palette()` for the palette exactly as stored.

Other atlas accessors: `atlas_centerlines()`, `atlas_context()`, `atlas_display()`, `atlas_geom()`, `atlas_geometry_type()`, `atlas_hemi()`, `atlas_labels()`, `atlas_meshes()`, `atlas_palette()`, `atlas_polygons()`, `atlas_provenance()`, `atlas_region_semantics()`, `atlas_regions()`, `atlas_sf()`, `atlas_type()`, `atlas_undrawn()`, `atlas_vertices()`, `atlas_views()`, `atlas_views_empty()`, `legacy_region_map()`

Examples

```
# A well-formed palette comes back untouched
identical(atlas_plot_palette(dk()), atlas_palette(dk()))
```

atlas_polygons *Get atlas polygons for 2D rendering*

Description

Returns the brain_polygons representation of the atlas geometry, converting from sf when needed. The sf-optional counterpart to atlas_sf().

Usage

```
atlas_polygons(atlas)
```

Arguments

atlas	a ggseg_atlas object

Value

a brain_polygons data.frame

See Also

Other atlas accessors: atlas_centerlines(), atlas_context(), atlas_display(), atlas_geom(), atlas_geometry_type(), atlas_hemi(), atlas_labels(), atlas_meshes(), atlas_palette(), atlas_plot_palette(), atlas_provenance(), atlas_region_semantics(), atlas_regions(), atlas_sf(), atlas_type(), atlas_undrawn(), atlas_vertices(), atlas_views(), atlas_views_empty(), legacy_region_map()

Examples

```
polys <- atlas_polygons(dk())
```

atlas_provenance *How an atlas was built*

Description

Returns the build record an atlas carries, or NULL for an atlas built before provenance existed or by a pipeline that did not record one.

Usage

```
atlas_provenance(atlas)
```

Arguments

atlas	a ggseg_atlas object

Details

Provenance answers a question the rest of the class cannot: not what the atlas contains, but how it came to contain it. Two atlases produced by the same pipeline can differ in view count, polygon simplification or palette source because they were built with different arguments, and without a record the only way to find out is to locate the build script that produced them – which lives in a different repository and may have changed since.

Value

A named list, or `NULL`. When present it carries at least `pipeline` (what built the atlas, including its version) and `created` (when). Any further fields are the producing pipeline's own, commonly the source parcellation and the arguments the build was given.

See Also

`set_atlas_provenance()` to attach one.

Other atlas accessors: `atlas_centerlines()`, `atlas_context()`, `atlas_display()`, `atlas_geom()`, `atlas_geometry_type()`, `atlas_hemi()`, `atlas_labels()`, `atlas_meshes()`, `atlas_palette()`, `atlas_plot_palette()`, `atlas_polygons()`, `atlas_region_semantics()`, `atlas_regions()`, `atlas_sf()`, `atlas_type()`, `atlas_undrawn()`, `atlas_vertices()`, `atlas_views()`, `atlas_views_empty()`, `legacy_region_map()`

Examples

```
atlas_provenance(dk())
```

---

atlas_region_semantics

Does this atlas's `region` *aggregate across hemispheres?*

Description

Answers which of the two conventions described in core_contract an atlas follows, so a consumer joining results across parcellations does not have to work it out.

Usage

```
atlas_region_semantics(atlas)
```

Arguments

`atlas`	A `ggseg_atlas`.

Details

Computed from `core` rather than stored, so it cannot disagree with the data it describes: a stored flag would be one more thing to keep true through every verb that edits an atlas.

Value

One of:

`"shared"` `region` names are shared between the hemispheres, so grouping by `region` aggregates a structure across sides.

`"per-hemisphere"` No name is shared, so `region` is 1:1 with `label` and grouping by it aggregates nothing.

`"unilateral"` The atlas has fewer than two lateral hemispheres, so the question does not arise.

See Also

Other atlas accessors: `atlas_centerlines()`, `atlas_context()`, `atlas_display()`, `atlas_geom()`, `atlas_geometry_type()`, `atlas_hemi()`, `atlas_labels()`, `atlas_meshes()`, `atlas_palette()`, `atlas_plot_palette()`, `atlas_polygons()`, `atlas_provenance()`, `atlas_regions()`, `atlas_sf()`, `atlas_type()`, `atlas_undrawn()`, `atlas_vertices()`, `atlas_views()`, `atlas_views_empty()`, `legacy_region_map()`

Examples

```
atlas_region_semantics(dk())
atlas_region_semantics(aseg())
```

---

atlas_regions *Extract the region column of an atlas*

Description

Extract the region column of an atlas

Usage

```
atlas_regions(x)

brain_regions(x)
```

Arguments

`x`	brain atlas

Value

The `region` column of `core`, unchanged: one element per `core` row, in `core` row order, with repeats and `NA`s retained. It is therefore row-aligned with `atlas_labels()` and `atlas_display()`. Use `unique()` for the distinct set. A zero-length character vector when the atlas carries no `region` column.

See Also

Other atlas accessors: atlas_centerlines(), atlas_context(), atlas_display(), atlas_geom(), atlas_geometry_type(), atlas_hemi(), atlas_labels(), atlas_meshes(), atlas_palette(), atlas_plot_palette(), atlas_polygons(), atlas_provenance(), atlas_region_semantics(), atlas_sf(), atlas_type(), atlas_undrawn(), atlas_vertices(), atlas_views(), atlas_views_empty(), legacy_region_map()

Examples

```
atlas_regions(dk())
atlas_regions(aseg())
```

---

atlas_relabel *Re-key an atlas's labels*

Description

`label` is an atlas's join key: `core`, the palette's names and every geometry payload (`geom`, `vertices`, `meshes`, `centerlines`) are keyed on it. Rewriting it in one of those places and not the others silently decouples geometry from metadata, which is how a renamed region ends up drawn in the wrong colour or not at all, so this rewrites all of them together.

Usage

```
atlas_relabel(atlas, mapping)

relabel_atlas(atlas, mapping)
```

Arguments

`atlas`	a `ggseg_atlas` object
`mapping`	Named character vector: names are the atlas's current labels, values the replacements.

Details

Labels absent from `mapping` are left alone, so contextual geometry such as the `cortex_` silhouette survives a partial re-key untouched.

This is the supported way to correct an atlas's identifiers – the operation `tracula()` itself needed when its `.bbr.prep` suffix was restored. Use atlas_region_rename() instead to change the display names in `region`, which is not a join key.

Value

The `ggseg_atlas` with `mapping` applied to `core$label`, the palette's names and every payload slot that carries a `label` column.

Functions

- `relabel_atlas()`: Deprecated. Use `atlas_relabel()`: everything else that acts on an atlas begins `atlas_`, and a verb-first name never appeared when completing `atlas_`, so this one was present but unfindable.

See Also

`atlas_region_rename()` to rename regions rather than re-key labels.

Other atlas manipulations: `atlas_core_add()`, `atlas_manipulation`, `atlas_structure_reorder()`, `drawing_manipulation`, `region_manipulation`, `view_manipulation`

Examples

```
a <- atlas_relabel(aseg(), c("Left-Thalamus" = "lh_thalamus"))
head(atlas_labels(a))
```

`atlas_sf` *Get atlas data for 2D rendering*

Description

Returns sf data joined with core region info and palette colours. This is the interception point used by ggseg for plotting: it always returns sf geometry, converting from the polygon representation when needed.

Usage

```
atlas_sf(atlas)
```

Arguments

`atlas`	a ggseg_atlas object

Value

sf data.frame ready for plotting

See Also

Other atlas accessors: `atlas_centerlines()`, `atlas_context()`, `atlas_display()`, `atlas_geom()`, `atlas_geometry_type()`, `atlas_hemi()`, `atlas_labels()`, `atlas_meshes()`, `atlas_palette()`, `atlas_plot_palette()`, `atlas_polygons()`, `atlas_provenance()`, `atlas_region_semantics()`, `atlas_regions()`, `atlas_type()`, `atlas_undrawn()`, `atlas_vertices()`, `atlas_views()`, `atlas_views_empty()`, `legacy_region_map()`

Examples

```
sf_data <- atlas_sf(dk())
head(sf_data)
```

`atlas_structure_reorder`

Reorder the structures of an atlas

Description

Moves structures within an atlas's geometry, the way `dplyr::relocate()` moves columns. Geometry rows are drawn in the order they appear, so this is what decides which structure is painted over which where two overlap.

Overlap is common in subcortical atlases: a slab is a thick slice of the volume flattened into one panel, so structures that never touch in the brain can still overlap on the page.

Usage

```
atlas_structure_reorder(
  atlas,
  structures,
  .before = NULL,
  .after = NULL,
  match_on = c("label", "region")
)
```

Arguments

`atlas`	A `ggseg_atlas` object.
`structures`	Structures to move, as a character vector matched against `match_on`. They are moved together, keeping the order given here.
`.before, .after`	One structure to move `structures` next to. Give at most one of the two.
`match_on`	Column of `core` to match against: `"label"`, the default, or `"region"`. `label` is the identifier that is unique within an atlas and stable across them, so every verb that selects rows by name or pattern reads it unless told otherwise.

Details

Later rows are drawn on top. With neither `.before` nor `.after`, the structures move to the front, matching `dplyr::relocate()`, which means they are drawn *first*, and so end up *behind* everything else. To bring a structure to the front visually, move it after the last one.

The order is a property of the structure, not of a single view: an atlas holds one geometry row per structure with its views nested inside, so a structure keeps the same depth in every view it appears in.

Value

The ggseg_atlas, with its geometry rows in the new order.

See Also

atlas_view_reorder(), which moves whole views around the canvas rather than structures within them.

