

# Package ‘fsbrain’

June 21, 2020

**Type** Package

**Title** Managing and Visualizing Brain Surface Data

**Version** 0.3.0

**Maintainer** Tim Schäfer <ts+code@rcmd.org>

**Description** Provides high-level access to 'FreeSurfer' <<http://freesurfer.net/>> neuroimaging data on the level of subjects and groups. Load morphometry data, surfaces and brain parcellations based on atlases. Mask data using labels, load data for specific atlas regions only, and visualize data and results directly in 'R'.

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**URL** <https://github.com/dfsp-spirit/fsbrain>

**BugReports** <https://github.com/dfsp-spirit/fsbrain/issues>

**Imports** reshape, freesurferformats (>= 0.1.11), pkgfilecache (>= 0.1.1), rgl, squash, fields, data.table

**Suggests** knitr, rmarkdown, testthat (>= 2.1.0), magick, misc3d, viridis, RColorBrewer

**VignetteBuilder** knitr

**RoxygenNote** 7.1.0.9000

**NeedsCompilation** no

**Author** Tim Schäfer [aut, cre] (<<https://orcid.org/0000-0002-3683-8070>>)

**Repository** CRAN

**Date/Publication** 2020-06-21 04:30:02 UTC

## R topics documented:

alphablend . . . . .	5
annot.outline . . . . .	6
apply.label.to.morphdata . . . . .	7

apply.labeldata.to.morphdata . . . . .	8
apply.transform . . . . .	8
arrange.brainview.images . . . . .	9
brainviews . . . . .	10
clip.data . . . . .	11
cm.cbry . . . . .	12
cm.div . . . . .	12
cm.heat . . . . .	13
cm.qual . . . . .	13
cm.seq . . . . .	14
collayer.bg . . . . .	14
collayer.bg.atlas . . . . .	15
collayer.bg.meancurv . . . . .	16
collayer.bg.sulc . . . . .	17
collayer.from.annot . . . . .	18
collayer.from.annotdata . . . . .	18
collayer.from.mask.data . . . . .	19
collayer.from.morphlike.data . . . . .	20
collayers.merge . . . . .	21
coloredmesh.from.annot . . . . .	21
coloredmesh.from.label . . . . .	22
coloredmesh.from.mask . . . . .	24
coloredmesh.from.morph.native . . . . .	25
coloredmesh.from.morph.standard . . . . .	26
coloredmesh.from.morphdata . . . . .	28
coloredmesh.from.preloaded.data . . . . .	29
coloredmesh.plot.colorbar.separate . . . . .	29
coloredmeshes.from.color . . . . .	31
colorlist.brain.clusters . . . . .	32
combine.colorbar.with.brainview.animation . . . . .	33
combine.colorbar.with.brainview.image . . . . .	34
cube3D.tris . . . . .	35
cubes3D.tris . . . . .	36
delete_all_optional_data . . . . .	37
desaturate . . . . .	37
download_fsaverage . . . . .	38
download_optional_data . . . . .	39
export.coloredmesh.ply . . . . .	39
face.edges . . . . .	40
find.freesurferhome . . . . .	41
find.subjectsdir.of . . . . .	41
fs.coloredmesh . . . . .	42
fs.home . . . . .	43
fsaverage.path . . . . .	43
fup . . . . .	44
get.atlas.region.names . . . . .	44
get.view.angle.names . . . . .	45
getIn . . . . .	46

get_optional_data_filepath . . . . .	47
group.agg.atlas.native . . . . .	47
group.agg.atlas.standard . . . . .	49
group.annot . . . . .	51
group.concat.measures.native . . . . .	52
group.concat.measures.standard . . . . .	53
group.label . . . . .	54
group.label.from.annot . . . . .	55
group.morph.agg.native . . . . .	56
group.morph.agg.standard . . . . .	58
group.morph.agg.standard.vertex . . . . .	60
group.morph.native . . . . .	61
group.morph.standard . . . . .	62
group.multimorph.agg.native . . . . .	64
group.multimorph.agg.standard . . . . .	65
hasIn . . . . .	67
hemilist.derive.hemi . . . . .	68
hemilist.get.combined.data . . . . .	68
hemilist.unwrap . . . . .	69
hemilist.wrap . . . . .	69
is.fs.coloredmesh . . . . .	70
is.fs.coloredvoxels . . . . .	70
is.hemilist . . . . .	71
label.border . . . . .	71
label.colFn . . . . .	72
label.colFn.inv . . . . .	73
label.from.annotdata . . . . .	73
label.to.annot . . . . .	74
labeldata.from.mask . . . . .	75
list_optional_data . . . . .	76
mask.from.labeldata.for.hemi . . . . .	77
mesh.vertex.neighbors . . . . .	78
mkco.cluster . . . . .	79
mkco.div . . . . .	80
mkco.heat . . . . .	80
mkco.seq . . . . .	81
print.fs.coloredmesh . . . . .	81
print.fs.coloredvoxels . . . . .	82
qc.for.group . . . . .	82
qc.from.regionwise.df . . . . .	83
qc.from.segstats.tables . . . . .	84
qc.vis.failcount.by.region . . . . .	84
ras2vox_tkr . . . . .	85
read.colorcsv . . . . .	86
read.md.demographics . . . . .	86
read.md.subjects . . . . .	87
regions.to.ignore . . . . .	88
report.on.demographics . . . . .	89

rglot . . . . .	90
rglvoxels . . . . .	90
shift.hemis.apart . . . . .	91
spread.values.over.annot . . . . .	92
spread.values.over.hemi . . . . .	93
spread.values.over.subject . . . . .	94
subject.annot . . . . .	96
subject.atlas.agg . . . . .	97
subject.filepath.morph.native . . . . .	98
subject.filepath.morph.standard . . . . .	99
subject.label . . . . .	100
subject.label.from.annot . . . . .	101
subject.lobes . . . . .	103
subject.mask . . . . .	104
subject.morph.native . . . . .	105
subject.morph.standard . . . . .	106
subject.surface . . . . .	108
subject.volume . . . . .	109
vdata.split.by.hemi . . . . .	110
vis.color.on.subject . . . . .	111
vis.coloredmeshes . . . . .	112
vis.colortable.legend . . . . .	113
vis.data.on.fsaverage . . . . .	114
vis.data.on.subject . . . . .	116
vis.fs.surface . . . . .	118
vis.labeldata.on.subject . . . . .	119
vis.mask.on.subject . . . . .	121
vis.path.along.verts . . . . .	123
vis.region.values.on.subject . . . . .	124
vis.subject.annot . . . . .	126
vis.subject.label . . . . .	128
vis.subject.morph.native . . . . .	130
vis.subject.morph.standard . . . . .	132
vis.symmetric.data.on.subject . . . . .	134
vislayout.from.coloredmeshes . . . . .	136
vol.boundary.box . . . . .	138
vol.boundary.box.apply . . . . .	139
vol.hull . . . . .	139
vol.imagestack . . . . .	140
vol.intensity.to.color . . . . .	141
vol.mask.from.segmentation . . . . .	142
vol.merge . . . . .	142
vol.overlay.colors.from.activation . . . . .	143
vol.overlay.colors.from.colortable . . . . .	144
vol.planes . . . . .	145
vol.slice . . . . .	145
vol.vox.from.crs . . . . .	146
volvis.contour . . . . .	147

volvis.lightbox . . . . .	148
volvis.voxels . . . . .	149
vox2ras_tkr . . . . .	150
write.region.aggregated . . . . .	151
write.region.values . . . . .	152
write.region.values.fsaverage . . . . .	153

<b>Index</b>	<b>155</b>
--------------	------------

---

alphablend	<i>Perform alpha blending for pairs of RGBA colors.</i>
------------	---

---

### Description

Implements the *\*over\** alpha blending operation.

### Usage

```
alphablend(front_color, back_color, silent = TRUE)
```

### Arguments

front_color	rgba color strings, the upper color layer or foreground
back_color	rgba color strings, the lower color layer or background
silent	logical, whether to suppress messages

### Value

rgba color strings, the alpha-blended colors

### References

see the *\*Alpha blending\** section on [https://en.wikipedia.org/wiki/Alpha\\_compositing](https://en.wikipedia.org/wiki/Alpha_compositing)

### See Also

Other color functions: [desaturate\(\)](#)

---

annot.outline                      *Compute outline vertex colors from annotation.*

---

### Description

For each region in an atlas, compute the outer border and color the respective vertices in the region-specific color from the annot's colortable.

### Usage

```
annot.outline(
    annotdata,
    surface_mesh,
    background = "white",
    silent = TRUE,
    expand_inwards = 0L,
    outline_color = NULL,
    limit_to_regions = NULL
)
```

### Arguments

annotdata	an annotation, as returned by functions like <a href="#">subject.annot</a> .
surface_mesh	brain surface mesh, as returned by functions like <a href="#">subject.surface</a> or <a href="#">read.fs.surface</a> .
background	color, the background color to assign to the non-border parts of the regions. Defaults to 'white'.
silent	logical, whether to suppress status messages.
expand_inwards	integer, additional thickness of the borders. Increases computation time, defaults to 0L.
outline_color	NULL or a color string (like 'black' or '#000000'), the color to use for the borders. If left at the default value 'NULL', the colors from the annotation color lookup table will be used.
limit_to_regions	vector of character strings or NULL, a list of regions for which to draw the outline (see <a href="#">get.atlas.region.names</a> ). If NULL, all regions will be used. If (and only if) this parameter is used, the 'outline_color' parameter can be a vector of color strings, one color per region.

### Value

vector of colors, one color for each mesh vertex

### Note

Sorry for the computational time, the mesh datastructure is not ideal for neighborhood search.

---

`apply.label.to.morphdata`*Load a label from file and apply it to morphometry data.*

---

### Description

This function will set all values in morphdata which are *\*not\** part of the label loaded from the file to NA (or whatever is specified by 'masked\_data\_value'). This is typically used to ignore values which are not part of the cortex (or any other label) during your analysis.

### Usage

```
apply.label.to.morphdata(  
  morphdata,  
  subjects_dir,  
  subject_id,  
  hemi,  
  label,  
  masked_data_value = NA  
)
```

### Arguments

morphdata	numerical vector, the morphometry data for one hemisphere
subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier
hemi	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
label	string, 'fs.label' instance, or label vertex data. If a string, interpreted as the file name of the label file, without the hemi part (if any), optionally including the '.label' suffix. E.g., 'cortex.label' or 'cortex' for '?h.cortex.label'.
masked_data_value	numerical, the value to set for all morphometry data values of vertices which are <i>*not*</i> part of the label. Defaults to NA.

### Value

numerical vector, the masked data.

### See Also

Other label functions: [apply.labeldata.to.morphdata\(\)](#), [subject.lobes\(\)](#), [subject.mask\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.subject.label\(\)](#)

Other morphometry data functions: [apply.labeldata.to.morphdata\(\)](#), [group.morph.native\(\)](#), [group.morph.standard\(\)](#), [subject.morph.native\(\)](#), [subject.morph.standard\(\)](#)

---

apply.labeldata.to.morphdata

*Apply a label to morphometry data.*

---

### Description

This function will set all values in morphdata which are *\*not\** part of the labeldata to NA (or whatever is specified by 'masked\_data\_value'). This is typically used to ignore values which are not part of the cortex (or any other label) during your analysis.

### Usage

```
apply.labeldata.to.morphdata(morphdata, labeldata, masked_data_value = NA)
```

### Arguments

morphdata        numerical vector, the morphometry data for one hemisphere  
 labeldata        integer vector or 'fs.label' instance. A label as returned by [subject.label](#).  
 masked\_data\_value        numerical, the value to set for all morphometry data values of vertices which are *\*not\** part of the label. Defaults to NA.

### Value

numerical vector, the masked data.

### See Also

Other label functions: [apply.label.to.morphdata\(\)](#), [subject.lobes\(\)](#), [subject.mask\(\)](#), [vis.labeldata.on.subject.vis.subject.label\(\)](#)

Other morphometry data functions: [apply.label.to.morphdata\(\)](#), [group.morph.native\(\)](#), [group.morph.standard\(\)](#), [subject.morph.native\(\)](#), [subject.morph.standard\(\)](#)

---

apply.transform

*Apply matmult transformation to input.*

---

### Description

Apply affine transformation, like a *\*vox2ras\_tkr\** transformation, to input. This is just matrix multiplication for different input objects.

### Usage

```
apply.transform(object, matrix_fun)
```



### Arguments

object	numerical vector/matrix or Triangles3D instance, the coordinates or object to transform.
matrix_fun	a 4x4 affine matrix or a function returning such a matrix. If 'NULL', the input is returned as-is. In many cases you may want to use a matrix computed from the header of a volume file, e.g., the 'vox2ras' matrix of the respective volume. See the 'mghheader.*' functions in the *freesurferformats* package to obtain these matrices.

### Value

the input after application of the affine matrix (matrix multiplication)

---

arrange.brainview.images

*Combine several brainview images into a new figure.*

---

### Description

Combine several brainview images into a new figure.

### Usage

```
arrange.brainview.images(
  brainview_images,
  output_img,
  colorbar_img = NULL,
  silent = FALSE,
  grid_like = TRUE,
  border_geometry = "5x5",
  background_color = "white"
)
```

### Arguments

brainview_images	vector of character strings, paths to the brainview images, usually in PNG format
output_img	path to output image that including the file extension
colorbar_img	path to the main image containing the separate colorbar, usually an image in PNG format
silent	logical, whether to suppress messages
grid_like	logical, whether to arrange the images in a grid-like fashion. If FALSE, they will all be merged horizontally.

border_geometry	string, a geometry string passed to <code>magick::image_border</code> to define the borders to add to each image tile. The default value adds 5 pixels, both horizontally and vertically.
background_color	string, a valid ImageMagick color string such as "white" or "#000080". The color to use when extending images (e.g., when creating the border).