Other atlas manipulations: atlas_core_add(), atlas_manipulation, atlas_relabel(), drawing_manipulation, region_manipulation, view_manipulation

Examples

```
a <- aseg()

# Draw the thalamus on top of everything else
last <- utils::tail(atlas_labels(a), 1)
a <- atlas_structure_reorder(a, "Left-Thalamus", .after = last)

# Or tuck it behind its neighbours
a <- atlas_structure_reorder(a, "Left-Thalamus", .before = "Left-Putamen")
```

---

atlas_type *Detect atlas type*

Description

Detect atlas type

Usage

```
atlas_type(x)
```

Arguments

x	brain atlas object

Value

Character string: one of "cortical", "subcortical", "tract" or "cerebellar".

See Also

set_atlas_type() to set the type.

Other atlas accessors: atlas_centerlines(), atlas_context(), atlas_display(), atlas_geom(), atlas_geometry_type(), atlas_hemi(), atlas_labels(), atlas_meshes(), atlas_palette(), atlas_plot_palette(), atlas_polygons(), atlas_provenance(), atlas_region_semantics(), atlas_regions(), atlas_sf(), atlas_undrawn(), atlas_vertices(), atlas_views(), atlas_views_empty(), legacy_region_map()

Examples

```
atlas_type(dk())
atlas_type(aseg())
atlas_type(tracula())
```

`atlas_undrawn` *The regions an atlas names but never draws*

Description

A region in `core` with no geometry in any view is invisible: it joins, it carries a colour, and nothing appears. It is the failure mode of choosing which slabs to keep, and it is silent – the atlas validates, plots without complaint, and simply omits the region.

Usage

```
atlas_undrawn(atlas)
```

Arguments

`atlas`	A `ggseg_atlas`.

Details

Build scripts narrow a pipeline's slabs to the few that read well. That choice is a measurement of the geometry underneath it, so it goes stale when the geometry changes. Asserting on this after the narrowing turns a silent omission into a failed build.

Value

The `core` labels with no geometry in any view, as a character vector; empty when every region is drawn somewhere.

See Also

`atlas_view_keep()` and `atlas_view_remove()`, which narrow the views this checks.

Other atlas accessors: `atlas_centerlines()`, `atlas_context()`, `atlas_display()`, `atlas_geom()`, `atlas_geometry_type()`, `atlas_hemi()`, `atlas_labels()`, `atlas_meshes()`, `atlas_palette()`, `atlas_plot_palette()`, `atlas_polygons()`, `atlas_provenance()`, `atlas_region_semantics()`, `atlas_regions()`, `atlas_sf()`, `atlas_type()`, `atlas_vertices()`, `atlas_views()`, `atlas_views_empty()`, `legacy_region_map()`

Examples

```
atlas_undrawn(aseg())
```

atlas_vertices *Get atlas vertices for 3D rendering*

Description

Returns vertices data joined with core region info and palette colours. Used for cortical atlases with vertex-based rendering.

Usage

```
atlas_vertices(atlas)
```

Arguments

atlas	a ggseg_atlas object

Value

data.frame with vertices ready for 3D rendering

See Also

Other atlas accessors: atlas_centerlines(), atlas_context(), atlas_display(), atlas_geom(), atlas_geometry_type(), atlas_hemi(), atlas_labels(), atlas_meshes(), atlas_palette(), atlas_plot_palette(), atlas_polygons(), atlas_provenance(), atlas_region_semantics(), atlas_regions(), atlas_sf(), atlas_type(), atlas_undrawn(), atlas_views(), atlas_views_empty(), legacy_region_map()

Examples

```
verts <- atlas_vertices(dk())
head(verts)
```

atlas_views *Get available views in atlas*

Description

Get available views in atlas

Usage

```
atlas_views(atlas)

brain_views(atlas)
```

Arguments

atlas	A ggseg_atlas object

Value

Character vector of view names, or NULL if no sf data

See Also

Other atlas accessors: atlas_centerlines(), atlas_context(), atlas_display(), atlas_geom(), atlas_geometry_type(), atlas_hemi(), atlas_labels(), atlas_meshes(), atlas_palette(), atlas_plot_palette(), atlas_polygons(), atlas_provenance(), atlas_region_semantics(), atlas_regions(), atlas_sf(), atlas_type(), atlas_undrawn(), atlas_vertices(), atlas_views_empty(), legacy_region_map()

Examples

```
atlas_views(aseg())
atlas_views(tracula())
```

---

atlas_views_empty *The views an atlas has that draw no region*

Description

A view holding nothing but context is a panel of grey: it takes up room in the layout, carries a title, and shows no region of the atlas. It is the mirror of atlas_undrawn() – that reports regions no view draws, this reports views that draw no region – and the same silence applies, since the atlas validates and plots without complaint.

Usage

```
atlas_views_empty(atlas)
```

Arguments

atlas	A ggseg_atlas.

Details

Usually it is a consequence of your own editing rather than of the build: remove the ventricles from an aseg, or prune small depictions, and a view whose only content was those regions is left empty.

This reports rather than removes, so the decision stays with you, and it measures the atlas you hand it. To act on the answer use ggseg.extra::atlas_view_remove_empty(), which takes the atlas once and so always measures the atlas it is about to change — naming the atlas twice to feed this into atlas_view_remove() reads as a pipe but is not one, and in a longer pipeline the name in hand is the atlas *before* the editing that emptied the view. ggseg.extra::atlas_view_remove_sparse() is the build-time counterpart, for views a pipeline left *thin* against a threshold rather than empty.

Value

The view names drawing no `core` region, as a character vector; empty when every view draws at least one, and when the atlas carries no 2D geometry at all.

See Also

`atlas_undrawn()`, its mirror, and `ggseg.extra::atlas_view_remove_empty()` to act on the result.

Other atlas accessors: `atlas_centerlines()`, `atlas_context()`, `atlas_display()`, `atlas_geom()`, `atlas_geometry_type()`, `atlas_hemi()`, `atlas_labels()`, `atlas_meshes()`, `atlas_palette()`, `atlas_plot_palette()`, `atlas_polygons()`, `atlas_provenance()`, `atlas_region_semantics()`, `atlas_regions()`, `atlas_sf()`, `atlas_type()`, `atlas_undrawn()`, `atlas_vertices()`, `atlas_views()`, `legacy_region_map()`

Examples

```
atlas_views_empty(aseg())
```

---

attach_sfc *Attach an* `sfc` *column and class a data frame as* `sf`*, without sf*

Description

`merge()`, `cbind()` and `[` dispatch to sf's own methods only when the sf namespace is loaded. Without it each strips a different part of the sf representation: `merge()` loses the class, the `sf_column` attribute and the column's own `sfc` class; `cbind()` loses the class and the attribute; subsetting loses the column's class. Attribute-space work therefore happens with the geometry detached and this is the one place that puts it back.

Usage

```
attach_sfc(x, geometry, sf_column = "geometry", after = NULL)
```

Arguments

`x`	a data.frame, without its geometry column.
`geometry`	an `sfc`, in `x`'s row order.
`sf_column`	name to give the geometry column.
`after`	insert the column after this many other columns, so the result keeps the column order callers already see. `NULL` appends.

Details

`agr` is a named factor over the non-geometry columns with sf's three levels and no values set, which is what `st_as_sf()` produces for a frame carrying no attribute-geometry relationships.

Value

`x` with `geometry` attached, classed as `sf`.

See Also

`reorder_sfc()` to permute the geometry alongside a reordered attribute table.

Other sf-optional helpers: `reorder_sfc()`

Examples

```
atlas <- dk()
sf_data <- atlas_sf(atlas)
geometry <- sf_data$geometry

# Attribute-space work with the geometry detached, then reattached.
attrs <- sf_data
attrs$geometry <- NULL
class(attrs) <- "data.frame"
attrs$keep <- attrs$label %in% atlas_labels(atlas)[1:5]

rebuilt <- attach_sfc(attrs, geometry)
class(rebuilt)
inherits(rebuilt$geometry, "sfc")
```

`convert_legacy_brain_atlas` *Convert legacy ggseg atlases to ggseg_atlas format*

Description

[Superseded]

Convert old-style ggseg (2D) and ggseg3d (3D) atlases into the new `ggseg_atlas` format. This is a bridge function for working with existing atlases during the transition period.

For new atlases, use `ggsegExtra::create_cortical_atlas()` or `ggsegExtra::create_subcortical_atlas()` instead - they produce better results with proper vertex indices.

The function handles three scenarios:

- **Both 2D and 3D**: Merges geometry with vertex data
- **3D only**: Extracts vertices, infers indices from mesh coordinates
- **2D only**: Keeps geometry, 3D rendering unavailable

If the 3D atlas already contains vertex indices (newer ggseg3d atlases), those are preserved. Otherwise, vertex indices are inferred from mesh coordinates using hash-based matching (no FreeSurfer needed).

Usage

```
convert_legacy_brain_atlas(
  atlas_2d = NULL,
  atlas_3d = NULL,
  atlas_name = NULL,
  type = NULL,
  surface = "inflated",
  brain_meshes = NULL
)

unify_legacy_atlases(
  atlas_2d = NULL,
  atlas_3d = NULL,
  atlas_name = NULL,
  type = NULL,
  surface = "inflated",
  brain_meshes = NULL
)
```

Arguments

atlas_2d	A ggseg_atlas (or legacy brain_atlas) with 2D geometry, or NULL.
atlas_3d	A ggseg3d_atlas with mesh data, or NULL.
atlas_name	Name for the output atlas. If NULL, derived from input.
type	Atlas type: "cortical" or "subcortical". If NULL, inferred from the input atlases.
surface	Which surface to match against when inferring vertices (e.g., "inflated"). Must match the 3D atlas surface exactly.
brain_meshes	Optional user-supplied brain meshes for vertex inference.

Value

A ggseg_atlas object.

Examples

```
new_atlas <- convert_legacy_brain_atlas(atlas_2d = dk())
```

---

core_contract *What an atlas's three name columns mean*

Description

Every ggseg_atlas keys its core on three columns, and they are not interchangeable.

The two conventions for `region`

A parcellation that names its structures anatomically gives both hemispheres the same name, so `region` aggregates them and `hemi` disambiguates: grouping by `region` gives one row per structure.

A parcellation that numbers its parcels does not. `region_0001` and `region_0002` are two different parcels, not one structure seen twice, and no hemisphere-free name exists to recover – the source never gave one. There `region` is 1:1 with `label` and grouping by it aggregates nothing.

Both are correct for their source. What was missing was a way to ask which you have, short of testing whether any name is shared across hemispheres. `atlas_region_semantics()` is that question.

See Also

`atlas_region_semantics()`, `atlas_display()`, `atlas_context()`

---

dk *Desikan-Killiany Cortical Atlas*

Description

Returns the Desikan-Killiany cortical parcellation atlas with 34 regions per hemisphere (68 total) on the cortical surface.

Usage

```
dk()
```

Details

This atlas is based on the FreeSurfer `aparc` annotation and is one of the most widely used cortical parcellations in neuroimaging research.

The atlas works with both ggseg (2D polygon plots) and ggseg3d (3D mesh visualizations) from a single object.

Value

A `ggseg_atlas` object with components:

atlas Character. Atlas name ("dk")

type Character. Atlas type ("cortical")

palette Named character vector of colours for each region

data A `ggseg_data_cortical` object containing:

- **vertices** Data frame with `label` and `vertices` columns
- **sf** Simple features data frame for 2D rendering

Core columns

`core` carries one row per atlas region with the columns `hemi`, `region`, `label` and `names`, plus the atlas's own grouping column. `label` is the stable atlas identifier and the key the palette and the geometry are keyed on; `region` is a short, hemisphere-free key derived from it; `names` is the curated long-form display name, spelled out to read well in a figure legend.

`region` held the long-form name until version 0.1.0. `names` is not a record of those old values – use `legacy_region_map()` to translate them.

Regions

The atlas contains 34 regions per hemisphere including: banks of superior temporal sulcus, caudal anterior cingulate, caudal middle frontal, cuneus, entorhinal, fusiform, inferior parietal, inferior temporal, isthmus cingulate, lateral occipital, lateral orbitofrontal, lingual, medial orbitofrontal, middle temporal, parahippocampal, paracentral, pars opercularis, pars orbitalis, pars triangularis, pericalcarine, postcentral, posterior cingulate, precentral, precuneus, rostral anterior cingulate, rostral middle frontal, superior frontal, superior parietal, superior temporal, supramarginal, frontal pole, temporal pole, transverse temporal, and insula.

References

Desikan RS, Segonne F, Fischl B, et al. (2006). An automated labeling system for subdividing the human cerebral cortex on MRI scans into gyral based regions of interest. NeuroImage, 31(3):968-980. doi:10.1016/j.neuroimage.2006.01.021

Fischl B, van der Kouwe A, Destrieux C, et al. (2004). Automatically parcellating the human cerebral cortex. Cerebral Cortex, 14(1):11-22. doi:10.1093/cercor/bhg087

See Also

`aseg()` for subcortical structures, `ggseg_atlas()` for the atlas class constructor

Other ggseg_atlases: `aseg()`, `suit()`, `tracula()`

Examples

```
dk()
plot(dk())
atlas_regions(dk())
atlas_labels(dk())
```

---

drawing_manipulation *Change which regions are drawn inside a view*

Description

Verbs that prune depictions, each by a different criterion:

Usage

```
atlas_prune_region(
  atlas,
  pattern,
  match_on = c("label", "region"),
  views = NULL,
  match = c("exact", "pattern")
)

atlas_prune_small(
  atlas,
  min_area,
  views = NULL,
  scope = c("region", "piece"),
  labels = NULL,
  exclude = NULL,
  match = c("exact", "pattern")
)

atlas_prune_weak(atlas, threshold = 0.5, weights = NULL)

atlas_view_select(atlas, threshold = 0.5, weights = NULL)

atlas_view_remove_region(
  atlas,
  pattern,
  match_on = c("label", "region"),
  views = NULL,
  match = c("exact", "pattern")
)

atlas_view_remove_small(
  atlas,
  min_area,
  views = NULL,
  scope = c("region", "piece"),
  labels = NULL,
  exclude = NULL,
  match = c("exact", "pattern")
)
```

Arguments

atlas	A ggseg_atlas object
pattern	Character pattern to match. Uses grepl(..., ignore.case = TRUE).
match_on	Column of core to match against: "label", the default, or "region". label is the identifier that is unique within an atlas and stable across them, so every verb that selects rows by name or pattern reads it unless told otherwise.

views	Views to scope the pruning to, or NULL (the default) for every view. Matched whole and case-insensitively against the atlas's own views, so a name has to be one atlas_views() reports. Give them as a character vector, or in one string separated by "|"; the two are equivalent. Names that match no view are warned about and otherwise ignored.
match	How views is read. "exact", the default, takes each name as a whole view name. "pattern" takes views as a regular expression instead, matched as a substring — "axial" then reaches axial_1 and axial_2 as well. Prefer "exact": a name that happens to sit inside another view's name silently scopes to both views under "pattern".
min_area	For atlas_prune_small(): minimum area to keep. What it applies to depends on scope; with the default scope = "region" context geometries are never removed.
scope	Whether min_area applies to a label's whole geometry in a view ("region", the default) or to each disconnected piece ("piece").
labels, exclude	For atlas_prune_small(): optional regex scoping which labels are considered. labels restricts removal to matching labels, exclude spares them. Only one may be given. Useful with scope = "piece", where a thin structure such as a cortical ribbon has legitimately small pieces that should not be treated as specks.
threshold	For atlas_prune_weak(): minimum share, between 0 and 1, of a region's best-view area that a view must hold to keep drawing it. Higher values are more aggressive.
weights	For atlas_prune_weak(): optional named numeric vector of per-view multipliers applied before comparing areas. Overrides the automatic hemisphere-coverage weighting for the views named.

Details

- atlas_prune_region(): by name
- atlas_prune_small(): by absolute area
- atlas_prune_weak(): by area as a share of the region's best view

None of these adds a view, and none removes a region from core: the region stays and is still legended, while its depiction in some view goes. A view can still disappear as a consequence — atlas_prune_weak() drops a view that keeps no region, and the other two drop a view whose last remaining geometry they removed.

atlas_prune_small() and atlas_prune_weak() are the same operation asked two ways, an absolute area against a share of the region's largest appearance, with one difference that decides between them: atlas_prune_weak() keeps every region in at least one view, while atlas_prune_small() with the default scope = "region" can take a region's geometry out of every view, leaving it legended but drawn nowhere — and the higher the threshold, the more regions that happens to. Check atlas_undrawn() after raising min_area.

For views as objects, see view_manipulation; to remove a region from the atlas altogether, region_manipulation.

Value

Modified ggseg_atlas object

Functions

- atlas_prune_small(): Remove geometry below a minimum area threshold. With scope = "region" (the default) a label's whole geometry in a view is removed when its combined area is too small, and context geometries (labels not in core) are never removed. With scope = "piece" individual disconnected pieces are removed while the rest of the region stays – use this to clear stray specks left by volumetric projection. A region's largest piece anywhere in the atlas is always kept, so no region disappears – a speck in one view goes when the same region is drawn properly in another. Context is cleaned too. Optionally scope to specific views. Views are re-packed via atlas_view_gather() in case any view shrank.
- atlas_prune_weak(): Keep each region only in the views that show it well.

  A slice or projection slab catches a structure in cross-section as readily as along its length, so most regions leave a sliver in most views. That leaves every panel cluttered and every region drawn several times over. A region is kept only where it is substantially represented: in any view holding at least threshold of the area it reaches in its best view.

  Regions are compared as a whole, across all the labels that share a region in core, so bilateral structures stay together – assigning left and right independently splits pairs across panels, which reads as an error rather than a choice. Every region is guaranteed to survive in at least one view, and context geometry (labels absent from core, such as a cortical outline) is never touched.

  Single-hemisphere views are the awkward case: a sagittal panel cuts one hemisphere while axial and coronal panels show both, so on raw area it loses every comparison and empties out. Such views are detected from the hemispheres actually present in each view and their areas scaled to match, so they compete fairly. Pass weights to override.

  What it costs. The shares are spread rather than clustered, so no threshold separates slivers from substance: on the bundled aseg() the median depiction outside a region's best view still holds 56% of it. Raising threshold buys a less crowded figure by dropping depictions that are anatomically real, and atlas_undrawn() will not report the loss, because every region is kept somewhere. Look at the views.
- atlas_view_select(): Deprecated. Use atlas_prune_weak(): the old name read as the verb for choosing which views an atlas keeps, which is atlas_view_keep(), and not for choosing which views draw a region.
- atlas_view_remove_region(): Deprecated. Use atlas_prune_region(): the old name read as a verb on views, when it removes no view — the region stays in core and only its depiction in the named views goes. For removing views themselves see atlas_view_remove().
- atlas_view_remove_small(): Deprecated. Use atlas_prune_small(): the old name read as a verb on views, when it removes no view. Note the near-namesake in ggseg.extra, atlas_view_remove_sparse(), which really does drop views; the two were a letter apart in intent and are not any more.

See Also

atlas_manipulation for the other groups

Other atlas manipulations: atlas_core_add(), atlas_manipulation, atlas_relabel(), atlas_structure_reorder(), region_manipulation, view_manipulation

Examples

```
dk() |> atlas_prune_small(min_area = 10)
```

`get_brain_mesh` *Get brain surface mesh*

Description

Retrieves a brain surface mesh for the specified hemisphere and surface type. By default, provides the inflated fsaverage5 surface from internal data. Other surfaces (pial, white, semi-inflated) require the ggseg3d package.