### Value

vector of character strings, paths to the brainview images. This is the input parameter 'brainview\_images'.

---

brainviews	<i>Show one or more views of the given meshes in rgl windows.</i>
------------	---

---

### Description

Show one or more views of the given meshes in rgl windows.

### Usage

```
brainviews(
  views,
  coloredmeshes,
  rgloptions = list(),
  rglactions = list(),
  style = "default",
  draw_colorbar = FALSE
)
```

### Arguments

views	list of strings. Valid entries include: 'si': single interactive view. 'sd_<angle>': single view from angle <angle>. The <angle> part must be one of the strings returned by <code>get.view.angle.names</code> . Example: 'sd_caudal'. 'sr': single rotating view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
coloredmeshes	list of coloredmesh or renderable. A coloredmesh is a named list as returned by the <code>coloredmesh.from.*</code> functions. It has the entries 'mesh' of type <code>tmesh3d</code> , a 'col', which is a color specification for such a mesh.
rgloptions	option list passed to <code>par3d</code> . Example: <code>rgloptions = list("windowRect"=c(50,50,1000,1000))</code>
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action.
style	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.

`draw_colorbar` logical, whether to draw a colorbar. **WARNING:** The colorbar is drawn to a subplot, and this only works if there is enough space for it. You will have to increase the plot size using the `'rlgoptions'` parameter for the colorbar to show up. Defaults to `FALSE`. See [coloredmesh.plot.colorbar.separate](#) for an alternative.

### Value

list of coloredmeshes. The coloredmeshes used for the visualization.

### See Also

[get.view.angle.names](#)

---

clip.data

*Clip data at quantiles to remove outliers.*

---

### Description

Set all data values outside the given quantile range to the border values. This is useful to properly visualize morphometry data that includes outliers. These outliers negatively affect the colormap, as all the non-outlier values become hard to distinguish. This function can be used to filter the data before plotting it.

### Usage

```
clip.data(data, lower = 0.05, upper = 0.95)
```

### Arguments

`data`, numeric vector. The input data. Can also be a hemi list.  
`lower`, numeric. The probability for the lower quantile, defaults to `'0.05'`.  
`upper`, numeric. The probability for the upper quantile, defaults to `'0.95'`.

### Value

numeric vector. The output data.

### Examples

```
full_data = rnorm(50, 3, 1);
clipped = clip.data(full_data);
```

---

`cm.cbry`*Get cyan blue red yellow colormap function.*

---

**Description**

Get cyan blue red yellow colormap function.

**Usage**

```
cm.cbry()
```

**Note**

Returns a diverging palette with negative values in blue/cyan and positive ones in red/yellow, suitable for visualizing data that is centered around zero. Often used for clusters in neuroscience.

---

`cm.div`*Return the standard fsbrain diverging colormap.*

---

**Description**

Return the standard fsbrain diverging colormap.

**Usage**

```
cm.div(report = FALSE)
```

**Arguments**

`report` logical, whether to print a message with a name of the chosen colormap, in format `package::function#palette`.

**Note**

Returns some diverging palette, suitable for visualizing data that is centered around zero.

---

cm.heat	<i>Return the standard fsbrain heat colormap.</i>
---------	---

---

**Description**

Return the standard fsbrain heat colormap.

**Usage**

```
cm.heat(report = FALSE)
```

**Arguments**

report	logical, whether to print a message with a name of the chosen colormap, in format <code>package::function#palette</code> .
--------	--

**Note**

The heat palette is a sequential, single-hue palette.

---

cm.qual	<i>Return the standard fsbrain qualitative colormap.</i>
---------	--

---

**Description**

Return the standard fsbrain qualitative colormap.

**Usage**

```
cm.qual(report = FALSE)
```

**Arguments**

report	logical, whether to print a message with a name of the chosen colormap, in format <code>package::function#palette</code> .
--------	--

**Note**

Returns some qualitative palette, suitable for visualizing categorical data.

---

cm.seq	<i>Return the standard fsbrain sequential colormap.</i>
--------	---

---

**Description**

Return the standard fsbrain sequential colormap.

**Usage**

```
cm.seq(report = FALSE)
```

**Arguments**

report	logical, whether to print a message with a name of the chosen colormap, in format <code>package::function#palette</code> .
--------	--

**Note**

This returns a sequential, multi-hue palette.

---

collayer.bg	<i>Compute binarized mean curvature surface color layer.</i>
-------------	--

---

**Description**

Compute a binarized mean curvature surface color layer, this is intended as a background color layer. You can merge it with your data layer using [collayers.merge](#).

**Usage**

```
collayer.bg(subjects_dir, subject_id, bg, hemi = "both")
```

**Arguments**

subjects_dir	character string, the FreeSurfer SUBJECTS_DIR.
subject_id	character string, the subject identifier.
bg	character string, a background name. One of 'curv', 'sulc', or 'aparc'. If this is already a colorlayer in a hemilist, it will be returned as-is.
hemi	character string, one of 'lh', 'rh', or 'both'. The latter will merge the data for both hemis into a single vector.

**Value**

a color layer, i.e., vector of color strings in a hemilist

**See Also**

You can plot the return value using [vis.color.on.subject](#).

Other surface color layer: [collayer.bg.atlas\(\)](#), [collayer.bg.meancurv\(\)](#), [collayer.bg.sulc\(\)](#), [collayer.from.annotdata\(\)](#), [collayer.from.annot\(\)](#), [collayer.from.mask.data\(\)](#), [collayer.from.morphlike.d](#), [collayers.merge\(\)](#)

---

collayer.bg.atlas	<i>Compute atlas or annotation surface color layer.</i>
-------------------	---

---

**Description**

Compute atlas or annotation surface color layer.

**Usage**

```
collayer.bg.atlas(
    subjects_dir,
    subject_id,
    hemi = "both",
    atlas = "aparc",
    grayscale = FALSE,
    outline = FALSE,
    outline_surface = "white"
)
```

**Arguments**

subjects_dir	character string, the FreeSurfer SUBJECTS_DIR.
subject_id	character string, the subject identifier.
hemi	character string, one of 'lh', 'rh', or 'both'. The latter will merge the data for both hemis into a single vector.
atlas	character string, the atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
grayscale	logical, whether to convert the atlas colors to grayscale
outline	logical, whether to draw an outline only instead of filling the regions. Defaults to 'FALSE'. Instead of passing 'TRUE', one can also pass a list of extra parameters to pass to <a href="#">annot.outline</a> , e.g., <code>outline=list('outline_color'='#000000')</code> .
outline_surface	character string, the surface to load. Only relevant when 'outline' is used. (In that case the surface mesh is needed to compute the vertices forming the region borders.)

**Value**

a color layer, i.e., vector of color strings in a hemilist

**Note**

Using 'outline' mode is quite slow, and increasing the border thickness makes it even slower.

**See Also**

You can plot the return value using [vis.color.on.subject](#).

Other surface color layer: [collayer.bg.meancurv\(\)](#), [collayer.bg.sulc\(\)](#), [collayer.bg\(\)](#), [collayer.from.annotdata\(\)](#), [collayer.from.annot\(\)](#), [collayer.from.mask.data\(\)](#), [collayer.from.morphlike.d](#), [collayers.merge\(\)](#)

---

`collayer.bg.meancurv`    *Compute binarized mean curvature surface color layer.*

---

**Description**

Compute a binarized mean curvature surface color layer, this is intended as a background color layer. You can merge it with your data layer using [collayers.merge](#).

**Usage**

```
collayer.bg.meancurv(
  subjects_dir,
  subject_id,
  hemi = "both",
  cortex_only = FALSE,
  bin_colors = c("#898989", "#5e5e5e"),
  bin_thresholds = c(0)
)
```

**Arguments**

<code>subjects_dir</code>	character string, the FreeSurfer SUBJECTS_DIR.
<code>subject_id</code>	character string, the subject identifier.
<code>hemi</code>	character string, one of 'lh', 'rh', or 'both'. The latter will merge the data for both hemis into a single vector.
<code>cortex_only</code>	logical, whether to restrict pattern computation to the cortex.
<code>bin_colors</code>	vector of two character strings, the two colors to use.
<code>bin_thresholds</code>	vector of 1 or 2 double values, the curvature threshold values used to separate gyri from sulci.

**Value**

a color layer, i.e., vector of color strings in a hemilist



**See Also**

You can plot the return value using [vis.color.on.subject](#).

Other surface color layer: [collayer.bg.atlas\(\)](#), [collayer.bg.sulc\(\)](#), [collayer.bg\(\)](#), [collayer.from.annotdata\(\)](#), [collayer.from.annot\(\)](#), [collayer.from.mask.data\(\)](#), [collayer.from.morphlike.data\(\)](#), [collayers.merge\(\)](#)

---

collayer.bg.sulc	<i>Compute binarized sulcal depth surface color layer.</i>
------------------	--

---

**Description**

Compute a binarized sulcal depth surface color layer, this is intended as a background color layer. You can merge it with your data layer using [collayers.merge](#).

**Usage**

```
collayer.bg.sulc(
  subjects_dir,
  subject_id,
  hemi = "both",
  cortex_only = FALSE,
  bin_colors = c("#898989", "#5e5e5e"),
  bin_thresholds = c(0)
)
```

**Arguments**

subjects_dir	character string, the FreeSurfer SUBJECTS_DIR.
subject_id	character string, the subject identifier.
hemi	character string, one of 'lh', 'rh', or 'both'. The latter will merge the data for both hemis into a single vector.
cortex_only	logical, whether to restrict pattern computation to the cortex.
bin_colors	vector of two character strings, the two colors to use.
bin_thresholds	vector of 1 or 2 double values, the curvature threshold values used to separate gyri from sulci.

**Value**

a color layer, i.e., vector of color strings in a hemilist

**See Also**

You can plot the return value using [vis.color.on.subject](#).

Other surface color layer: [collayer.bg.atlas\(\)](#), [collayer.bg.meancurv\(\)](#), [collayer.bg\(\)](#), [collayer.from.annotdata\(\)](#), [collayer.from.annot\(\)](#), [collayer.from.mask.data\(\)](#), [collayer.from.morphlike.d](#), [collayers.merge\(\)](#)

---

collayer.from.annot     *Compute surface color layer from annotation or atlas data.*

---

### Description

Compute surface color layer from annotation or atlas data.

### Usage

```
collayer.from.annot(subjects_dir, subject_id, hemi, atlas)
```

### Arguments

subjects_dir	character string, the FreeSurfer SUBJECTS_DIR.
subject_id	character string, the subject identifier.
hemi	character string, one of 'lh', 'rh', or 'both'.
atlas	character string, the atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.

### Value

named hemi list, each entry is a vector of color strings, one color per surface vertex. The coloring represents the atlas data.

### See Also

You can plot the return value using [vis.color.on.subject](#).

Other surface color layer: [collayer.bg.atlas\(\)](#), [collayer.bg.meancurv\(\)](#), [collayer.bg.sulc\(\)](#), [collayer.bg\(\)](#), [collayer.from.annotdata\(\)](#), [collayer.from.mask.data\(\)](#), [collayer.from.morphlike.data\(\)](#), [collayers.merge\(\)](#)

---

collayer.from.annotdata     *Compute surface color layer from annotation or atlas data.*

---

### Description

Compute surface color layer from annotation or atlas data.

### Usage

```
collayer.from.annotdata(lh_annotdata = NULL, rh_annotdata = NULL)
```

**Arguments**

lh\_annotdata    loaded annotation data for left hemi, as returned by [subject.annot](#)  
rh\_annotdata    loaded annotation data for right hemi

**Value**

named hemi list, each entry is a vector of color strings, one color per surface vertex. The coloring represents the atlas data.

**See Also**

You can plot the return value using [vis.color.on.subject](#).

Other surface color layer: [collayer.bg.atlas\(\)](#), [collayer.bg.meancurv\(\)](#), [collayer.bg.sulc\(\)](#), [collayer.bg\(\)](#), [collayer.from.annot\(\)](#), [collayer.from.mask.data\(\)](#), [collayer.from.morphlike.data\(\)](#), [collayers.merge\(\)](#)

---

`collayer.from.mask.data`

*Compute surface color layer from morph-like data.*

---

**Description**

Compute surface color layer from morph-like data.

**Usage**

```
collayer.from.mask.data(
  lh_data = NULL,
  rh_data = NULL,
  makecmap_options = list(colFn = label.colFn)
)
```

**Arguments**

lh\_data            integer vector, can be NULL  
rh\_data            numerical vector, can be NULL  
makecmap\_options    named list of parameters to pass to [makecmap](#). Must not include the unnamed first parameter, which is derived from 'measure'.

**Value**

named hemi list, each entry is a vector of color strings, one color per surface vertex. The coloring represents the label data.

**See Also**

You can plot the return value using [vis.color.on.subject](#).

Other surface color layer: [collayer.bg.atlas\(\)](#), [collayer.bg.meancurv\(\)](#), [collayer.bg.sulc\(\)](#), [collayer.bg\(\)](#), [collayer.from.annotdata\(\)](#), [collayer.from.annot\(\)](#), [collayer.from.morphlike.data\(\)](#), [collayers.merge\(\)](#)

---

collayer.from.morphlike.data

*Compute surface color layer from morph-like data.*

---

**Description**

Compute surface color layer from morph-like data.

**Usage**

```
collayer.from.morphlike.data(
  lh_morph_data = NULL,
  rh_morph_data = NULL,
  makecmap_options = list(colFn = squash::jet),
  return_metadata = FALSE
)
```

**Arguments**

`lh_morph_data` numerical vector, can be NULL

`rh_morph_data` numerical vector, can be NULL

`makecmap_options` named list of parameters to pass to [makecmap](#). Must not include the unnamed first parameter, which is derived from 'measure'.

`return_metadata` logical, whether to return additional metadata as entry 'metadata' in the returned list

**Value**

named hemi list, each entry is a vector of color strings, one color per surface vertex. The coloring represents the morph data.

**See Also**

You can plot the return value using [vis.color.on.subject](#).

Other surface color layer: [collayer.bg.atlas\(\)](#), [collayer.bg.meancurv\(\)](#), [collayer.bg.sulc\(\)](#), [collayer.bg\(\)](#), [collayer.from.annotdata\(\)](#), [collayer.from.annot\(\)](#), [collayer.from.mask.data\(\)](#), [collayers.merge\(\)](#)

---

collayers.merge	<i>Merge two or more color layers based on their transparency values.</i>
-----------------	---

---

### Description

Merge several color layers into one based on their transparency and alpha blending. In the final result, the lower layers are visible through the transparent or 'NA' parts (if any) of the upper layers.

### Usage

```
collayers.merge(collayers, opaque_background = "#FFFFFF")
```

### Arguments

`collayers` named list, the values must be vectors, matrices or arrays of color strings (as produced by [rgb](#)). The names are freeform and do not really matter. All values must have the same length.

`opaque_background` a single color string or 'NULL'. If a color string, this color will be used as a final opaque background layer to ensure that the returned colors are all opaque. Pass 'NULL' to skip this, which may result in a return value that contains non-opaque color values.

### Value

a color layer, i.e., vector of color strings in a hemilist

### See Also

Other surface color layer: [collayer.bg.atlas\(\)](#), [collayer.bg.meancurv\(\)](#), [collayer.bg.sulc\(\)](#), [collayer.bg\(\)](#), [collayer.from.annotdata\(\)](#), [collayer.from.annot\(\)](#), [collayer.from.mask.data\(\)](#), [collayer.from.morphlike.data\(\)](#)

---

coloredmesh.from.annot
------------------------

---

*Create a coloredmesh from an annotation of an atlas.*

---

### Description

Create a coloredmesh from an annotation of an atlas.

**Usage**

```

coloredmesh.from.annot(
    subjects_dir,
    subject_id,
    atlas,
    hemi,
    surface = "white",
    outline = FALSE
)

```

**Arguments**

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
atlas	string or a loaded annotation. If a string, interpreted as the atlas name that should be loaded to get the annotation. E.g., "aparc", "aparc.2009s", or "aparc.DKAtlas". Used to construct the name of the annotation file to be loaded.
hemi	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
surface	character string or 'fs.surface' instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
outline	logical, whether to draw an outline only instead of filling the regions. Defaults to FALSE. Only makes sense if you did not pass an outline already. The current implementation for outline computation is rather slow, so setting this to TRUE will considerably increase computation time.

**Value**

coloredmesh. A named list with entries: "mesh" the `tmesh3d` mesh object. "col": the mesh colors. "render", logical, whether to render the mesh. "hemi": the hemisphere, one of 'lh' or 'rh'.

**See Also**

Other coloredmesh functions: `coloredmesh.from.label()`, `coloredmesh.from.mask()`, `coloredmesh.from.morph.nat`, `coloredmesh.from.morph.standard()`, `coloredmesh.from.morphdata()`, `coloredmeshes.from.color()`

---

coloredmesh.from.label

*Create a coloredmesh from a label.*

---

**Description**

Create a coloredmesh from a label.

**Usage**

```

coloredmesh.from.label(
    subjects_dir,
    subject_id,
    label,
    hemi,
    surface = "white",
    colormap = NULL,
    makecmap_options = list(colFn = squash::rainbow2)
)

```

**Arguments**

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
label	string or vector of integers. If a string, the name of the label file, without the hemi part (if any), but including the '.label' suffix. E.g., 'cortex.label' for '?h.cortex.label'. Alternatively, the already loaded label data as a vector of integers.
hemi	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
surface	character string or 'fs.surface' instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
colormap	a colormap function. See the squash package for some colormaps. Defaults to <a href="#">jet</a> .
makecmap_options	named list of parameters to pass to <a href="#">makecmap</a> . Must not include the unnamed first parameter, which is derived from 'measure'.

**Value**

coloredmesh. A named list with entries: "mesh" the [tmesh3d](#) mesh object. "col": the mesh colors. "render", logical, whether to render the mesh. "hemi": the hemisphere, one of 'lh' or 'rh'.

**See Also**

Other coloredmesh functions: [coloredmesh.from.annot\(\)](#), [coloredmesh.from.mask\(\)](#), [coloredmesh.from.morph.nat](#), [coloredmesh.from.morph.standard\(\)](#), [coloredmesh.from.morphdata\(\)](#), [coloredmeshes.from.color\(\)](#)

---

coloredmesh.from.mask *Create a coloredmesh from a mask.*

---

### Description

Create a coloredmesh from a mask.

### Usage

```
coloredmesh.from.mask(
  subjects_dir,
  subject_id,
  mask,
  hemi,
  surface = "white",
  colormap = NULL,
  surface_data = NULL,
  makecmap_options = list(colFn = squash::rainbow2)
)
```

### Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
mask	logical vector, contains one logical value per vertex.
hemi	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
surface	character string or 'fs.surface' instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
colormap	a colormap function. See the squash package for some colormaps. Defaults to <a href="#">jet</a> .
surface_data	optional surface mesh object, as returned by <a href="#">subject.surface</a> . If given, used instead of loading the surface data from disk (which users of this function may already have done). Defaults to NULL.
makecmap_options	named list of parameters to pass to <a href="#">makecmap</a> . Must not include the unnamed first parameter, which is derived from 'measure'.

### Value

coloredmesh. A named list with entries: "mesh" the [tmesh3d](#) mesh object. "col": the mesh colors. "render", logical, whether to render the mesh. "hemi": the hemisphere, one of 'lh' or 'rh'.



**See Also**

Other mask functions: [mask.from.labeldata.for.hemi\(\)](#), [vis.mask.on.subject\(\)](#)

Other coloredmesh functions: [coloredmesh.from.annot\(\)](#), [coloredmesh.from.label\(\)](#), [coloredmesh.from.morph.native\(\)](#), [coloredmesh.from.morph.standard\(\)](#), [coloredmesh.from.morphdata\(\)](#), [coloredmeshes.from.color\(\)](#)

---

coloredmesh.from.morph.native

*Create a coloredmesh from native space morphometry data.*

---

**Description**

Create a coloredmesh from native space morphometry data.

**Usage**

```
coloredmesh.from.morph.native(
    subjects_dir,
    subject_id,
    measure,
    hemi,
    surface = "white",
    colormap = NULL,
    clip = NULL,
    cortex_only = FALSE,
    makecmap_options = list(colFn = squash::jet)
)
```

**Arguments**

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
measure	string. The morphometry data to use. E.g., 'area' or 'thickness'. Pass NULL to render the surface in white, without any data. One can also pass the pre-loaded morphometry data as a numerical vector, the length of which must match the number of surface vertices.
hemi	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
surface	character string or 'fs.surface' instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
colormap	a colormap function. See the squash package for some colormaps. Defaults to <a href="#">jet</a> .
clip	numeric vector of length 2 or NULL. If given, the 2 values are interpreted as lower and upper percentiles, and the morph data is clipped at the given lower and upper percentile (see <a href="#">clip.data</a> ). Defaults to NULL (no data clipping).

cortex_only	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>*not*</i> part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subject. Defaults to FALSE.
makecmap_options	named list of parameters to pass to <code>makecmap</code> . Must not include the unnamed first parameter, which is derived from 'measure'.

### Value

coloredmesh. A named list with entries: "mesh" the `tmesh3d` mesh object. "col": the mesh colors. "render", logical, whether to render the mesh. "hemi": the hemisphere, one of 'lh' or 'rh'.

### See Also

Other coloredmesh functions: `coloredmesh.from.annot()`, `coloredmesh.from.label()`, `coloredmesh.from.mask()`, `coloredmesh.from.morph.standard()`, `coloredmesh.from.morphdata()`, `coloredmeshes.from.color()`

---

coloredmesh.from.morph.standard

*Create a coloredmesh from standard space morphometry data.*

---

### Description

Create a coloredmesh from standard space morphometry data.

### Usage

```
coloredmesh.from.morph.standard(
  subjects_dir,
  subject_id,
  measure,
  hemi,
  fwhm,
  surface = "white",
  template_subject = "fsaverage",
  template_subjects_dir = NULL,
  colormap = NULL,
  clip = NULL,
  cortex_only = FALSE,
  makecmap_options = list(colFn = squash::jet)
)
```

**Arguments**

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
measure	string. The morphometry data to use. E.g., 'area' or 'thickness'. Pass NULL to render the surface in white, without any data. One can also pass the pre-loaded morphometry data as a numerical vector, the length of which must match the number of surface vertices.
hemi	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
fwhm	string, smoothing setting. The smoothing part of the filename, typically something like '0', '5', '10', ..., or '25'.
surface	character string or 'fs.surface' instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
template_subject	The template subject used. This will be used as part of the filename, and its surfaces are loaded for data visualization. Defaults to 'fsaverage'.
template_subjects_dir	The template subjects dir. If 'NULL', the value of the parameter 'subjects_dir' is used. Defaults to NULL. If you have FreeSurfer installed and configured, and are using the standard fsaverage subject, try passing the result of calling 'file.path(Sys.getenv('FREESURFER_HOME'), 'subjects')'.
colormap	a colormap function. See the squash package for some colormaps. Defaults to <a href="#">jet</a> .
clip	numeric vector of length 2 or NULL. If given, the 2 values are interpreted as lower and upper percentiles, and the morph data is clipped at the given lower and upper percentile (see <a href="#">clip.data</a> ). Defaults to NULL (no data clipping).
cortex_only	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are *not* part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subject. Defaults to FALSE.
makecmap_options	named list of parameters to pass to <a href="#">makecmap</a> . Must not include the unnamed first parameter, which is derived from 'measure'.

**Value**

coloredmesh. A named list with entries: "mesh" the [tmesh3d](#) mesh object. "col": the mesh colors. "render", logical, whether to render the mesh. "hemi": the hemisphere, one of 'lh' or 'rh'.

**See Also**

Other coloredmesh functions: [coloredmesh.from.annot\(\)](#), [coloredmesh.from.label\(\)](#), [coloredmesh.from.mask\(\)](#), [coloredmesh.from.morph.native\(\)](#), [coloredmesh.from.morphdata\(\)](#), [coloredmeshes.from.color\(\)](#)

---

coloredmesh.from.morphdata

*Create a coloredmesh from arbitrary data.*

---

## Description

Create a coloredmesh from arbitrary data.

## Usage

```
coloredmesh.from.morphdata(
    subjects_dir,
    vis_subject_id,
    morph_data,
    hemi,
    surface = "white",
    colormap = NULL,
    makecmap_options = list(colFn = squash::jet)
)
```

## Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
vis_subject_id	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
morph_data	string. The morphometry data to use. E.g., 'area' or 'thickness'.
hemi	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
surface	character string or 'fs.surface' instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
colormap	a colormap function. See the squash package for some colormaps. Defaults to <a href="#">jet</a> .
makecmap_options	named list of parameters to pass to <a href="#">makecmap</a> . Must not include the unnamed first parameter, which is derived from 'measure'.

## Value

coloredmesh. A named list with entries: "mesh" the [tmesh3d](#) mesh object. "col": the mesh colors. "render", logical, whether to render the mesh. "hemi": the hemisphere, one of 'lh' or 'rh'.

## See Also

Other coloredmesh functions: [coloredmesh.from.annot\(\)](#), [coloredmesh.from.label\(\)](#), [coloredmesh.from.mask\(\)](#), [coloredmesh.from.morph.native\(\)](#), [coloredmesh.from.morph.standard\(\)](#), [coloredmeshes.from.color\(\)](#)

---

`coloredmesh.from.preloaded.data`*Generate coloredmesh from loaded data.*

---

### Description

Generate coloredmesh from loaded data.

### Usage

```
coloredmesh.from.preloaded.data(  
  fs_surface,  
  morph_data = NULL,  
  col = NULL,  
  hemi = "lh",  
  makecmap_options = mkco.seq()  
)
```

### Arguments

<code>fs_surface</code>	an <code>fs.surface</code> instance
<code>morph_data</code>	numerical vector, per-vertex data (typically morphometry) for the mesh. If given, takes precedence over <code>'col'</code> parameter.
<code>col</code>	vector of colors, typically hex color strings like <code>'#FF00FF'</code> . The per-vertex-colors for the mesh. Alternative to <code>morph_data</code> .
<code>hemi</code>	character string, one of <code>'lh'</code> or <code>'rh'</code> . Metadata, the hemisphere. May be used by visualization functions to decide whether to draw the mesh in certain views.
<code>makecmap_options</code>	named list of parameters to pass to <code>makecmap</code> . Must not include the unnamed first parameter, which is derived from <code>'measure'</code> .

### Value

as `fs.coloredmesh` instance

---

`coloredmesh.plot.colorbar.separate`*Draw colorbar for coloredmeshes in separate 2D plot.*

---

## Description

Draw a colorbar for the coloredmeshes to a separate 2D plot. Due to the suboptimal handling of colorbar drawing in the three-dimensional multi-panel views, it is often desirable to plot the colorbar in a separate window, export it from there and then manually add it to the final plot version in some image manipulation software like Inkscape. If you need more control over the colormap than offered by this function (e.g., setting the color value for NA values or making a symmetric colormap to ensure that the zero point for divergent colormaps is a neutral color), you should write custom code, and the return value from this function will come in handy to do that.