Usage

```
get_brain_mesh(
  hemisphere = c("lh", "rh"),
  surface = "inflated",
  brain_meshes = NULL
)
```

Arguments

`hemisphere`	`"lh"` or `"rh"`
`surface`	Surface type (default `"inflated"`). Other surfaces require ggseg3d or a custom `brain_meshes` argument.
`brain_meshes`	Optional user-supplied mesh data. Accepts either `list(lh = list(vertices, faces), rh = ...)` or the legacy `list(lh_inflated = list(vertices, faces), ...)` format.

Value

A list with `vertices` (data.frame with x, y, z) and `faces` (data.frame with i, j, k), or NULL if the mesh is not available. The face indices of the bundled inflated surfaces are **0-based** (0 to `nrow(vertices) - 1`), matching `get_cerebellar_mesh()`. Per-region subcortical meshes from `ggseg_data_subcortical()` are 1-based instead, and a user-supplied `brain_meshes` is returned exactly as given, so a consumer must not assume a single base across mesh sources.

Examples

```
mesh <- get_brain_mesh("lh")
head(mesh$vertices)
```

get_cerebellar_mesh *Get SUIT cerebellar surface mesh*

Description

Retrieves the shared SUIT cerebellar surface mesh used for vertex-based cerebellar atlas rendering. The mesh contains 28,935 original surface vertices plus 1,078 additional vertices forming a cap over the peduncular surface (where the cerebellum meets the brainstem). The cap vertices (indices 28,935–30,012) are duplicates of the boundary loop and a centroid, designed to render as an opaque grey wall when not assigned to any atlas region.

Usage

```
get_cerebellar_mesh()
```

Value

A list with `vertices` (data.frame with x, y, z) and `faces` (data.frame with i, j, k, 0-based indices).

Examples

```
mesh <- get_cerebellar_mesh()
nrow(mesh$vertices)
```

ggseg_atlas *Constructor for ggseg atlas*

Description

Creates an object of class 'ggseg_atlas' for plotting brain parcellations using ggseg (2D) and ggseg3d (3D).

Usage

```
ggseg_atlas(atlas, type, core, data, palette = NULL, provenance = NULL)

brain_atlas(atlas, type, core, data, palette = NULL)
```

Arguments

`atlas`	atlas short name, length one
`type`	atlas type: "cortical", "subcortical", "tract", or "cerebellar"
`core`	data.frame with required columns hemi, region, label. `label` is the key the palette and every geometry slot are joined on, so it must be unique: a duplicated `label` is an error. May contain additional columns for grouping or metadata (e.g., lobe, network, Brodmann area).

data	a ggseg_atlas_data object created by ggseg_data_cortical(), ggseg_data_subcortical(), ggseg_data_tract(), or ggseg_data_cerebellar(). Must match the specified type.
palette	named character vector of colours keyed by label
provenance	optional named list recording how the atlas was built. Must carry pipeline and created when supplied; see set_atlas_provenance(). NULL for an atlas whose build is unrecorded.

Value

an object of class 'ggseg_atlas'

The type vocabulary is closed

type accepts exactly four values, and the constructor rejects any other. This is deliberate rather than a limitation waiting to be lifted. Between them, cortical, subcortical, tract and cerebellar cover the major ways a brain is drawn on a 2D surface, and 3D is covered by meshes.

A type is not merely a label: it implies a geometry form, a view vocabulary and a rendering contract that both ggseg and ggseg3d must implement. An atlas carrying a type no renderer knows how to draw is one that cannot be plotted, so the vocabulary is governed rather than open. The ecosystem is open to adding a type, but adding one is a discussion to raise with the maintainers first, not something a contributing package can do on its own. Atlases *of the existing types* need no such discussion: that path is open, documented, and is how every distributed atlas package was built.

Examples

```
core <- data.frame(
  hemi = c("left", "left"),
  region = c("region1", "region2"),
  label = c("lh_region1", "lh_region2")
)
vertices <- data.frame(
  label = c("lh_region1", "lh_region2"),
  vertices = I(list(c(1L, 2L, 3L), c(4L, 5L, 6L)))
)
atlas <- ggseg_atlas(
  atlas = "test",
  type = "cortical",
  core = core,
  data = ggseg_data_cortical(vertices = vertices)
)
```

---

ggseg_data_cerebellar *Create cerebellar atlas data*

Description

Creates a data object for cerebellar brain atlases. Cerebellar atlases use sf polygons from a SUIT flatmap for 2D rendering and vertex indices into the shared SUIT cerebellar surface mesh for 3D rendering.

Usage

```
ggseg_data_cerebellar(geom = NULL, vertices = NULL, meshes = NULL, ...)
```

Arguments

geom	2D geometry for rendering, stored in the single geom slot: either an sf data.frame (columns label, view, geometry) or a brain_polygons data.frame (see sf_to_polygons()). The class of geom determines the rendering path used downstream.
vertices	data.frame with columns label and vertices (list-column of integer vectors). Each vector contains 0-based vertex indices into the SUIT cerebellar surface (see get_cerebellar_mesh()). Only for surface regions.
meshes	Optional data.frame with columns label and mesh (list-column of mesh objects with vertices and faces). For deep cerebellar structures that are not on the cortical surface. Same format as ggseg_data_subcortical() meshes.
...	Captures a deprecated sf argument: if supplied it is converted to the polygon representation via sf_to_polygons() and a deprecation warning is issued. Prefer passing 2D geometry via geom.

Details

The shared mesh (see get_cerebellar_mesh()) includes a cap over the peduncular surface where the cerebellum meets the brainstem. Vertices on this cap (indices 28,935–30,012) are not assigned to any atlas region and render as na_colour in 3D, analogous to the medial wall in cortical atlases.

Deep cerebellar structures (e.g. dentate, interposed, fastigial nuclei) that are not on the cortical surface are stored as individual per-region meshes in meshes, following the same format as subcortical atlases. Their 2D sf geometries use views other than "flatmap" (e.g. "nuclei").

Value

An object of class c("ggseg_data_cerebellar", "ggseg_atlas_data")

Examples

```
data <- ggseg_data_cerebellar(
  vertices = data.frame(
    label = c("lobule_I", "lobule_II"),
    vertices = I(list(c(1L, 2L, 3L), c(4L, 5L, 6L)))
  )
)
```

`ggseg_data_cortical` *Create cortical atlas data*

Description

Creates a data object for cortical brain atlases. Cortical atlases use vertex indices that map regions to vertices on a shared brain surface mesh (e.g., fsaverage5).

Usage

```
ggseg_data_cortical(geom = NULL, vertices = NULL, ...)

brain_data_cortical(sf = NULL, vertices = NULL)
```

Arguments

`geom`	2D geometry for rendering, stored in the single `geom` slot: either an sf data.frame (columns `label`, `view`, `geometry`) or a `brain_polygons` data.frame (see `sf_to_polygons()`). The class of `geom` determines the rendering path used downstream.
`vertices`	data.frame with columns label and vertices (list-column of integer vectors). Each vector contains vertex indices for that region.
`...`	Captures a deprecated `sf` argument: if supplied it is converted to the polygon representation via `sf_to_polygons()` and a deprecation warning is issued. Prefer passing 2D geometry via `geom`.
`sf`	Deprecated. Pass 2D geometry via `geom` instead.

Value

An object of class c("ggseg_data_cortical", "ggseg_atlas_data")

Examples

```
data <- ggseg_data_cortical(
  vertices = data.frame(
    label = c("bankssts", "caudalanteriorcingulate"),
    vertices = I(list(c(1L, 2L, 3L), c(4L, 5L, 6L)))
  )
)
```

ggseg_data_subcortical *Create subcortical atlas data*

Description

Creates a data object for subcortical brain atlases. Subcortical atlases use individual 3D meshes for each structure (e.g., hippocampus, amygdala).

Usage

```
ggseg_data_subcortical(geom = NULL, meshes = NULL, ...)

brain_data_subcortical(sf = NULL, meshes = NULL)
```

Arguments

geom	2D geometry for rendering, stored in the single geom slot: either an sf data.frame (columns label, view, geometry) or a brain_polygons data.frame (see sf_to_polygons()). The class of geom determines the rendering path used downstream.
meshes	data.frame with columns label and mesh (list-column). Each mesh is a list with: • vertices: data.frame with x, y, z columns • faces: data.frame with i, j, k columns (1-based triangle indices)
...	Captures a deprecated sf argument: if supplied it is converted to the polygon representation via sf_to_polygons() and a deprecation warning is issued. Prefer passing 2D geometry via geom.
sf	Deprecated. Pass 2D geometry via geom instead.

Value

An object of class c("ggseg_data_subcortical", "ggseg_atlas_data")

Examples

```
data <- ggseg_data_subcortical(
  meshes = data.frame(
    label = "hippocampus_left",
    mesh = I(list(list(
      vertices = data.frame(x = 1:10, y = 1:10, z = 1:10),
      faces = data.frame(i = 1:3, j = 2:4, k = 3:5)
    )))
  )
)
```

`ggseg_data_tract` *Create tract atlas data*

Description

Creates a data object for white matter tract atlases. Stores centerlines compactly; tube meshes are generated at render time for efficiency.

Usage

```
ggseg_data_tract(geom = NULL, centerlines = NULL, meshes = NULL, ...)

brain_data_tract(sf = NULL, centerlines = NULL, meshes = NULL, ...)
```

Arguments

`geom`	2D geometry for rendering, stored in the single `geom` slot: either an sf data.frame (columns `label`, `view`, `geometry`) or a `brain_polygons` data.frame (see `sf_to_polygons()`). The class of `geom` determines the rendering path used downstream.
`centerlines`	data.frame with columns: • label: tract identifier (character) • points: list-column of n x 3 matrices (centerline coordinates) • tangents: list-column of n x 3 matrices (for orientation colouring)
`meshes`	**[Deprecated]** Use `centerlines` instead. If supplied, it is converted to the centerlines format.
`...`	Captures a deprecated `sf` argument (converted to polygons) and absorbs legacy fields (e.g. tube_radius, tube_segments) from old cached atlas objects.
`sf`	Deprecated. Pass 2D geometry via `geom` instead.