## Usage

```
coloredmesh.plot.colorbar.separate(
  coloredmeshes,
  show = FALSE,
  image.plot_extra_options = list(horizontal = FALSE, legend.cex = 1.8, legend.width =
    2, legend.mar = 12, axis.args = list(cex.axis = 5)),
  png_options = list(filename = "fsbrain_cbar.png", width = 1400, height = 1400, bg =
    "#FFFFFF00"),
  silent = FALSE,
  trim_png = TRUE
)
```

## Arguments

- |                          |  |
|--------------------------|--|
| coloredmeshes            | list of coloredmeshes. A coloredmesh is a named list as returned by the ‘coloredmesh.from’ functions. It has the entries ‘mesh’ of type tmesh3d, a ‘col’, which is a color specification for such a mesh. The ‘vis*’ functions (like <a href="#">vis.subject.morph.native</a> ) all return a list of coloredmeshes.  |
| show                     | logical, Whether to open the resulting plot. Defaults to ‘TRUE’.   |
| image.plot_extra_options | named list of extra options to pass to <a href="#">image.plot</a> . This can be used to add a legend to the colorbar, rotate the colorbar, or whatever. The options "legend_only", "zlim", and "col" are computed and set for you by this function, so there is no need to pass these. Your list will be merged with the internal options, so you could overwrite named arguments if needed. |
| png_options              | Options to pass to <a href="#">png</a> , see the docs of that function for details. Allow you to save the plot as a png bitmap image. Example: png_options = list("filename"="fsbrain_cbar.png") Defaults to NULL, which will not save anything.   |
| silent                   | logical, whether to suppress messages. Defaults to ‘FALSE’.  |
| trim_png                 | logical, whether to trim the output PNG image using image magick, i.e., remove everything but the foreground. Ignored unless an output PNG image is actually written (see ‘png_options’) and the ‘magick’ package is installed.  |

## Note

If you increase the output resolution of the colorbar (using ‘png\_options’), you will have to increase the font sizes as well (using ‘image.plot\_extra\_options’), otherwise the axis and legend labels will be hard to read.

**See Also**

Other colorbar functions: [combine.colorbar.with.brainview.animation\(\)](#), [combine.colorbar.with.brainview.ima](#)

**Examples**

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
coloredmeshes = vis.subject.morph.native(subjects_dir, 'subject1',
  'thickness', 'lh', views=c('t4'));
coloredmesh.plot.colorbar.separate(coloredmeshes);

# Or plot a colorbar with a label:
coloredmesh.plot.colorbar.separate(coloredmeshes,
  image.plot_extra_options = list("legend.lab"="Thickness [mm]",
  horizontal=TRUE, legend.cex=1.5, legend.line=-3));

## End(Not run)
```

---

coloredmeshes.from.color

*Create coloredmeshes for both hemis using pre-defined colors.*

---

**Description**

Create coloredmeshes for both hemis using pre-defined colors.

**Usage**

```
coloredmeshes.from.color(
  subjects_dir,
  subject_id,
  color_data,
  hemi,
  surface = "white",
  metadata = list()
)
```

**Arguments**

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
color_data	a hemilist containing vectors of hex color strings
hemi	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.

surface	character string or 'fs.surface' instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
metadata	a named list, can contain whatever you want. Typical entries are: 'src_data' a hemilist containing the source data from which the 'color_data' was created, optional. If available, it is encoded into the coloredmesh and can be used later to plot a colorbar. 'makecmap_options': the options used to created the colormap from the data.

**Value**

named list of coloredmeshes. Each entry is a named list with entries: "mesh" the [tmesh3d](#) mesh object. "col": the mesh colors. "render", logical, whether to render the mesh. "hemi": the hemisphere, one of 'lh' or 'rh'.

**See Also**

Other coloredmesh functions: [coloredmesh.from.annot\(\)](#), [coloredmesh.from.label\(\)](#), [coloredmesh.from.mask\(\)](#), [coloredmesh.from.morph.native\(\)](#), [coloredmesh.from.morph.standard\(\)](#), [coloredmesh.from.morphdata\(\)](#)

---

colorlist.brain.clusters

*Return diverging color list*

---

**Description**

Return diverging color list

**Usage**

```
colorlist.brain.clusters(num_colors)
```

**Arguments**

num\_colors      integer, the number of colors you want

**Value**

vector of colors



---

```
combine.colorbar.with.brainview.animation
```

*Combine a colorbar and a brain animation in gif format into a new animation.*

---

## Description

Combine a colorbar and a brain animation in gif format into a new animation.

## Usage

```
combine.colorbar.with.brainview.animation(  
    brain_animation,  
    colorbar_img,  
    output_animation,  
    offset = "+0+0",  
    extend_brainview_img_height_by = 0L,  
    silent = FALSE,  
    allow_colorbar_shrink = TRUE  
)
```

## Arguments

brain_animation	path to the brain animation in gif format
colorbar_img	path to the main image containing the separate colorbar, usually an image in PNG format
output_animation	path to output image in gif format, must include the '.gif' file extension
offset	offset string passed to <code>magick::image_composite</code> . Allows you to shift the location of the colorbar in the final image.
extend_brainview_img_height_by	integer value in pixels, the size of the lower border to add to the <code>brainview_img</code> . Use this if the lower part of the colorbar is off the image canvas.
silent	logical, whether to silence all messages
allow_colorbar_shrink	logical, whether to shrink the colorbar to the width of the animation in case it is considerably wider (more than 20 percent). Defaults to TRUE.

## See Also

Other colorbar functions: [coloredmesh.plot.colorbar.separate\(\)](#), [combine.colorbar.with.brainview.image\(\)](#)

---

```
combine.colorbar.with.brainview.image
```

*Combine a colorbar and a brainview image into a new figure.*

---

### Description

Combine a colorbar and a brainview image into a new figure.

### Usage

```
combine.colorbar.with.brainview.image(
    brainview_img = "fsbrain_arranged.png",
    colorbar_img = "fsbrain_cbar.png",
    output_img = "fsbrain_merged.png",
    offset = "+0+0",
    extend_brainview_img_height_by = NULL,
    silent = FALSE,
    allow_colorbar_shrink = TRUE,
    horizontal = FALSE
)
```

### Arguments

<code>brainview_img</code>	path to the main image containing the view of the brain, usually an image in PNG format.
<code>colorbar_img</code>	path to the main image containing the separate colorbar, usually an image in PNG format.
<code>output_img</code>	path to output image, including the file extension.
<code>offset</code>	offset string passed to <code>magick::image_composite</code> . Allows you to shift the location of the colorbar in the final image.
<code>extend_brainview_img_height_by</code>	integer value in pixels, the size of the lower border to add to the <code>brainview_img</code> . Increase this if the lower part of the colorbar is off the image canvas.
<code>silent</code>	logical, whether to silence all messages
<code>allow_colorbar_shrink</code>	logical, whether to shrink the colorbar to the width of the animation in case it is considerably wider (more than 20 percent). Defaults to TRUE.
<code>horizontal</code>	logical, whether the colorbar is horizontal. If so, it will be added below the <code>'brainview_img'</code> . If it is vertical, it will be added to the right of the <code>'brainview_img'</code> .

### See Also

Other colorbar functions: [coloredmesh.plot.colorbar.separate\(\)](#), [combine.colorbar.with.brainview.animation\(\)](#)

---

 cube3D.tris

*Return triangles for a 3D cube or cuboid.*


---

### Description

Each row of the returned matrix encodes a point (the x, y, and z coordinates), and 3 consecutive rows encode a triangle. Obviously, a point will occur several times (as part of several triangles). The result can be passed to [triangles3d](#) to render a 3D box. The defaults for the parameters will create a cube with edge length 1 centered at (0, 0, 0).

### Usage

```
cube3D.tris(
  xmin = -0.5,
  xmax = 0.5,
  ymin = -0.5,
  ymax = 0.5,
  zmin = -0.5,
  zmax = 0.5,
  center = NULL,
  edge_length = 1
)
```

### Arguments

xmin	numeric, minimal x coordinate
xmax	numeric, maximal x coordinate
ymin	numeric, minimal y coordinate
ymax	numeric, maximal y coordinate
zmin	numeric, minimal z coordinate
zmax	numeric, maximal z coordinate
center	numeric vector of length 3 or NULL, coordinates where to center a cube with the edge length defined in parameter 'edge_length'. If this is not 'NULL', the parameters 'xmin', 'xmax', ... will be ignored, and their values will be computed for a cube based on the 'center' and 'edge_length'. Note that you can only create cubes using 'center' and 'edge_length', while the min/max methods allows the construction of cuboids.
edge_length	numeric, the edge length of the cube. Only used if parameter 'center' is used, ignored otherwise.

### Value

numerical matrix with 36 rows and 3 columns, the 3D coordinates. Each row encodes a point (the x, y, and z coordinates), and 3 consecutive rows encode a triangle.

**Examples**

```
# Create a cube with edge length 2, centered at (3,4,5):
cube_coords = cube3D.tris(center=c(3,4,5), edge_length=2.0);
# Create the same cube using the min/max method:
cube_coords = cube3D.tris(xmin=2, xmax=4, ymin=3, ymax=5, zmin=4, zmax=6);
# Create a cuboid:
cuboid_coords = cube3D.tris(xmin=2, xmax=4, ymin=3, ymax=9, zmin=4, zmax=5);
# To render the cuboid:
#rgl::triangles3d(cuboid_coords, col="red");
```

---

cubes3D.tris

*Vectorized version of cube3D.tris*


---

**Description**

Vectorized version of cube3D.tris

**Usage**

```
cubes3D.tris(centers, edge_length = 1)
```

**Arguments**

centers	numerical matrix with 3 columns. Each column represents the x, y, z coordinates of a center at which to create a cube.
edge_length	numerical vector or scalar, the edge length. Must have length 1 (same edge length for all cubes), or the length must be identical to the number of rows in parameter 'centers'.

**Value**

matrix of triangle coordinates, see [cube3D.tris](#).

**Examples**

```
# Plot a 3D cloud of 20000 voxels:
centers = matrix(rnorm(20000*3)*100, ncol=3);
rgl::triangles3d(cubes3D.tris(centers));
```

---

`delete_all_optional_data`*Delete all data in the package cache.*

---

**Description**

Delete all data in the package cache.

**Usage**

```
delete_all_optional_data()
```

**Value**

integer. The return value of the unlink() call: 0 for success, 1 for failure. See the unlink() documentation for details.

---

`desaturate`*Perform simple desaturation or grayscale conversion of RGBA colors.*

---

**Description**

Perform simple desaturation or grayscale conversion of RGBA colors.

**Usage**

```
desaturate(color, gamma_correct = FALSE)
```

**Arguments**

`color`            rgba color strings

`gamma_correct`   logical, whether to apply non-linear gamma correction. First performs gamma expansion, then applies the gray-scale channel weights, then gamma compression.

**Value**

rgba color strings, the grayscale colors. The information from one of the three rgb channels would be enough. The alpha value is not touched.

**Note**

Assumes sRGB color space.

## References

see [https://en.wikipedia.org/wiki/Grayscale#Converting\\_color\\_to\\_grayscale](https://en.wikipedia.org/wiki/Grayscale#Converting_color_to_grayscale)

## See Also

Other color functions: [alphablend\(\)](#)

---

download\_fsaverage     *Download the FreeSurfer v6 fsaverage subject.*

---

## Description

Download some relevant files from the FreeSurfer v6 fsaverage subject. The files are subject to the FreeSurfer software license, see parameter 'accept\_freesurfer\_license' for details. This data is not required for the package to work. If you are working on a machine that has FreeSurfer installed, you already have this data anyways and do not need it. If not, it is very convenient to have it if you are using the fsaverage template subject to analyze your standard space data, as it is required for visualization of such data.

## Usage

```
download_fsaverage(accept_freesurfer_license = FALSE)
```

## Arguments

`accept_freesurfer_license`  
logical, whether you accept the FreeSurfer license for fsaverage, available at <https://surfer.nmr.mgh.harvard.edu/fswiki/FreeSurferSoftwareLicense>. Defaults to FALSE.

## Value

Named list. The list has entries: "available": vector of strings. The names of the files that are available in the local file cache. You can access them using `get_optional_data_file()`. "missing": vector of strings. The names of the files that this function was unable to retrieve.

---

`download_optional_data`*Download optional data for this package if required.*

---

### Description

Ensure that the optional data is available locally in the package cache. Will try to download the data only if it is not available. This data is not required for the package to work, but it is used in the examples, in the unit tests and also in the example code from the vignette. Downloading it is highly recommended.

### Usage

```
download_optional_data()
```

### Value

Named list. The list has entries: "available": vector of strings. The names of the files that are available in the local file cache. You can access them using `get_optional_data_file()`. "missing": vector of strings. The names of the files that this function was unable to retrieve.

---

`export.coloredmesh.ply`*Export a coloredmeshes with vertexcolors in PLY format.*

---

### Description

Exports coloredmeshes with vertex coloring to standard mesh files in Stanford Triangle (PLY) format. This is very hand for rendering in external standard 3D modeling software like **Blender**.

### Usage

```
export.coloredmesh.ply(filepath, coloredmesh)
```

### Arguments

<code>filepath</code>	The export filepath, including file name and extension.
<code>coloredmesh</code>	an 'fs.coloredmesh' instance, as returned (silently) by all surface visualization functions, like <a href="#">vis.subject.morph.native</a> .

**Examples**

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
coloredmeshes = vis.subject.morph.native(subjects_dir, 'subject1', 'thickness');
export.coloredmesh.ply('~/.subject1_thickness_1h.ply', coloredmeshed$1h);

## End(Not run)
```

---

face.edges

*Enumerate all edges of the given faces or mesh.*


---

**Description**

Compute edges of a tri-mesh. Can compute all edges, or only a subset, given by the face indices in the mesh.