Value

An object of class c("ggseg_data_tract", "ggseg_atlas_data")

Examples

```
centerlines_df <- data.frame(
  label = "cst_left",
  points = I(list(matrix(rnorm(150), ncol = 3))),
  tangents = I(list(matrix(rnorm(150), ncol = 3)))
)
data <- ggseg_data_tract(centerlines = centerlines_df)
```

`ggseg_data_wm` *Create white-matter atlas data*

Description

Creates a data object for white-matter brain atlases. A white-matter atlas stores what a subcortical one stores – sf polygons for the 2D slab views and a mesh per structure for 3D – so this is the subcortical shape under a name of its own, and the object inherits `ggseg_data_subcortical` so the methods written for it apply unchanged.

Usage

```
ggseg_data_wm(geom = NULL, meshes = NULL, ...)
```

Arguments

`geom`	An sf data frame of polygons, or `NULL`.
`meshes`	A data frame of per-structure meshes, or `NULL`.
`...`	Passed to the shared geometry resolver.

Details

White-matter *tracts* are a different thing: `ggseg_data_tract()` also carries centerlines, which a white-matter label map has none of.

Value

A `ggseg_data_wm` object.

See Also

`ggseg_data_subcortical()`, `ggseg_data_tract()`

Examples

```
ggseg_data_wm(
  meshes = data.frame(
    label = "cst_left",
    mesh = I(list(list(
      vertices = data.frame(x = 1:10, y = 1:10, z = 1:10),
      faces = data.frame(i = 1:3, j = 2:4, k = 3:5)
    )))
  )
)
```

`is_ggseg_atlas` *Check ggseg atlas class*

Description

These functions check both the class tag and structural validity by passing the object through `ggseg_atlas()`. An object that carries the right class but fails validation returns `FALSE`.

Usage

```
is_ggseg_atlas(x)

is_cortical_atlas(x)

is_subcortical_atlas(x)

is_tract_atlas(x)

is_cerebellar_atlas(x)

is_wm_atlas(x)

is_brain_atlas(x)
```

Arguments

x	an object

Value

logical

See Also

`assert_ggseg_atlas()`, which aborts with the reason instead of reducing it to `FALSE`.

Examples

```
is_ggseg_atlas(dk())
is_cortical_atlas(dk())
is_subcortical_atlas(aseg())
is_tract_atlas(tracula())
```

is_ggseg3d_atlas *Check if object is a legacy ggseg3d atlas*

Description

Check if object is a legacy ggseg3d atlas

Usage

```
is_ggseg3d_atlas(x)
```

Arguments

x	an object

Value

logical

Examples

```
is_ggseg3d_atlas(dk())
```

legacy_region_map *Map pre-0.1.0 region names to current region keys*

Description

Before version 0.1.0 the `region` column of the bundled atlases held a long-form display name. From 0.1.0 it holds a short, hemisphere-free key derived from `label`, and the long-form name lives in `display`. Joins and filters written against the old values therefore match nothing and fail silently: `merge(all.x = TRUE)` and the ggseg fill scale both yield `NA`, which draws as a blank parcel rather than raising an error.

Usage

```
legacy_region_map(atlas)
```

Arguments

atlas	a `ggseg_atlas` object

Details

This returns the translation between the two, so old code can be updated mechanically:

```
map <- legacy_region_map(dk())
my_data$region <- unname(map[my_data$region])
```

The mapping comes from a table of exactly what each bundled atlas shipped at 0.0.4, not from the atlas's current `display` column – `display` is the curated display name and several 0.0.4 `region` values were mechanically derived (`"ventraldc"`, `"cc posterior"`), so the two are deliberately not the same. Where two old names shared one `label`, as `"Thalamus"` and `"Thalamus Proper"` did in `aseg`, both still translate even though the alias row itself is gone from `core`.

Value

A named character vector whose names are the pre-0.1.0 `region` values and whose values are the current `region` keys. Zero-length for an atlas this package ships no legacy table for, which is every atlas except the bundled four.

See Also

`atlas_display()` for the curated display names, `atlas_regions()` for the current keys.

Other atlas accessors: `atlas_centerlines()`, `atlas_context()`, `atlas_display()`, `atlas_geom()`, `atlas_geometry_type()`, `atlas_hemi()`, `atlas_labels()`, `atlas_meshes()`, `atlas_palette()`, `atlas_plot_palette()`, `atlas_polygons()`, `atlas_provenance()`, `atlas_region_semantics()`, `atlas_regions()`, `atlas_sf()`, `atlas_type()`, `atlas_undrawn()`, `atlas_vertices()`, `atlas_views()`, `atlas_views_empty()`

Examples

```
head(legacy_region_map(aseg()))
legacy_region_map(tracula())[["SLF I"]]
legacy_region_map(aseg())[["Thalamus Proper"]]
```

---

mesh_space *Record what a mesh's coordinates mean*

Description

A surface mesh on its own is just numbers: nothing in `vertices` says which hemisphere it is, which vertex space an atlas must have been built against to index it, or how to get from its coordinates to a standard space. Without that, consumers infer those things, and a mesh inferred wrongly still renders perfectly – a brain mirrored left-to-right, or regions painted at the wrong vertices, looks like a brain. `mesh_space()` makes the mesh state them.

Usage

```
mesh_space(
  vertex_space,
  hemi,
  anatomical,
  to_mni305 = NULL,
  units = "mm",
  surface = NULL,
  source = NULL
)
```

Arguments

vertex_space	Vertex space the mesh is in, such as "fsaverage5". Atlas vertex indices are only meaningful against a named space.
hemi	"left", "right" or "midline".
anatomical	Whether the coordinates correspond to positions in a brain. FALSE for inflated and spherical surfaces.
to_mni305	A 4x4 matrix mapping vertex coordinates to MNI305, or NULL when anatomical is FALSE.
units	Coordinate units. Millimetres throughout the ecosystem.
surface	Optional surface name, such as "pial".
source	Optional free-text note on where the mesh came from, for reproducibility. Never read by code; atlas_provenance() remains the authoritative record.

Details

vertex_space is the mesh's identity as far as atlases are concerned. Atlases key regions to vertex indices, so an atlas built against one vertex space and drawn on another puts every region somewhere else. Checking that indices merely fit catches only the case where the other mesh is smaller.

Name a *numbered* space – "fsaverage5", "fsaverage6" – because those are fixed templates and stable across FreeSurfer releases: fsaverage5 is vertex-for-vertex the same mesh in 7.4.1 and 8.2.0, which is what makes an atlas's indices portable. Unnumbered "fsaverage" is not a stable identifier: it is whichever densest mesh that release ships, so the same string can mean different vertex counts and different indices. Record source if you must use it.

hemi is recorded because left and right are the one distinction this ecosystem must never get wrong, and a mesh that cannot state its own side cannot be checked against the data drawn on it.

to_mni305 is a 4x4 affine mapping the mesh's own vertex coordinates to MNI305, collapsing any axis permutation, sign flip and translation into one matrix. It is required when anatomical is TRUE and must be NULL otherwise: inflation and spherical registration move every vertex, so those surfaces have no anatomical position to map to and a matrix for them would be a number that looks usable.

Positioning a mesh for display – moving a hemisphere to meet the midline, say – is the renderer's decision, not a property of the mesh, so it is deliberately not recorded here.

Value

A mesh_space object, to be attached with set_mesh_space().

See Also

Other mesh spaces: set_mesh_space()

Examples

```
mesh_space(
  vertex_space = "fsaverage5",
  hemi = "left",
  anatomical = FALSE
)
```

---

migrate_atlas_display *Give an atlas built before* display *existed its display names*

Description

display holds the source's own name for a region, and an atlas built before the column existed looks as though that name is lost. It is not: label is the source name with a hemisphere affix on it, so the display name is recoverable exactly. Verified against every atlas that already carries the column – 1996 rows, no disagreement.

Usage

```
migrate_atlas_display(atlas, overwrite = FALSE)
```

Arguments

atlas	A ggseg_atlas.
overwrite	Replace an existing display column. Off by default, because an atlas that carries one got it from its source, and the source's own name beats one derived from a key.

Details

Atlases that already carry display are returned unchanged, so this is safe to run over a whole library.

Value

atlas, with display in core.

Examples

```
atlas <- dk()
atlas$core$display <- NULL
atlas_display(migrate_atlas_display(atlas))[1:3]
```

`migrate_atlas_files` *Migrate atlas* `.rda` *files to the sf-optional polygon format*

Description

Walks a directory of `.rda` files, finds every `ggseg_atlas` object inside them, and rewrites their 2D geometry into the single `geom` slot. By default the geometry is stored as `brain_polygons` (sf-optional); any legacy `sf` / `polygons` slots are dropped. Pass `keep_sf = TRUE` to store the geometry as sf instead.

Usage

```
migrate_atlas_files(
  path = "data",
  keep_sf = FALSE,
  quiet = FALSE,
  force = FALSE
)
```

Arguments

`path`	Directory containing `.rda` files to migrate. Defaults to `"data"`, the conventional location in R packages.
`keep_sf`	If `TRUE`, the geometry is stored in `geom` as sf. Default `FALSE` — the geometry is stored as `brain_polygons` (sf-optional).
`quiet`	If `TRUE`, suppress per-file status messages.
`force`	If `TRUE`, migrate even when the conversion cannot carry every geometry column across. The default `FALSE` aborts instead, naming the columns and the file, because the files are rewritten in place.

Details

Intended for downstream atlas-package maintainers across the ggsegverse ecosystem: run once against your `data/` directory, then drop `sf` from DESCRIPTION Imports.

Value

Invisibly, a character vector of paths to the files that were rewritten.