**Usage**

```
face.edges(surface_mesh, face_indices = "all")
```

**Arguments**

surface\_mesh     surface mesh, as loaded by [subject.surface](#) or [read.fs.surface](#).

face\_indices     integer vector, the face indices. Can also be the character string 'all' to use all faces.

**Value**

integer matrix of size (n, 2) where n is the number of edges. The indices (source and target vertex) in each row are **not** sorted, and the edges are **not** unique. I.e., each undirected edge 'u, v' (with the exception of edges on the mesh border) will occur twice in the result: once as 'u, v' and once as 'v, u'.

**See Also**

Other surface mesh functions: [label.border\(\)](#), [mesh.vertex.included.faces\(\)](#), [mesh.vertex.neighbors\(\)](#), [subject.surface\(\)](#), [vis.path.along.verts\(\)](#)



---

find.freesurferhome     *Find the FREESURFER\_HOME directory on disk.*

---

### Description

Try to find directory containing the FreeSurfer installation, based on environment variables and \*educated guessing\*.

### Usage

```
find.freesurferhome(mustWork = FALSE)
```

### Arguments

mustWork            logical. Whether the function should with an error stop if the directory cannot be found. If this is TRUE, the return value will be only the 'found\_at' entry of the list (i.e., only the path of the FreeSurfer installation dir).

### Value

named list with the following entries: "found": logical, whether it was found. "found\_at": Only set if found=TRUE, the path to the FreeSurfer installation directory (including the directory itself). See 'mustWork' for important information.

---

find.subjectsdir.of     *Find the subject directory containing the fsaverage subject (or others) on disk.*

---

### Description

Try to find directory containing the fsaverage subject (or any other subject) by checking in the following places and returning the first path where it is found: first, the directory given by the environment variable SUBJECTS\_DIR, then in the subir 'subjects' of the directory given by the environment variable FREESURFER\_HOME, and finally the base dir of the package cache. See the function [download\\_fsaverage](#) if you want to download fsaverage to your package cache and ensure it always gets found, no matter whether the environment variables are set or not.

### Usage

```
find.subjectsdir.of(subject_id = "fsaverage", mustWork = FALSE)
```

### Arguments

subject\_id            string, the subject id of the subject. Defaults to 'fsaverage'.  
 mustWork            logical. Whether the function should with an error stop if the directory cannot be found. If this is TRUE, the return value will be only the 'found\_at' entry of the list (i.e., only the path of the subjects dir).

**Value**

named list with the following entries: "found": logical, whether it was found. "found\_at": Only set if found=TRUE, the path to the fsaverage directory (NOT including the fsaverage dir itself). "found\_all\_locations": list of all locations in which it was found. See 'mustWork' for important information.

---

fs.coloredmesh	<i>fs.coloredmesh constructor</i>
----------------	-----------------------------------

---

**Description**

fs.coloredmesh constructor

**Usage**

```
fs.coloredmesh(
  mesh,
  col,
  hemi,
  render = TRUE,
  metadata = NULL,
  add_normals = FALSE
)
```

**Arguments**

mesh	a 'mesh3d' instance as returned by <a href="#">tmesh3d</a> or an 'fs.surface' brain surface mesh as returned by functions like <a href="#">subject.surface</a> .
col	vector of vertex colors for the mesh, one color per vertex. Expanded if exactly one color.
hemi	character string, one of 'lh' or 'rh'. This may be used by visualization functions to decide whether or not to show this mesh in a certain view.
render	logical, whether to render this mesh during visualization
metadata	optional, named list containing metadata
add_normals	logical, whether to compute normals and save them in the mesh.

**Value**

an 'fs.coloredmesh' instance. The only fields one should use in client code are 'mesh', 'hemi' and 'col', all others are considered internal and may change without notice.

---

fs.home	<i>Return FreeSurfer path.</i>
---------	--------------------------------

---

**Description**

Return FreeSurfer path.

**Usage**

```
fs.home()
```

**Value**

the FreeSurfer path, typically what the environment variable 'FREESURFER\_HOME' points to.

**Note**

This function will stop (i.e., raise an error) if the directory cannot be found.

---

fsaverage.path	<i>Return path to fsaverage dir.</i>
----------------	--------------------------------------

---

**Description**

Return path to fsaverage dir.

**Usage**

```
fsaverage.path()
```

**Value**

the path to the fsaverage directory (NOT including the 'fsaverage' dir itself).

**Note**

This function will stop (i.e., raise an error) if the directory cannot be found.

fup *Transform first character of a string to uppercase.*

---

### Description

Transform first character of a string to uppercase. This is useful when labeling plots. Important: this function does not know about different encodings, languages or anything, it just calls `toupper` for the first character.

### Usage

```
fup(word)
```

### Arguments

word,                    string. Any string.

### Value

string, the input string with the first character transformed to uppercase.

### Examples

```
word_up = fup("word");
```

---

get.atlas.region.names  
*Determine atlas region names from a subject.*

---

### Description

Determine atlas region names from a subject. **WARNING:** Not all subjects have all regions of an atlas. You should use an average subject like `fsaverage` to get all regions.

### Usage

```
get.atlas.region.names(  
  atlas,  
  template_subjects_dir = NULL,  
  template_subject = "fsaverage",  
  hemi = "lh"  
)
```

**Arguments**

atlas,	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
template_subjects_dir,	string. The directory containing the dir of the template_subject. E.g., the path to FREESURFER_HOME/subjects. If NULL, env vars will be searched for candidates, and the function will fail if they are not set correctly. Defaults to NULL.
template_subject,	string. The subject identifier. Defaults to 'fsaverage'.
hemi,	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded. Defaults to 'lh'. Should not matter much, unless you do not have the file for one of the hemis for some reason.

**Value**

vector of strings, the region names.

**See Also**

Other atlas functions: [group.agg.atlas.native\(\)](#), [group.agg.atlas.standard\(\)](#), [group.annot\(\)](#), [group.label.from.annot\(\)](#), [label.from.annotdata\(\)](#), [label.to.annot\(\)](#), [regions.to.ignore\(\)](#), [spread.values.over.annot\(\)](#), [spread.values.over.hemi\(\)](#), [spread.values.over.subject\(\)](#), [subject.annot\(\)](#), [subject.atlas.agg\(\)](#), [subject.label.from.annot\(\)](#), [subject.lobes\(\)](#)

**Examples**

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
atlas_regions = get.atlas.region.names('aparc',
  template_subjects_dir=subjects_dir, template_subject='subject1');
```

---

get.view.angle.names *Get list of valid view angle names.*

---

**Description**

The returned strings are used as constants to identify a view of type 'sd\_<angle>'. They can be used to construct entries for the parameter 'views' of functions like [vis.subject.morph.native](#), or directly as parameter 'view\_angles' for functions like [vislayout.from.coloredmeshes](#).

**Usage**

```
get.view.angle.names(angle_set = "all", add_sd_prefix = TRUE)
```

**Arguments**

- `angle_set` string, which view subset to return. Available subsets are: 'all' (or alias 't9'): for all 9 angles. 't4': for the t4 views. 'medial': the 2 medial views, one for each hemi. 'lateral': the 2 lateral views, one for each hemi. 'lh': medial and lateral for the left hemisphere. 'rh': medial and lateral for the right hemisphere.
- `add_sd_prefix` logical, whether the prefix 'sd\_' should be added to the string. This will construct full view names. If set to false, only the substring after the prefix 'sd\_' will be returned. This is used internally only and should not be needed in general.

**Value**

vector of character strings, all valid view angle strings.

---

`getIn` *Retrieve values from nested named lists*

---

**Description**

Retrieve values from nested named lists

**Usage**

```
getIn(named_list, listkeys, default = NULL)
```

**Arguments**

- `named_list` a named list
- `listkeys` vector of character strings, the nested names of the lists
- `default` the default value to return in case the requested value is 'NULL'.

**Value**

the value at the path through the lists, or 'NULL' (or the 'default') if no such path exists.

**Examples**

```
data = list("regions"=list("frontal"=list("thickness"=2.3, "area"=2345)));
getIn(data, c("regions", "frontal", "thickness")); # 2.3
getIn(data, c("regions", "frontal", "nosuchentry")); # NULL
getIn(data, c("regions", "nosuchregion", "thickness")); # NULL
getIn(data, c("regions", "nosuchregion", "thickness"), default=14); # 14
```

---

`get_optional_data_filepath`*Access a single file from the package cache by its file name.*

---

**Description**

Access a single file from the package cache by its file name.

**Usage**

```
get_optional_data_filepath(filename, mustWork = TRUE)
```

**Arguments**

`filename`, string. The filename of the file in the package cache.  
`mustWork`, logical. Whether an error should be created if the file does not exist. If `mustWork=FALSE` and the file does not exist, the empty string is returned.

**Value**

string. The full path to the file in the package cache or the empty string if there is no such file available. Use this in your application code to open the file.

---

`group.agg.atlas.native`*Aggregate native space morphometry data over brain atlas regions and subjects for a group of subjects.*

---

**Description**

Aggregate native space morphometry data over brain atlas regions, e.g., compute the mean thickness value over all regions in an atlas for all subjects.

**Usage**

```
group.agg.atlas.native(  
  subjects_dir,  
  subjects_list,  
  measure,  
  hemi,  
  atlas,  
  agg_fun = mean,  
  cache_file = NULL  
)
```

**Arguments**

<code>subjects_dir</code> ,	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
<code>subjects_list</code> ,	string vector. A vector of subject identifiers that match the directory names within <code>subjects_dir</code> .
<code>measure</code> ,	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
<code>hemi</code> ,	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded. If set to 'both', combined data for 'lh' and 'rh' will be used.
<code>atlas</code> ,	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKAtlas". Used to construct the name of the annotation file to be loaded.
<code>agg_fun</code> ,	function. An R function that aggregates data, typically max, mean, min or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to mean.
<code>cache_file</code> ,	string or NULL. If given, it is interpreted as path of a file, and the data will be cached in the file <code>cache_file</code> in RData format. If the file does not exist yet, the function will run and cache the data in the file. If the file exists, the function will load the data from the file instead of running. The filename should end in '.RData', but that is not enforced or checked in any way. WARNING: If cached data is returned, all parameters passed to this function (with the exception of 'cache_file') are ignored! Whether the cached data is for another <code>subjects_list</code> or <code>hemi</code> is NOT checked! You have to ensure this yourself, by using different filenames. Defaults to NULL.

**Value**

dataframe with aggregated values for all regions and subjects, with `n` columns and `m` rows, where `n` is the number of subjects and `m` is the number of regions.

**See Also**

Other aggregation functions: [group.agg.atlas.standard\(\)](#), [group.morph.agg.standard.vertex\(\)](#), [subject.atlas.agg\(\)](#)

Other atlas functions: [get.atlas.region.names\(\)](#), [group.agg.atlas.standard\(\)](#), [group.annot\(\)](#), [group.label.from.annot\(\)](#), [label.from.annotdata\(\)](#), [label.to.annot\(\)](#), [regions.to.ignore\(\)](#), [spread.values.over.annot\(\)](#), [spread.values.over.hemi\(\)](#), [spread.values.over.subject\(\)](#), [subject.annot\(\)](#), [subject.atlas.agg\(\)](#), [subject.label.from.annot\(\)](#), [subject.lobes\(\)](#)

**Examples**

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
agg = group.agg.atlas.native(subjects_dir, c('subject1', 'subject2'),
  'thickness', 'lh', 'aparc');
```



```
# Visualize the mean values. Could use any subject, typically
# one would use fsaverage. Here we use subject1:
agg$subject = NULL; # remove non-numeric column.
vis.region.values.on.subject(subjects_dir, 'subject1', 'aparc',
  lh_region_value_list=colMeans(agg), rh_region_value_list=NULL);
```

---

```
group.agg.atlas.standard
```

*Aggregate standard space morphometry data over brain atlas regions and subjects for a group of subjects.*

---

## Description

Aggregate standard space morphometry data over brain atlas regions, e.g., compute the mean thickness value over all regions in an atlas for all subjects.

## Usage

```
group.agg.atlas.standard(
  subjects_dir,
  subjects_list,
  measure,
  hemi,
  atlas,
  fwhm,
  agg_fun = mean,
  template_subject = "fsaverage",
  cache_file = NULL
)
```

## Arguments

subjects_dir,	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subjects_list,	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
measure,	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
hemi,	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded. If set to 'both', combined data for 'lh' and 'rh' will be used.
atlas,	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKAtlas". Used to construct the name of the annotation file to be loaded.

fwhm,	string. The smoothing setting which was applied when mapping data to the template subject. Usually one of '0', '5', '10', '15', '20', or '25'.
agg_fun,	function. An R function that aggregates data, typically max, mean, min or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to mean.
template_subject,	string. The template subject name. Defaults to 'fsaverage'. Must have its data in subjects_dir.
cache_file,	string or NULL. If given, it is interpreted as path of a file, and the data will be cached in the file cache_file in RData format. If the file does not exist yet, the function will run and cache the data in the file. If the file exists, the function will load the data from the file instead of running. The filename should end in '.RData', but that is not enforced or checked in any way. WARNING: If cached data is returned, all parameters passed to this function (with the exception of 'cache_file') are ignored! Whether the cached data is for another subjects_list or hemi is NOT checked! You have to ensure this yourself, by using different filenames. Defaults to NULL.

### Value

dataframe with aggregated values for all regions and subjects, with n columns and m rows, where n is the number of subjects and m is the number of regions.

### See Also

Other aggregation functions: [group.agg.atlas.native\(\)](#), [group.morph.agg.standard.vertex\(\)](#), [subject.atlas.agg\(\)](#)

Other atlas functions: [get.atlas.region.names\(\)](#), [group.agg.atlas.native\(\)](#), [group.annot\(\)](#), [group.label.from.annot\(\)](#), [label.from.annotdata\(\)](#), [label.to.annot\(\)](#), [regions.to.ignore\(\)](#), [spread.values.over.annot\(\)](#), [spread.values.over.hemi\(\)](#), [spread.values.over.subject\(\)](#), [subject.annot\(\)](#), [subject.atlas.agg\(\)](#), [subject.label.from.annot\(\)](#), [subject.lobes\(\)](#)

### Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
agg = group.agg.atlas.standard(subjects_dir, c('subject1', 'subject2'),
  'thickness', 'lh', 'aparc', fwhm='10');
# Visualize the mean values. Could use any subject, typically
# one would use fsaverage. Here we use subject1:
agg$subject = NULL; # remove non-numeric column.
vis.region.values.on.subject(subjects_dir, 'subject1', 'aparc',
  lh_region_value_list=colMeans(agg), rh_region_value_list=NULL);

## End(Not run)
```

---

`group.annot`*Load annotations for a group of subjects.*

---

### Description

Load a brain surface annotation, i.e., a cortical parcellation based on an atlas, for a group of subjects.

### Usage

```
group.annot(subjects_dir, subjects_list, hemi, atlas)
```

### Arguments

`subjects_dir`, string. The FreeSurfer SUBJECTS\_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.

`subjects_list`, vector of strings. The subject identifiers.

`hemi`, string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.

`atlas`, string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.

### Value

list of annotations, as returned by `freesurferformats::read.fs.annot()`. If `hemi` is 'both', the annotations are the results of merging over the hemispheres for each subject.

### See Also

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.label.from.annot()`, `label.from.annotdata()`, `label.to.annot()`, `regions.to.ignore()`, `spread.values.over.annot()`, `spread.values.over.hemi()`, `spread.values.over.subject()`, `subject.annot()`, `subject.atlas.agg()`, `subject.label.from.annot()`, `subject.lobes()`

### Examples

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
subjects_list = c("subject1", "subject2");
annotations = group.annot(subjects_dir, subjects_list, "lh", "aparc");
```

---

`group.concat.measures.native`*Concatenate native space data for a group of subjects.*

---

### Description

A measure is something like 'thickness' or 'area'. This function concatenates the native space data for all subjects into a single long vector for each measure. A dataframe is then created, in which each column is one such vector. This can be used to compute the correlation between measures on vertex level, for example.

### Usage

```
group.concat.measures.native(  
  subjects_dir,  
  subjects_list,  
  measures,  
  hemi,  
  cortex_only = FALSE  
)
```

### Arguments

<code>subjects_dir</code> ,	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
<code>subjects_list</code> ,	string vector. A vector of subject identifiers that match the directory names within <code>subjects_dir</code> .
<code>measures</code> ,	vector of strings. Names of the vertex-wise morphometry measures. E.g., <code>c("area", "thickness")</code> . Used to construct the names of the morphometry file to be loaded. The data of each measure will be one column in the resulting dataframe.
<code>hemi</code> ,	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
<code>cortex_only</code>	logical, whether to set non-cortex data to NA

### Value

dataframe with concatenated vertex values. Each column contains the values for one measure, concatenated for all subjects. WARNING: This dataframe can get large if you have many subjects.

### See Also

Other concatenation functions: [group.concat.measures.standard\(\)](#)

**Examples**

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
subjects_list = c('subject1', 'subject2');
cm = group.concat.measures.native(subjects_dir, subjects_list,
  c("thickness", "area"), "lh");
```

---

```
group.concat.measures.standard
```

*Concatenate standard space data for a group of subjects.*

---

**Description**

A measure is something like 'thickness' or 'area'. This function concatenates the standard space data for all subjects into a single long vector for each measure. A dataframe is then created, in which each column is one such vector. This can be used to compute the correlation between measures on vertex level, for example.

**Usage**

```
group.concat.measures.standard(
  subjects_dir,
  subjects_list,
  measures,
  hemi,
  fwhm_per_measure,
  cortex_only = FALSE
)
```

**Arguments**

subjects_dir,	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subjects_list,	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
measures,	vector of strings. Names of the vertex-wise morphometry measures. E.g., c("area", "thickness"). Used to construct the names of the morphometry file to be loaded. The data of each measure will be one column in the resulting dataframe.
hemi,	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
fwhm_per_measure,	vector of strings. The fwhm settings to use, for each measure. If this is a string instead of a vector of strings, the same fwhm will be used for all measures.
cortex_only	logical, whether to set non-cortex data to NA

**Value**

dataframe with concatenated vertex values. Each column contains the values for one measure, concatenated for all subjects. The column names are a concatenation of the measure, "\_fwhm", and the fwhm for that measure. **WARNING:** This dataframe can get large if you have many subjects.

**See Also**

Other concatenation functions: [group.concat.measures.native\(\)](#)

**Examples**

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
subjects_list = c('subject1', 'subject2');
cm = group.concat.measures.standard(subjects_dir, subjects_list,
  c("thickness", "area"), "lh", "10");
```

---

group.label

*Retrieve label data for a group of subjects.*

---

**Description**

Load a label (like 'label/lh.cortex.label') for a group of subjects from disk. Uses knowledge about the FreeSurfer directory structure to load the correct file.

**Usage**

```
group.label(
  subjects_dir,
  subjects_list,
  label,
  hemi,
  return_one_based_indices = TRUE
)
```

**Arguments**

subjects\_dir, string. The FreeSurfer SUBJECTS\_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.

subjects\_list, vector of strings. The subject identifiers.

label, string. Name of the label file, without the hemi part (if any), but including the '.label' suffix. E.g., 'cortex.label' for '?h.cortex.label'

hemi, string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.

return\_one\_based\_indices, logical. Whether the indices should be 1-based. Indices are stored zero-based in the file, but R uses 1-based indices. Defaults to TRUE, which means that 1 will be added to all indices read from the file before returning them.

### Value

named list of integer vectors with label data: Each name is a subject identifier from subjects\_list, and the values are lists of the vertex indices in the respective label. See 'return\_one\_based\_indices' for important information.

### See Also

Other label data functions: `labeldata.from.mask()`, `mask.from.labeldata.for.hemi()`, `subject.label()`

### Examples

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
subjects_list = c("subject1", "subject2");
labels = group.label(subjects_dir, subjects_list, 'cortex.label', "lh");
```

---

group.label.from.annot

*Extract a region from an atlas annotation as a label for a group of subjects.*

---

### Description

The returned label can be used to mask morphometry data, e.g., to set the values of a certain region to NaN or to extract only values from a certain region.

### Usage

```
group.label.from.annot(
  subjects_dir,
  subjects_list,
  hemi,
  atlas,
  region,
  return_one_based_indices = TRUE,
  invert = FALSE,
  error_on_invalid_region = TRUE
)
```

**Arguments**

subjects_dir,	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subjects_list,	vector of string. The subject identifiers.
hemi,	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
atlas,	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
region,	string. A valid region name for the annotation, i.e., one of the regions of the atlas.
return_one_based_indices,	logical. Whether the indices should be 1-based. Indices are stored zero-based in label files, but R uses 1-based indices. Defaults to TRUE.
invert,	logical. If TRUE, return the indices of all vertices which are NOT part of the region. Defaults to FALSE.
error_on_invalid_region,	logical. Whether to throw an error if the given region does not appear in the region list of the annotation. If set to FALSE, this will be ignored and an empty vertex list will be returned. Defaults to TRUE.

**Value**

named list of integer vectors with label data: for each subject, the list of vertex indices in the label.

**See Also**

Other atlas functions: [get.atlas.region.names\(\)](#), [group.agg.atlas.native\(\)](#), [group.agg.atlas.standard\(\)](#), [group.annot\(\)](#), [label.from.annotdata\(\)](#), [label.to.annot\(\)](#), [regions.to.ignore\(\)](#), [spread.values.over.annot\(\)](#), [spread.values.over.hemi\(\)](#), [spread.values.over.subject\(\)](#), [subject.annot\(\)](#), [subject.atlas.agg\(\)](#), [subject.label.from.annot\(\)](#), [subject.lobes\(\)](#)

---

group.morph.agg.native

*Aggregate native space morphometry data over one hemisphere for a group of subjects.*

---

**Description**

Compute the mean (or other aggregates) over all vertices of a subject from native space morphometry data (like 'surf/lh.area'). Uses knowledge about the FreeSurfer directory structure to load the correct file.



**Usage**

```
group.morph.agg.native(
  subjects_dir,
  subjects_list,
  measure,
  hemi,
  agg_fun = mean,
  cast = TRUE,
  format = "curv",
  cortex_only = FALSE,
  agg_fun_extra_params = NULL
)
```

**Arguments**

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subjects_list	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
measure,	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
hemi,	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
agg_fun	function. An R function that aggregates data, typically <code>max</code> , <code>mean</code> , <code>min</code> or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to <code>mean</code> .
cast	Whether a separate 'hemi' column should exist.
format	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
cortex_only	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>not</i> part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subjects. Also note that the aggregation function will need to be able to cope with NA values if you set this to TRUE. You can use 'agg_fun_extra_params' if needed to achieve that, depending on the function. For example, if you use the <code>mean</code> function, you could set <code>agg_fun_extra_params=list("na.rm"=TRUE)</code> to get the mean of the vertices which are not masked. Defaults to FALSE.
agg_fun_extra_params	named list, extra parameters to pass to the aggregation function 'agg_fun' besides the loaded morphometry data. This is useful if you have masked the data and need to ignore NA values in the <code>agg_fun</code> .

**Value**

dataframe with aggregated values for all subjects, with 3 columns and n rows, where n is the number of subjects. The 3 columns are 'subject\_id', 'hemi', and '<measure>' (e.g., "thickness"), the latter

contains the aggregated data.

### See Also

Other global aggregation functions: [group.morph.agg.standard\(\)](#), [group.multimorph.agg.native\(\)](#), [group.multimorph.agg.standard\(\)](#)

### Examples

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
subjects_list = c("subject1", "subject2");
fulldata = group.morph.agg.native(subjects_dir, subjects_list, "thickness", "lh");
head(fulldata);
```

---

group.morph.agg.standard

*Aggregate standard space (fsaverage) morphometry data over one hemisphere for a group of subjects.*

---

### Description

Compute the mean (or other aggregates) over all vertices of a subject from standard space morphometry data (like 'surf/lh.area.fwhm10.fsaverage.mgh'). Uses knowledge about the FreeSurfer directory structure to load the correct file.

### Usage

```
group.morph.agg.standard(
  subjects_dir,
  subjects_list,
  measure,
  hemi,
  fwhm,
  agg_fun = mean,
  template_subject = "fsaverage",
  format = "mgh",
  cast = TRUE,
  cortex_only = FALSE,
  agg_fun_extra_params = NULL
)
```

**Arguments**

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subjects_list	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
measure,	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
hemi,	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
fwhm	string. Smoothing as string, e.g. '10' or '25'.
agg_fun	function. An R function that aggregates data, typically <code>max</code> , <code>mean</code> , <code>min</code> or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to <code>mean</code> .
template_subject	string. Template subject name, defaults to 'fsaverage'.
format	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
cast	Whether a separate 'hemi' column should exist.
cortex_only	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>not</i> part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subjects. Also not that the aggregation function will need to be able to cope with NA values if you set this to TRUE. You can use 'agg_fun_extra_params' if needed to achieve that, depending on the function. For example, if you use the <code>mean</code> function, you could set <code>agg_fun_extra_params=list("na.rm"=TRUE)</code> to get the mean of the vertices which are not masked. Defaults to FALSE.
agg_fun_extra_params	named list, extra parameters to pass to the aggregation function 'agg_fun' besides the loaded morphometry data. This is useful if you have masked the data and need to ignore NA values in the <code>agg_fun</code> .

**Value**

dataframe with aggregated values for all subjects, with 2 columns and n rows, where n is the number of subjects. The 2 columns are 'subject\_id' and '<hemi>.<measure>' (e.g., "lh.thickness"), the latter contains the aggregated data.

**See Also**

Other global aggregation functions: `group.morph.agg.native()`, `group.multimorph.agg.native()`, `group.multimorph.agg.standard()`

---

```
group.morph.agg.standard.vertex
```

*Aggregate standard space morphometry data over subjects.*

---

## Description

Aggregate vertex-wise values over subjects, leading to one aggregated measure per vertex.

## Usage

```
group.morph.agg.standard.vertex(
  subjects_dir,
  subjects_list,
  measure,
  hemi,
  fwhm,
  agg_fun = mean,
  template_subject = "fsaverage",
  format = "mgh",
  cortex_only = FALSE,
  agg_fun_extra_params = NULL,
  split_by_hemi = FALSE
)
```

## Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subjects_list	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
measure	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
hemi	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
fwhm	string. Smoothing as string, e.g. '10' or '25'.
agg_fun	function. An R function that aggregates data, typically <code>max</code> , <code>mean</code> , <code>min</code> or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to <code>mean</code> .
template_subject	string. Template subject name, defaults to 'fsaverage'.
format	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
cortex_only	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>not</i> part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting

this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subjects. Also not that the aggregation function will need to be able to cope with NA values if you set this to TRUE. You can use 'agg\_fun\_extra\_params' if needed to achieve that, depending on the function. For example, if you use the `mean` function, you could set `agg_fun_extra_params=list("na.rm"=TRUE)` to get the mean of the vertices which are not masked. Defaults to FALSE.