Examples

```
# In an atlas package you would call this on the package's own data/
# directory. Here it runs against a throwaway copy, since it rewrites the
# files it is pointed at. Producing an sf atlas to migrate needs sf.
dir <- file.path(tempdir(), "data")
dir.create(dir, showWarnings = FALSE)
dk_atlas <- as_sf_atlas(dk())
save(dk_atlas, file = file.path(dir, "dk_atlas.rda"))

migrate_atlas_files(dir, quiet = TRUE)

load(file.path(dir, "dk_atlas.rda"))
is_atlas_polygon(dk_atlas) # TRUE
unlink(dir, recursive = TRUE)
```

---

migrate_atlas_hemi *Map a legacy* hemi *spelling onto the vocabulary*

Description

hemi is a closed vocabulary – "left", "right", "midline" or NA – and two spellings predate it. "middle" was an earlier name for the midline, and "vermis" is a cerebellar structure rather than a side, so both map onto "midline". The finer distinction belongs in a column of its own.

Usage

```
migrate_atlas_hemi(atlas)
```

Arguments

atlas	A ggseg_atlas, or a list shaped like one that no longer validates.

Details

An atlas carrying one cannot be constructed, so it cannot be passed through any verb that validates – including migrate_atlas_display(). This repairs core directly for that reason.

Value

atlas, with hemi in the vocabulary.

Examples

```
atlas <- dk()
atlas$core$hemi[1] <- "middle"
unique(migrate_atlas_hemi(atlas)$core$hemi)
```

read_atlas_files *Read in atlas data from all subjects*

Description

Recursively reads in all stats files for an atlas (given a unique character string), for all subjects in the subjects directory. Will add hemisphere and subject id to the data.

Usage

```
read_atlas_files(subjects_dir, atlas)
```

Arguments

subjects_dir	FreeSurfer subject directory
atlas	unique character combination identifying the atlas

Value

data.frame with stats information for subjects from FreeSurfer

Examples

```
subj_dir <- "/path/to/freesurfer/7.2.0/subjects/"
read_atlas_files(subj_dir, "aseg.stats")
read_atlas_files(subj_dir, "lh.aparc.stats")
```

read_freesurfer_stats *Read in raw FreeSurfer stats file*

Description

FreeSurfer atlas stats files have a format that can be difficult to easily read in to R. This function takes a raw stats-file from the subjects directory and reads it in as a data.frame.

Usage

```
read_freesurfer_stats(path, rename = TRUE)
```

Arguments

path	path to stats file
rename	logical. rename headers for ggseg compatibility

Value

data.frame with stats information for subjects from FreeSurfer

Examples

```
subj_dir <- "/path/to/freesurfer/7.2.0/subjects/"
aseg_stats <- file.path(subj_dir, "bert/stats/aseg.stats")
read_freesurfer_stats(aseg_stats)
```

---

read_freesurfer_table *Read in stats table from FreeSurfer*

Description

FreeSurfer has functions to create tables from raw stats files. If you have data already merged using the `aparcstats2table` or `asegstats2table` from FreeSurfer, this function will read in the data and prepare it for ggseg.

Usage

```
read_freesurfer_table(path, measure = NULL, ...)
```

Arguments

`path`	path to the table file
`measure`	which measure is the table of
`...`	additional arguments to `read.table`

Value

data.frame with stats information for subjects from FreeSurfer

Examples

```
file_path <- "all_subj_aseg.txt"
read_freesurfer_table(file_path)
```

`region_manipulation` *Change which regions an atlas has*

Description

Verbs that modify the regions `core` declares, and the geometry that does or does not belong to them:

Usage

```
atlas_region_remove(atlas, pattern, match_on = c("label", "region"))

atlas_region_contextual(
  atlas,
  pattern,
  match_on = c("label", "region"),
  ignore.case = TRUE
)

atlas_region_op(
  atlas,
  x,
  y,
  action = c("difference", "intersection", "union", "symdifference"),
  into = NULL,
  match_on = c("label", "region"),
  colour = NULL
)

atlas_context_remove(atlas)

atlas_region_rename(
  atlas,
  pattern,
  replacement,
  match_on = c("label", "region")
)

atlas_region_keep(atlas, pattern, match_on = c("label", "region"))
```

Arguments

`atlas`	A `ggseg_atlas` object
`pattern`	Character pattern to match. Uses `grepl(..., ignore.case = TRUE)`.
`match_on`	Column of `core` to match against: `"label"`, the default, or `"region"`. `label` is the identifier that is unique within an atlas and stable across them, so every verb that selects rows by name or pattern reads it unless told otherwise.

ignore.case	For atlas_region_contextual(): passed to grepl(). Defaults to TRUE for backwards compatibility, but note that a context pattern like "Thalamus" then also matches focus labels such as "hypothalamus"; set FALSE (and/or anchor the pattern) when that matters.
x, y	For atlas_region_op(): patterns selecting the two operands, or a selector such as context() or regions().
action	For atlas_region_op(): the boolean operation to apply.
into	For atlas_region_op(): label for the result region.
colour	For atlas_region_op(): optional fill for into. When supplied, into is registered in core and palette; when NULL, the result is contextual geometry only.
replacement	For atlas_region_rename(): replacement string or function.

Details

- atlas_region_remove(): completely remove regions
- atlas_region_keep(): keep only matching regions
- atlas_region_rename(): rename regions in core
- atlas_region_contextual(): keep geometry but remove from core/palette
- atlas_context_remove(): drop all contextual geometry
- atlas_region_op(): combine two regions' geometry with a boolean op (difference / intersection / union / symdifference)

To drop a region's drawing in some views while keeping the region itself, see drawing_manipulation.

Value

Modified ggseg_atlas object

Functions

- atlas_region_contextual(): Keep geometry for visual context but remove from core, palette, and 3D data. Context geometries render grey and don't appear in legends. Contextual rows are moved behind the remaining core regions so focus regions draw on top where they overlap in projection. Operates on whichever 2D representation the atlas carries (sf and/or polygons), keeping both in sync, and needs no sf for a polygon atlas.
- atlas_region_op(): Combine two sets of region geometry with a vector boolean operation (per view), writing the result to a new region into. x and y are patterns matched against match_on; within each view their matching geometries are unioned, then combined via action: "difference" (x minus y, e.g. punching white matter out of a cortex silhouette), "intersection", "union", or "symdifference". Inputs are left in place; any existing into geometry is replaced. With a colour, into becomes a legended core region; otherwise it stays contextual (grey) and is drawn behind the core regions. Boolean ops need a geometry engine, so this is the one manipulation helper that always requires sf installed; a polygon-only atlas is rehydrated for the operation and the result is returned in polygon form.

- `atlas_context_remove()`: Drop all contextual sf geometry — every sf row whose `label` is not present in `core`. Covers labels marked via `atlas_region_contextual()` plus pipeline-generated outlines (`cortex_`, `Background`, `unknown`). Remaining views are re-packed via `atlas_view_gather()` so the plot focuses tightly on the labelled regions.
- `atlas_region_rename()`: Rename regions matching a pattern. Only ever writes to the `region` column, never to `label`. `match_on` chooses the rows: those whose `label` (the default) or `region` matches `pattern`. The edit is always made to the region text those rows already have. A string `replacement` is substituted for `pattern` wherever the region contains it, so `atlas_region_rename(atlas, "Left-", "")` strips the prefix from regions that still carry it. A function `replacement` receives the regions of the matching rows and returns the new ones, which is how to set a region outright for a label: `atlas_region_rename(atlas, "^Brain-Stem$", \(region) "brai` A string that changes nothing, because the pattern is in the labels but not in their regions, is reported with a warning.
- `atlas_region_keep()`: Keep only matching regions. Non-matching regions are removed from core, palette, and 3D data but sf geometry is preserved for surface continuity.

See Also

atlas_manipulation for the other groups

Other atlas manipulations: `atlas_core_add()`, `atlas_manipulation`, `atlas_relabel()`, `atlas_structure_reorder()`, `drawing_manipulation`, `view_manipulation`

Examples

```
# Regions are named by the atlas, so take them from it rather than
# spelling them out here, where they would go stale unnoticed.
first_region <- atlas_regions(dk())[1]
dk() |> atlas_region_remove(first_region, match_on = "region")
```

`reorder_sfc` *Reorder an* `sfc` *without sf*

Description

Subsetting drops the `sfc` class without sf loaded. A permutation does not change the geometry set, so `bbox`, `crs`, `precision` and `n_empty` stay valid and the attributes are reattached unchanged.

Usage

```
reorder_sfc(geometry, i)
```

Arguments

`geometry`	an `sfc`.
`i`	row positions, in the order wanted.

Details

`i` must be a permutation or a repetition of row positions, not a subset: a selection that drops geometries leaves `bbox` and `n_empty` describing rows that are no longer present.

Value

`geometry` in the order given by `i`, still an `sfc`.

See Also

`attach_sfc()` to put the result back on a data frame.

Other sf-optional helpers: `attach_sfc()`

Examples

```
geometry <- atlas_sf(dk())$geometry
reversed <- reorder_sfc(geometry, rev(seq_along(geometry)))
inherits(reversed, "sfc")
```

---

selectors *Select an atlas's backdrop, or its regions, in a geometry verb*

Description

The geometry verbs take `labels` and `exclude` as patterns, and a backdrop has to be addressed separately from the parcels: what makes a cortical silhouette read as a brain is its sulci, which are fine detail, so it wants gentler simplification and a smoothing method that does not fill them.

Usage

```
context()

regions()
```

Details

Build scripts had been writing the pattern by hand, as `"^cortex"`. That is a guess about what the pipeline named the backdrop, and it was wrong twice over: the name varies across the library – `cortex_`, `cortex`, `lh_unknown`, `lh_Background+FreeSurfer_Defined_Medial_Wall` – and a pattern matching nothing is silent, so those steps did nothing at all for as long as a pipeline bug left the backdrop undrawn.