`agg_fun_extra_params`

named list, extra parameters to pass to the aggregation function 'agg\_fun' besides the loaded morphometry data. This is useful if you have masked the data and need to ignore NA values in the `agg_fun`.

`split_by_hemi` logical, whether to return a hemilist

### See Also

Other aggregation functions: [group.agg.atlas.native\(\)](#), [group.agg.atlas.standard\(\)](#), [subject.atlas.agg\(\)](#)

---

`group.morph.native`      *Retrieve native space morphometry data for a group of subjects.*

---

### Description

Load native space morphometry data (like 'surf/lh.area') for a group of subjects from disk. Uses knowledge about the FreeSurfer directory structure to load the correct file.

### Usage

```
group.morph.native(
  subjects_dir,
  subjects_list,
  measure,
  hemi,
  format = "curv",
  cortex_only = FALSE
)
```

### Arguments

`subjects_dir`, string. The FreeSurfer SUBJECTS\_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.

`subjects_list`, vector of strings. The subject identifiers.

`measure`, string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.

`hemi`, string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.

format, string. One of 'mgh', 'mgz', 'curv'. Defaults to 'curv'.

cortex\_only logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are *\*not\** part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subjects. Defaults to FALSE.

### Value

named list with native space morph data, the names are the subject identifiers from the subjects\_list, and the values are morphometry data vectors (of different length, as each subject has a different vertex count in native space).

### See Also

Other morphometry data functions: [apply.label.to.morphdata\(\)](#), [apply.labeldata.to.morphdata\(\)](#), [group.morph.standard\(\)](#), [subject.morph.native\(\)](#), [subject.morph.standard\(\)](#)

### Examples

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
subjects_list = c("subject1", "subject2");
data = group.morph.native(subjects_dir, subjects_list, "thickness", "lh");
```

---

group.morph.standard *Retrieve standard space morphometry data for a group of subjects.*

---

### Description

Load standard space morphometry data (like 'surf/lh.area') for a group of subjects from disk. Uses knowledge about the FreeSurfer directory structure to load the correct file.

### Usage

```
group.morph.standard(
  subjects_dir,
  subjects_list,
  measure,
  hemi = "both",
  fwhm = "10",
  template_subject = "fsaverage",
  format = "mgh",
  cortex_only = FALSE,
```

```

    df = FALSE
  )

```

### Arguments

`subjects_dir`, string. The FreeSurfer SUBJECTS\_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.

`subjects_list`, vector of strings. The subject identifiers.

`measure`, string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.

`hemi`, string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.

`fwhm`, string. Smoothing as string, e.g. '10' or '25'.

`template_subject`, string. Template subject name, defaults to 'fsaverage'.

`format`, string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.

`cortex_only` logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are *not* part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the template subject. Defaults to FALSE.

`df` logical, whether to return a dataframe instead of the named list. The dataframe will have one subject per column, and *n* rows, where *n* is the number of vertices of the template subject surface.

### Value

named list with standard space morph data, the names are the subject identifiers from the `subjects_list`, and the values are morphometry data vectors (all with identical length, the data is mapped to a template subject).

### See Also

Other morphometry data functions: [apply.label.to.morphdata\(\)](#), [apply.labeldata.to.morphdata\(\)](#), [group.morph.native\(\)](#), [subject.morph.native\(\)](#), [subject.morph.standard\(\)](#)

### Examples

```

## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
subjects_list = c("subject1", "subject2");
fulldata = group.morph.standard(subjects_dir, subjects_list, "thickness", "lh", fwhm='10');
mean(fulldata$subject1);

cortexdata = group.morph.standard(subjects_dir, subjects_list, "thickness",

```

```

    "lh", fwhm='10', cortex_only=FALSE);
mean(cortexdata$subject1, na.rm=TRUE);

## End(Not run)

```

---

```
group.multimorph.agg.native
```

*Aggregate native space morphometry data for multiple measures over hemispheres for a group of subjects.*

---

### Description

Compute the mean (or other aggregates) over all vertices of a subject from native space morphometry data (like 'surf/lh.area'). You can specify several measures and hemispheres. Uses knowledge about the FreeSurfer directory structure to load the correct files.

### Usage

```

group.multimorph.agg.native(
  subjects_dir,
  subjects_list,
  measures,
  hemis,
  agg_fun = mean,
  format = "curv",
  cast = TRUE,
  cortex_only = FALSE,
  agg_fun_extra_params = NULL
)

```

### Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subjects_list	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
measures	vector of strings. Names of the vertex-wise morphometry measures. E.g., c("area", "thickness"). Used to construct the names of the morphometry file to be loaded.
hemis	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
agg_fun	function. An R function that aggregates data, typically <code>max</code> , <code>mean</code> , <code>min</code> or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to <code>mean</code> .
format	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.



cast	Whether a separate 'hemi' column should exist.
cortex_only	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>*not*</i> part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subjects. Also not that the aggregation function will need to be able to cope with NA values if you set this to TRUE. You can use 'agg_fun_extra_params' if needed to achieve that, depending on the function. For example, if you use the <a href="#">mean</a> function, you could set <code>agg_fun_extra_params=list("na.rm"=TRUE)</code> to get the mean of the vertices which are not masked. Defaults to FALSE.
agg_fun_extra_params	named list, extra parameters to pass to the aggregation function 'agg_fun' besides the loaded morphometry data. This is useful if you have masked the data and need to ignore NA values in the <code>agg_fun</code> .

### Value

dataframe with aggregated values over all measures and hemis for all subjects, with m columns and n rows, where n is the number of subjects. The m columns are 'subject\_id' and '<hemi>.<measure>' (e.g., "lh.thickness") for all combinations of hemi and measure, the latter contains the aggregated data.

### See Also

Other global aggregation functions: [group.morph.agg.native\(\)](#), [group.morph.agg.standard\(\)](#), [group.multimorph.agg.standard\(\)](#)

### Examples

```
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
subjects_list = c("subject1", "subject2")
data = group.multimorph.agg.native(subjects_dir, subjects_list, c("thickness", "area"),
  c("lh", "rh"), cast=FALSE, cortex_only=TRUE, agg_fun=mean,
  agg_fun_extra_params=list("na.rm"=TRUE));
head(data);
```

---

group.multimorph.agg.standard

*Aggregate standard space (fsaverage) morphometry data for multiple measures over hemispheres for a group of subjects.*

---

**Description**

Compute the mean (or other aggregates) over all vertices of a subject from standard space morphometry data (like 'surf/lh.area.fwhm10.fsaverage.mgh'). You can specify several measures and hemispheres. Uses knowledge about the FreeSurfer directory structure to load the correct files.

**Usage**

```
group.multimorph.agg.standard(
  subjects_dir,
  subjects_list,
  measures,
  hemis,
  fwhm,
  agg_fun = mean,
  template_subject = "fsaverage",
  format = "mgh",
  cast = TRUE,
  cortex_only = FALSE,
  agg_fun_extra_params = NULL
)
```

**Arguments**

<code>subjects_dir</code> ,	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
<code>subjects_list</code> ,	string vector. A vector of subject identifiers that match the directory names within <code>subjects_dir</code> .
<code>measures</code> ,	vector of strings. Names of the vertex-wise morphometry measures. E.g., <code>c("area", "thickness")</code> . Used to construct the names of the morphometry file to be loaded.
<code>hemis</code> ,	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
<code>fwhm</code> ,	string. Smoothing as string, e.g. '10' or '25'.
<code>agg_fun</code> ,	function. An R function that aggregates data, typically <code>max</code> , mean, min or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to mean.
<code>template_subject</code> ,	string. Template subject name, defaults to 'fsaverage'.
<code>format</code> ,	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
<code>cast</code> ,	Whether a separate 'hemi' column should exist.
<code>cortex_only</code>	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>not</i> part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subjects.

Also not that the aggregation function will need to be able to cope with NA values if you set this to TRUE. You can use 'agg\_fun\_extra\_params' if needed to achieve that, depending on the function. For example, if you use the `mean` function, you could set `agg_fun_extra_params=list("na.rm"=TRUE)` to get the mean of the vertices which are not masked. Defaults to FALSE.

`agg_fun_extra_params`

named list, extra parameters to pass to the aggregation function 'agg\_fun' besides the loaded morphometry data. This is useful if you have masked the data and need to ignore NA values in the `agg_fun`.

### Value

dataframe with aggregated values over all measures and hemis for all subjects, with `m` columns and `n` rows, where `n` is the number of subjects. The `m` columns are 'subject\_id' and '<hemi>.<measure>' (e.g., "lh.thickness") for all combinations of hemi and measure, the latter contains the aggregated data.

### See Also

Other global aggregation functions: [group.morph.agg.native\(\)](#), [group.morph.agg.standard\(\)](#), [group.multimorph.agg.native\(\)](#)

---

hasIn

*Check for values in nested named lists*

---

### Description

Check for values in nested named lists

### Usage

```
hasIn(named_list, listkeys)
```

### Arguments

<code>named_list</code>	a named list
<code>listkeys</code>	vector of character strings, the nested names of the lists

### Value

whether a non-NULL value exists at the path

### Examples

```
data = list("regions"=list("frontal"=list("thickness"=2.3, "area"=2345)));
hasIn(data, c("regions", "nosuchregion")); # FALSE
```

---

`hemilist.derive.hemi`    *Derive 'hemi' string from the data in a hemilist*

---

**Description**

Derive 'hemi' string from the data in a hemilist

**Usage**

```
hemilist.derive.hemi(hemilist)
```

**Arguments**

`hemilist`            hemilist, an existing hemilist

**Value**

character string, one of 'lh', 'rh' or 'both'

---

`hemilist.get.combined.data`  
*Get combined data of hemi list*

---

**Description**

Get combined data of hemi list

**Usage**

```
hemilist.get.combined.data(hemi_list)
```

**Arguments**

`hemi_list`            named list, can have entries 'lh' and/or 'rh'

**Value**

the data combined with `c`, or NULL if both entries are NULL.

---

hemilist.unwrap	<i>Unwrap hemi data from a named hemi list.</i>
-----------------	---

---

**Description**

Unwrap hemi data from a named hemi list.

**Usage**

```
hemilist.unwrap(hemi_list, hemi = NULL, allow_null_list = FALSE)
```

**Arguments**

hemilist	named list, can have entries 'lh' and/or 'rh'
hemi	character string, the hemi data name to retrieve from the list. Can be NULL if the list only has a single entry.
allow_null_list	logical, whether to silently return NULL instead of raising an error if 'hemilist' is NULL

**Value**

the data

---

hemilist.wrap	<i>Wrap data into a named hemi list.</i>
---------------	--

---

**Description**

Wrap data into a named hemi list.

**Usage**

```
hemilist.wrap(data, hemi, hemilist = NULL)
```

**Arguments**

data	something to wrap, typically some data for a hemisphere, e.g., a vector of morphometry data values. If NULL, the name will not be created.
hemi	character string, one of 'lh' or 'rh'. The name to use for the data in the returned list.
hemilist	optional hemilist, an existing hemilist to add the entry to. If left at the default value 'NULL', a new list will be created.

**Value**

named list, with the 'data' in the name given by parameter 'hemi'

---

is.fs.coloredmesh      *Check whether object is an fs.coloredmesh (S3)*

---

**Description**

Check whether object is an fs.coloredmesh (S3)

**Usage**

```
is.fs.coloredmesh(x)
```

**Arguments**

x                    any 'R' object

**Value**

TRUE if its argument is a coloredmesh (that is, has "fs.coloredmesh" amongst its classes) and FALSE otherwise.

---

is.fs.coloredvoxels      *Check whether object is an fs.coloredvoxels instance (S3)*

---

**Description**

Check whether object is an fs.coloredvoxels instance (S3)

**Usage**

```
is.fs.coloredvoxels(x)
```

**Arguments**

x                    any 'R' object

**Value**

TRUE if its argument is a fs.coloredvoxels instance (that is, has "fs.coloredvoxels" amongst its classes) and FALSE otherwise.

---

is.hemilist	<i>Check whether x is a hemilist</i>
-------------	--------------------------------------

---

**Description**

A hemilist is a named list with entries 'lh' and/or 'rh'.

**Usage**

```
is.hemilist(x)
```

**Arguments**

x                    any R object

**Value**

whether 'x' is a hemilist

---

label.border	<i>Compute border of a label.</i>
--------------	-----------------------------------

---

**Description**

Compute the border of a label (i.e., a subset of the vertices of a mesh). The border thickness can be specified. Useful to draw the outline of a region, e.g., a significant cluster on the surface or a part of a ROI from a brain parcellation.

**Usage**