These ask the atlas instead. The lookup table decides what is backdrop when the atlas is built; this is how that decision is read back.

Value

A function of an atlas returning a pattern, which `labels` and `exclude` accept.

See Also

atlas_context()

Examples

```
# gentler on the backdrop than on the parcels
## Not run:
atlas |>
  atlas_simplify(keep = 0.5, labels = context()) |>
  atlas_simplify(keep = 0.25, exclude = context())

## End(Not run)
```

---

set_atlas_palette *Set the palette of an atlas*

Description

Replaces the colour palette of a brain atlas, validating that `value` is a named character vector and warning if it does not cover every atlas label. This is the setter counterpart to the `atlas_palette()` accessor; it returns the modified atlas, so it composes with the pipe.

Usage

```
set_atlas_palette(atlas, value)
```

Arguments

`atlas`	a `ggseg_atlas` object
`value`	Named character vector of colours keyed by atlas `label`.

Value

The `ggseg_atlas` with its palette replaced.

See Also

`atlas_palette()` to read the palette.

Other atlas setters: `set_atlas_type()`

Examples

```
a <- aseg()
labs <- atlas_labels(a)
a <- set_atlas_palette(a, setNames(grDevices::rainbow(length(labs)), labs))
```

set_atlas_provenance *Record how an atlas was built*

Description

Attaches a build record to an atlas. Intended for the pipelines in **ggseg.extra**, which know the arguments a build was given; it is exported so a third party constructing an atlas by hand can record the same thing rather than leaving it unanswerable.

Usage

```
set_atlas_provenance(atlas, value)
```

Arguments

atlas	a ggseg_atlas object
value	A named list, or NULL to drop the record. Must contain pipeline, a single string naming what built the atlas and its version, and created, a single string or Date for when. Further fields are yours.

Value

atlas, with value as its provenance.

See Also

atlas_provenance() to read one back.

Examples

```
atlas <- set_atlas_provenance(
  dk(),
  list(pipeline = "hand-built 1.0", created = "2026-10-29")
)
atlas_provenance(atlas)$pipeline
```

set_atlas_type *Set the type of an atlas*

Description

Replaces the type of a brain atlas. The type is held in three coupled places: the type field, the leading <type>_atlas class, and the ggseg_data_<type> class of the data payload. This setter reconstructs the atlas through ggseg_atlas() so all three stay in agreement – assigning the type field directly leaves the subclass stale and is_tract_atlas() and friends disagreeing with atlas_type().

Usage

```
set_atlas_type(atlas, value)
```

Arguments

atlas	a ggseg_atlas object
value	Atlas type; one of "cortical", "subcortical", "tract" or "cerebellar".

Details

Because type and payload are coupled, the new type must match the data the atlas already carries: a "tract" atlas needs ggseg_data_tract() (centerlines), a "subcortical" atlas needs ggseg_data_subcortical() (meshes). Retyping an atlas whose payload does not match is an error – rebuild the payload with the matching ggseg_data_*() constructor first.

Value

The ggseg_atlas with its type, subclass and payload class in agreement.

See Also

atlas_type() to read the type.

Other atlas setters: set_atlas_palette()

Examples

```
a <- aseg()
a <- set_atlas_type(a, "subcortical")
atlas_type(a)
```

---

set_mesh_space *Attach or read a mesh's coordinate record*

Description

Attach or read a mesh's coordinate record

Usage

```
set_mesh_space(mesh, space)

get_mesh_space(mesh)
```

Arguments

mesh	A list with vertices and faces.
space	A mesh_space() object.

Value

`set_mesh_space()` returns `mesh` with the record attached. `get_mesh_space()` returns the record, or `NULL` when the mesh carries none – meshes from outside the core packages legitimately do not.

See Also

Other mesh spaces: `mesh_space()`

Examples

```
mesh <- get_brain_mesh("lh")
space <- mesh_space("fsaverage5", "left", anatomical = FALSE)
mesh <- set_mesh_space(mesh, space)
get_mesh_space(mesh)$vertex_space
```

---

suit *SUIT Cerebellar Lobular Atlas*

Description

Returns the SUIT cerebellar parcellation (Diedrichsen et al., 2009): the cerebellar cortex split into anatomical lobules plus the deep nuclei (dentate, interposed, fastigial).

Usage

```
suit()
```

Details

Surface lobules carry vertex indices into the shared SUIT cerebellar mesh (see `get_cerebellar_mesh()`); deep nuclei carry individual 3D meshes. The 2D geometry is stored in the sf-optional polygon (`geom`) representation, so the atlas renders with ggseg without requiring sf installed.

Value

A `ggseg_atlas` object with components:

atlas Character. Atlas name ("suit")

type Character. Atlas type ("cerebellar")

palette Named character vector of colours for each region

data A `ggseg_data_cerebellar` object containing:

- **geom** A `brain_polygons` table for 2D rendering
- **vertices** Vertex indices for surface lobules
- **meshes** Per-structure 3D meshes for the deep nuclei

Core columns

`core` carries one row per atlas region with the columns `hemi`, `region`, `label` and `names`, plus the atlas's own grouping column. `label` is the stable atlas identifier and the key the palette and the geometry are keyed on; `region` is a short, hemisphere-free key derived from it; `names` is the curated long-form display name, spelled out to read well in a figure legend.

`region` held the long-form name until version 0.1.0. `names` is not a record of those old values – use `legacy_region_map()` to translate them.

References

Diedrichsen J, Balsters JH, Flavell J, et al. (2009). A probabilistic MR atlas of the human cerebellum. NeuroImage, 46(1):39-46. doi:10.1016/j.neuroimage.2009.01.045

See Also

`dk()` for cortical parcellation, `aseg()` for subcortical structures, `tracula()` for white-matter tracts, `ggseg_atlas()` for the atlas class constructor

Other ggseg_atlases: `aseg()`, `dk()`, `tracula()`

Examples

```
suit()
atlas_regions(suit())
atlas_geometry_type(suit())
```

---

tracula *TRACULA White Matter Tract Atlas*

Description

Returns the TRACULA (TRActs Constrained by UnderLying Anatomy) white matter bundle atlas in MNI space.

Usage

```
tracula()
```

Details

This atlas contains major white matter tracts reconstructed from diffusion MRI using FreeSurfer's TRACULA training data. It works with both ggseg (2D slice projections) and ggseg3d (3D tube mesh visualizations).

Value

A ggseg_atlas object with components:

atlas Character. Atlas name ("tracula")

type Character. Atlas type ("tract")

palette Named character vector of colours for each tract

data A ggseg_data_tract object containing:

centerlines List of centerline matrices per tract

sf Simple features data frame for 2D rendering

Core columns

core carries one row per atlas region with the columns hemi, region, label and names, plus the atlas's own grouping column. label is the stable atlas identifier and the key the palette and the geometry are keyed on; region is a short, hemisphere-free key derived from it; names is the curated long-form display name, spelled out to read well in a figure legend.

region held the long-form name until version 0.1.0. names is not a record of those old values – use legacy_region_map() to translate them.

Tract labels

label is the name FreeSurfer's TRACULA writes its per-pathway outputs under, including the trailing .bbr.prep (for example lh.af.bbr.prep), so a table of TRACULA output joins to the atlas as it comes. label_short is the same identifier with the suffix stripped – the name FreeSurferColorLUT.txt gives the pathway at ids 5100-5399.

References

Yendiki A, Panneck P, Srinivasan P, et al. (2011). Automated probabilistic reconstruction of white-matter pathways in health and disease using an atlas of the underlying anatomy. Frontiers in Neuroinformatics, 5:23. doi:10.3389/fninf.2011.00023

See Also

dk() for cortical parcellation, aseg() for subcortical structures, ggseg_atlas() for the atlas class constructor

Other ggseg_atlases: aseg(), dk(), suit()

Examples

```
tracula()
plot(tracula())
atlas_regions(tracula())
```

`view_manipulation` *Change which views an atlas has, and where they sit*

Description

Verbs that act on views as objects:

Usage

```
atlas_view_remove(atlas, views, match = c("exact", "pattern"))

atlas_view_keep(atlas, views, match = c("exact", "pattern"))

atlas_view_gather(atlas, gap = 0.15)

atlas_view_reorder(atlas, order, gap = 0.15)
```

Arguments

`atlas`	A `ggseg_atlas` object
`views`	View names to act on. Matched whole and case-insensitively against the atlas's own views, so a name has to be one `atlas_views()` reports. Give them as a character vector, or in one string separated by `"|"`; the two are equivalent. Names that match no view are warned about and otherwise ignored.
`match`	How `views` is read. `"exact"`, the default, takes each name as a whole view name. `"pattern"` takes `views` as a regular expression instead, matched as a substring — `"axial"` then reaches `axial_1` and `axial_2` as well. Prefer `"exact"`: a name that happens to sit inside another view's name silently takes both views with it under `"pattern"`.
`gap`	Proportional gap between views (default 0.15 = 15% of max width).
`order`	For `atlas_view_reorder()`: character vector of desired view order. Unspecified views appended at end.

Details

- `atlas_view_remove()`: remove entire views
- `atlas_view_keep()`: keep only matching views
- `atlas_view_reorder()`: change view order
- `atlas_view_gather()`: reposition views to close gaps

None of these changes what is drawn inside a view. For that — dropping a region's depiction in some views while the region itself stays — see drawing_manipulation.

Value

Modified `ggseg_atlas` object

Functions

- `atlas_view_keep()`: Keep only views matching pattern. Remaining views are re-packed via `atlas_view_gather()` so the layout stays tight, as `atlas_view_remove()` does.
- `atlas_view_gather()`: Reposition remaining views to close gaps after view removal.
- `atlas_view_reorder()`: Reorder views and reposition. Views not in `order` are appended at end.

See Also

atlas_manipulation for the other groups

Other atlas manipulations: `atlas_core_add()`, `atlas_manipulation`, `atlas_relabel()`, `atlas_structure_reorder()`, `drawing_manipulation`, `region_manipulation`

Examples

```
# Views are named by the atlas, so ask it rather than naming one here.
dk() |> atlas_view_keep(atlas_views(dk())[1])
```

Index

∗ **atlas accessors**
atlas_centerlines, 8
atlas_context, 9
atlas_display, 10
atlas_geom, 11
atlas_geometry_type, 12
atlas_hemi, 13
atlas_labels, 14
atlas_meshes, 15
atlas_palette, 16
atlas_plot_palette, 17
atlas_polygons, 18
atlas_provenance, 18
atlas_region_semantics, 19
atlas_regions, 20
atlas_sf, 22
atlas_type, 24
atlas_undrawn, 25
atlas_vertices, 26
atlas_views, 26
atlas_views_empty, 27
legacy_region_map, 45
∗ **atlas manipulations**
atlas_core_add, 10
atlas_manipulation, 15
atlas_relabel, 21
atlas_structure_reorder, 23
drawing_manipulation, 32
region_manipulation, 53
view_manipulation, 63
∗ **atlas setters**
set_atlas_palette, 57
set_atlas_type, 58
∗ **cerebellar_atlases**
suit, 60
∗ **cortical_atlases**
dk, 31
∗ **ggseg_atlases**
aseg, 5
dk, 31
suit, 60
tracula, 61
∗ **mesh spaces**
mesh_space, 46
set_mesh_space, 59
∗ **sf-optional helpers**
attach_sfc, 28
reorder_sfc, 55
∗ **subcortical_atlases**
aseg, 5
∗ **tract_atlases**
tracula, 61

as_brain_atlas *(*as_ggseg_atlas*)*, 3
as_ggseg_atlas, 3
as_polygon_atlas, 4
as_polygon_atlas(), *5*
as_sf_atlas, 5
as_sf_atlas(), *4*
aseg, 5
aseg(), *32*, *35*, *61*, *62*
assert_ggseg_atlas, 7
assert_ggseg_atlas(), *44*
atlas_centerlines, 8
atlas_centerlines(), *9*, *11–14*, *16–22*, *24–28*, *46*
atlas_context, 9
atlas_context(), *8*, *11–14*, *16–22*, *24–28*, *31*, *46*, *57*
atlas_context_remove *(*region_manipulation*)*, 53
atlas_core_add, 10
atlas_core_add(), *15*, *22*, *24*, *35*, *55*, *64*
atlas_display, 10
atlas_display(), *8, 9*, *12–14*, *16–22*, *24–28*, *31*, *46*
atlas_geom, 11
atlas_geom(), *8, 9*, *11*, *13, 14*, *16–22*, *24–28*, *46*

atlas_geometry_type, 12
atlas_geometry_type(), *8, 9*, *11–14*, *16–22*, *24–28*, *46*
atlas_hemi, 13
atlas_hemi(), *8, 9*, *11–14*, *16–22*, *24–28*, *46*
atlas_labels, 14
atlas_labels(), *8, 9*, *11–13*, *16–22*, *24–28*, *46*
atlas_manipulation, *10*, 15, *22*, *24*, *35*, *55*, *64*
atlas_meshes, 15
atlas_meshes(), *8, 9*, *11–14*, *16–22*, *24–28*, *46*
atlas_palette, 16
atlas_palette(), *8, 9*, *11–14*, *16–22*, *24–28*, *46*, *57*
atlas_plot_palette, 17
atlas_plot_palette(), *8, 9*, *11–14*, *16*, *18–22*, *24–28*, *46*
atlas_polygons, 18
atlas_polygons(), *8, 9*, *11–14*, *16, 17*, *19–22*, *24–28*, *46*
atlas_provenance, 18
atlas_provenance(), *8, 9*, *11–14*, *16–18*, *20–22*, *24–28*, *46, 47*, *58*
atlas_prune_region *(*drawing_manipulation*)*, 32
atlas_prune_small *(*drawing_manipulation*)*, 32
atlas_prune_weak *(*drawing_manipulation*)*, 32
atlas_region_contextual *(*region_manipulation*)*, 53
atlas_region_contextual(), *55*
atlas_region_keep *(*region_manipulation*)*, 53
atlas_region_op *(*region_manipulation*)*, 53
atlas_region_remove *(*region_manipulation*)*, 53
atlas_region_rename *(*region_manipulation*)*, 53
atlas_region_rename(), *21, 22*
atlas_region_semantics, 19
atlas_region_semantics(), *8, 9*, *11–14*, *16–19*, *21, 22*, *24–28*, *31*, *46*
atlas_regions, 20
atlas_regions(), *8, 9*, *11–14*, *16–20*, *22*, *24–28*, *46*
atlas_relabel, 21
atlas_relabel(), *10*, *15*, *24*, *35*, *55*, *64*
atlas_sf, 22
atlas_sf(), *8, 9*, *11–14*, *16–21*, *24–28*, *46*
atlas_structure_reorder, 23
atlas_structure_reorder(), *10*, *15*, *22*, *35*, *55*, *64*
atlas_type, 24
atlas_type(), *8, 9*, *11–14*, *16–22*, *25–28*, *46*, *58, 59*
atlas_undrawn, 25
atlas_undrawn(), *8, 9*, *11–14*, *16–22*, *24*, *26–28*, *34, 35*, *46*
atlas_vertices, 26
atlas_vertices(), *8, 9*, *11–14*, *16–22*, *24, 25*, *27, 28*, *46*
atlas_view_gather *(*view_manipulation*)*, 63
atlas_view_gather(), *35*, *55*, *64*
atlas_view_keep *(*view_manipulation*)*, 63
atlas_view_keep(), *25*, *35*
atlas_view_remove *(*view_manipulation*)*, 63
atlas_view_remove(), *25*, *27*, *35*, *64*
atlas_view_remove_region *(*drawing_manipulation*)*, 32
atlas_view_remove_small *(*drawing_manipulation*)*, 32
atlas_view_reorder *(*view_manipulation*)*, 63
atlas_view_reorder(), *24*
atlas_view_select *(*drawing_manipulation*)*, 32
atlas_views, 26
atlas_views(), *8, 9*, *11–14*, *16–22*, *24–26*, *28*, *34*, *46*, *63*
atlas_views_empty, 27
atlas_views_empty(), *8, 9*, *11–14*, *16–22*, *24–27*, *46*
attach_sfc, 28
attach_sfc(), *56*

brain_atlas *(*ggseg_atlas*)*, 37
brain_data_cortical *(*ggseg_data_cortical*)*, 40
brain_data_subcortical *(*ggseg_data_subcortical*)*, 41
brain_data_tract *(*ggseg_data_tract*)*, 42

brain_labels (atlas_labels), 14
brain_regions (atlas_regions), 20
brain_views (atlas_views), 26

context (selectors), 56
context(), *54*
convert_legacy_brain_atlas, 29
core_contract, *19*, 30

dk, 31
dk(), *7*, *61*, *62*
dplyr::relocate(), *23*
drawing_manipulation, *10*, *15*, *22*, *24*, 32, *54*, *55*, *63*, *64*

get_brain_mesh, 36
get_cerebellar_mesh, 37
get_cerebellar_mesh(), *36*, *39*, *60*
get_mesh_space (set_mesh_space), 59
ggseg_atlas, 37
ggseg_atlas(), *7*, *32*, *44*, *58*, *61*, *62*
ggseg_data_cerebellar, 38
ggseg_data_cerebellar(), *38*
ggseg_data_cortical, 40
ggseg_data_cortical(), *38*
ggseg_data_subcortical, 41
ggseg_data_subcortical(), *36*, *38*, *39*, *43*, *59*
ggseg_data_tract, 42
ggseg_data_tract(), *38*, *43*, *59*
ggseg_data_wm, 43
grepl(), *54*

is_atlas_polygon (atlas_geometry_type), 12
is_atlas_sf (atlas_geometry_type), 12
is_brain_atlas (is_ggseg_atlas), 44
is_cerebellar_atlas (is_ggseg_atlas), 44
is_cortical_atlas (is_ggseg_atlas), 44
is_ggseg3d_atlas, 45
is_ggseg_atlas, 44
is_ggseg_atlas(), *7*
is_subcortical_atlas (is_ggseg_atlas), 44
is_tract_atlas (is_ggseg_atlas), 44
is_tract_atlas(), *58*
is_wm_atlas (is_ggseg_atlas), 44

legacy_region_map, 45
legacy_region_map(), *6*, *8–14*, *16–22*, *24–28*, *32*, *61*, *62*

mesh_space, 46
mesh_space(), *59*, *60*
migrate_atlas_display, 48
migrate_atlas_display(), *50*
migrate_atlas_files, 49
migrate_atlas_files(), *4*
migrate_atlas_hemi, 50

read_atlas_files, 51
read_freesurfer_stats, 51
read_freesurfer_table, 52
region_manipulation, *10*, *15*, *22*, *24*, *34*, *35*, 53, *64*
regions (selectors), 56
regions(), *54*
relabel_atlas (atlas_relabel), 21
reorder_sfc, 55
reorder_sfc(), *29*

selectors, 56
set_atlas_palette, 57
set_atlas_palette(), *16*, *59*
set_atlas_provenance, 58
set_atlas_provenance(), *19*, *38*
set_atlas_type, 58
set_atlas_type(), *24*, *57*
set_mesh_space, 59
set_mesh_space(), *48*
sf_to_polygons(), *39–42*
sfheaders::sf_multipolygon(), *5*
suit, 60
suit(), *7*, *32*, *62*

tracula, 61
tracula(), *7*, *32*, *61*

unify_legacy_atlases (convert_legacy_brain_atlas), 29

view_manipulation, *10*, *15*, *22*, *24*, *34*, *35*, *55*, 63