```
label.border(  
  surface_mesh,  
  label,  
  inner_only = TRUE,  
  expand_inwards = 0L,  
  derive = FALSE  
)
```

**Arguments**

surface\_mesh    surface mesh, as loaded by [subject.surface](#) or [read.fs.surface](#).  
label           instance of class 'fs.label' or an integer vector, the vertex indices. This function only makes sense if they form a patch on the surface, but that is not checked.

inner_only	logical, whether only faces consisting only of label_vertices should be considered to be label faces. If FALSE, faces containing at least one label vertex will be used. Defaults to TRUE. Leave this alone if in doubt, especially if you want to draw several label borders which are directly adjacent on the surface.
expand_inwards	integer, border thickness extension. If given, once the border has been computed, it is extended by the given graph distance. It is guaranteed that the border only extends inwards, i.e., it will never extend to vertices which are not part of the label.
derive	logical, whether the returned result should also include the border edges and faces in addition to the border vertices. Takes longer if requested, defaults to FALSE.

### Value

the border as a list with the following entries: 'vertices': integer vector, the vertex indices of the border. Iff the parameter 'derive' is TRUE, the following two additional fields are included: 'edges': integer matrix of size (n, 2) for n edges. Each row defines an edge by its start and target vertex. 'faces': integer vector, the face indices of the border.

### See Also

Other surface mesh functions: [face.edges\(\)](#), [mesh.vertex.included.faces\(\)](#), [mesh.vertex.neighbors\(\)](#), [subject.surface\(\)](#), [vis.path.along.verts\(\)](#)

---

label.colFn	<i>A simple colormap function for binary colors.</i>
-------------	--

---

### Description

Useful for plotting labels.

### Usage

```
label.colFn(n = 2L, col_a = "#228B22", col_b = "#FFFFFF")
```

### Arguments

n	positive integer, the number of colors. Must be 1 or 2 for this function.
col_a	color string, the foreground color
col_b	color string, the background color

### Value

vector of 'n' RGB colorstrings



---

label.colFn.inv      *A simple colormap function for binary colors.*

---

### Description

Useful for plotting labels.

### Usage

```
label.colFn.inv(n = 2L, col_a = "#228B22", col_b = "#FFFFFF")
```

### Arguments

n                    positive integer, the number of colors. Must be 1 or 2 for this function.  
col\_a                color string, the foreground color  
col\_b                color string, the background color

### Value

vector of 'n' RGB colorstrings

---

label.from.annotdata      *Extract a region from an annotation as a label.*

---

### Description

The returned label can be used to mask morphometry data, e.g., to set the values of a certain region to 'NaN' or to extract only values from a certain region.

### Usage

```
label.from.annotdata(  
  annotdata,  
  region,  
  return_one_based_indices = TRUE,  
  invert = FALSE,  
  error_on_invalid_region = TRUE  
)
```

**Arguments**

- annotdata,      annotation. An annotation for one hemisphere, as returned by `subject.annot`. This must be the loaded data, not a path to a file.
- region,          string. A valid region name for the annotation, i.e., one of the regions of the atlas used to create the annotation.
- return\_one\_based\_indices,      logical. Whether the indices should be 1-based. Indices are stored zero-based in label files, but R uses 1-based indices. Defaults to TRUE.
- invert,          logical. If TRUE, return the indices of all vertices which are NOT part of the region. Defaults to FALSE.
- error\_on\_invalid\_region,      logical. Whether to throw an error if the given region does not appear in the region list of the annotation. If set to FALSE, this will be ignored and an empty vertex list will be returned. Defaults to TRUE.

**Value**

integer vector with label data: the list of vertex indices in the label. See `'return_one_based_indices'` for important information.

**See Also**

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.annot()`, `group.label.from.annot()`, `label.to.annot()`, `regions.to.ignore()`, `spread.values.over.annot()`, `spread.values.over.hemi()`, `spread.values.over.subject()`, `subject.annot()`, `subject.atlas.agg()`, `subject.label.from.annot()`, `subject.lobes()`

---

label.to.annot	<i>Merge several labels into an annotation</i>
----------------	--

---

**Description**

Merge several labels and a colortable into an annotation.

**Usage**

```
label.to.annot(
  label_vertices_by_region,
  num_vertices_in_surface,
  colortable_df = NULL,
  index_of_unknown_region = 1L
)
```

**Arguments**

- `label_vertices_by_region`  
named list of integer vectors, the keys are strings which define region names, and the values are integer vectors: the vertex indices of the region.
- `num_vertices_in_surface`  
integer, total number of vertices in the surface mesh
- `colortable_df` NULL or dataframe, a colortable. It must contain the columns 'struct\_name', 'r', 'g', 'b', and 'a'. All other columns will be derived if missing. The entries in 'struct\_name' must match keys from the 'label\_vertices\_by\_region' parameter. There must be one more row in here than there are labels. This row identifies the 'unknown' region (see also parameter 'index\_of\_unknown\_region'). If NULL, a colortable will be auto-generated.
- `index_of_unknown_region`  
positive integer, the index of the row in 'colortable\_df' that defines the 'unknown' or 'background' region to which all vertices will be assigned which are *not* part of any of the given labels.

**Value**

an annotation, see [read.fs.annot](#) for details.

**See Also**

Other atlas functions: [get.atlas.region.names\(\)](#), [group.agg.atlas.native\(\)](#), [group.agg.atlas.standard\(\)](#), [group.annot\(\)](#), [group.label.from.annot\(\)](#), [label.from.annotdata\(\)](#), [regions.to.ignore\(\)](#), [spread.values.over.annot\(\)](#), [spread.values.over.hemi\(\)](#), [spread.values.over.subject\(\)](#), [subject.annot\(\)](#), [subject.atlas.agg\(\)](#), [subject.label.from.annot\(\)](#), [subject.lobes\(\)](#)

**Examples**

```
# Create two labels. Real-word labels would have more vertices, of course.
label1 = c(46666, 46777);
label2 = c(99888, 99889);
label_vertices = list("region1"=label1, "region2"=label2);
colortable_df = data.frame("struct_index"=seq(0, 2),
  "struct_name"=c("unknown", "region1", "region2"),
  "r"=c(255L, 255L, 0L), "g"=c(255L, 0L, 255L), "b"=c(255L, 0L, 0L), "a"=c(0L, 0L, 0L));
annot = label.to.annot(label_vertices, 100000, colortable_df);
```

---

labeldata.from.mask    *Create labeldata from a mask.*

---

**Description**

Create labeldata from a mask. This function is trivial and only calls [which](#) after performing basic sanity checks.

**Usage**

```
labeldata.from.mask(mask, invert = FALSE)
```

**Arguments**

mask	a logical vector
invert	Whether to report the inverse the mask before determining the indices. Defaults to FALSE.

**Value**

labeldata. The list of indices which are TRUE in the mask (or the ones which FALSE if 'invert' is TRUE).

**See Also**

Other label data functions: [group.label\(\)](#), [mask.from.labeldata.for.hemi\(\)](#), [subject.label\(\)](#)

---

list\_optional\_data     *Get file names available in package cache.*

---

**Description**

Get file names of optional data files which are available in the local package cache. You can access these files with `get_optional_data_file()`.

**Usage**

```
list_optional_data()
```

**Value**

vector of strings. The file names available, relative to the package cache.

---

```
mask.from.labeldata.for.hemi
```

*Create a binary mask from labels.*

---

## Description

Create a binary mask for the data of a single hemisphere from one or more labels. A label contains the vertex indices which are part of it, but often having a mask in more convenient.

## Usage

```
mask.from.labeldata.for.hemi(  
  labels,  
  num_vertices_in_hemi,  
  invert_labels = FALSE,  
  existing_mask = NULL  
)
```

## Arguments

labels	list of labels. A label is just a vector of vertex indices. It can be created manually, but is typically loaded from a label file using <a href="#">subject.label</a> .
num_vertices_in_hemi	integer. The number of vertices of the surface for which the mask is created. This must be for a single hemisphere.
invert_labels	logical, whether to invert the label data.
existing_mask	an existing mask to modify or NULL. If it is NULL, a new mask will be created before applying any labels, and the values set during initialization of this new mask are the negation of the 'invert_label' parameter. Defaults to NULL.

## Value

logical vector. The mask. It contains a logical value for each vertex. By default, the vertex indices from the labels are FALSE and the rest are TRUE, but this can be changed with the parameter 'invert\_labels'.

## See Also

Other label data functions: [group.label\(\)](#), [labeldata.from.mask\(\)](#), [subject.label\(\)](#)

Other mask functions: [coloredmesh.from.mask\(\)](#), [vis.mask.on.subject\(\)](#)

## Examples

```
fsbrain::download_optional_data();  
  
# Define the data to use:
```

```

subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
subject_id = 'subject1';
surface = 'white';
hemi = 'both';
atlas = 'aparc';
region = 'bankssts';

# Create a mask from a region of an annotation:
lh_annot = subject.annot(subjects_dir, subject_id, 'lh', atlas);
rh_annot = subject.annot(subjects_dir, subject_id, 'rh', atlas);
lh_label = label.from.annotdata(lh_annot, region);
rh_label = label.from.annotdata(rh_annot, region);
lh_mask = mask.from.labeldata.for.hemi(lh_label, length(lh_annot$vertices));
rh_mask = mask.from.labeldata.for.hemi(rh_label, length(rh_annot$vertices));

# Edit the mask: add the vertices from another region to it:
region2 = 'medialorbitofrontal';
lh_label2 = label.from.annotdata(lh_annot, region2);
rh_label2 = label.from.annotdata(rh_annot, region2);
lh_mask2 = mask.from.labeldata.for.hemi(lh_label2, length(lh_annot$vertices),
existing_mask = lh_mask);
rh_mask2 = mask.from.labeldata.for.hemi(rh_label2, length(rh_annot$vertices),
existing_mask = rh_mask);

```

---

mesh.vertex.neighbors *Compute neighborhood of a vertex*

---

## Description

Given a set of query vertex indices and a mesh *\*m\**, compute all vertices which are adjacent to the query vertices in the mesh. A vertex *\*u\** is *\*adjacent\** to another vertex *\*v\** iff there exists an edge *\*e = (u, v)\** in *\*m\**. While you could call this function repeatedly with the old output as its new input to extend the neighborhood, you should maybe use a proper graph library for this.

## Usage

```

mesh.vertex.neighbors(
  surface,
  source_vertices,
  k = 1L,
  restrict_to_vertices = NULL
)

```

## Arguments

surface            a surface as returned by functions like [subject.surface](#) or [read.fs.surface](#).

`source_vertices`  
Vector of source vertex indices.

`k`  
positive integer, how often to repeat the procedure and grow the neighborhood, using the output 'vertices' as the 'source\_vertices' for the next iteration. Warning: settings this to high values will be very slow for large meshes.

`restrict_to_vertices`  
integer vector of vertex indices. If given, the neighborhood growth will be limited to the given vertex indices. Defaults to NULL, which means the neighborhood is not restricted.

**Value**

the neighborhood as a list with two entries: "faces": integer vector, the face indices of all faces the source\_vertices are a part of. "vertices": integer vector, the unique vertex indices of all vertices of the faces in the 'faces' property. These vertex indices include the indices of the source\_vertices themselves.

**See Also**

Other surface mesh functions: [face.edges\(\)](#), [label.border\(\)](#), [mesh.vertex.included.faces\(\)](#), [subject.surface\(\)](#), [vis.path.along.verts\(\)](#)

---

<code>mkco.cluster</code>	<i>Return recommended 'makecmap_options' for diverging cluster data.</i>
---------------------------	--

---

**Description**

This function returns recommended visualization settings (a colormap function and suitable other settings) for the given type of data. The return value is meant to be passed as parameter 'makecmap\_options' to the vis.\* functions, e.g., [vis.subject.morph.native](#).

**Usage**

```
mkco.cluster()
```

**Value**

named list, visualization settings to be used as 'makecmap\_options' for diverging data.

**Note**

This uses a cyan blue red yellow colormap, which is popular for displaying clusters in neuroscience.

---

mkco.div	<i>Return recommended 'makecmap_options' for diverging data.</i>
----------	--

---

**Description**

This function returns recommended visualization settings (a colormap function and suitable other settings) for the given type of data. The return value is meant to be passed as parameter 'makecmap\_options' to the vis.\* functions, e.g., [vis.subject.morph.native](#).

**Usage**

```
mkco.div()
```

**Value**

named list, visualization settings to be used as 'makecmap\_options' for diverging data.

---

mkco.heat	<i>Return recommended 'makecmap_options' for sequential data with heatmap style.</i>
-----------	--

---

**Description**

This function returns recommended visualization settings (a colormap function and suitable other settings) for the given type of data. The return value is meant to be passed as parameter 'makecmap\_options' to the vis.\* functions, e.g., [vis.subject.morph.native](#).

**Usage**

```
mkco.heat()
```

**Value**

named list, visualization settings to be used as 'makecmap\_options' for sequential data with heatmap style.



---

mkco.seq	<i>Return recommended 'makecmap_options' for sequential data.</i>
----------	---

---

### Description

This function returns recommended visualization settings (a colormap function and suitable other settings) for the given type of data. The return value is meant to be passed as parameter 'makecmap\_options' to the vis.\* functions, e.g., `vis.subject.morph.native`.

### Usage

```
mkco.seq()
```

### Value

named list, visualization settings to be used as 'makecmap\_options' for sequential data.

---

<code>print.fs.coloredmesh</code>	<i>Print description of a brain coloredmesh (S3).</i>
-----------------------------------	---

---

### Description

Print description of a brain coloredmesh (S3).

### Usage

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

### Arguments

x	brain surface with class 'fs.coloredmesh'.
...	further arguments passed to or from other methods

---

```
print.fs.coloredvoxels
```

*Print description of fs.coloredvoxels (S3).*

---

### Description

Print description of fs.coloredvoxels (S3).

### Usage

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

### Arguments

x	brain voxel tris with class 'fs.coloredvoxels'.
...	further arguments passed to or from other methods

---

```
qc.for.group
```

*Perform data quality check based on computed region stats.*

---

### Description

Determine subjects that potentially failed segmentation, based on region-wise morphometry data. The stats can be computed from any kind of data, but something like area or volume most likely works best. The stats are based on the mean of the region values, so the measure should at least roughly follow a normal distribution.

### Usage

```
qc.for.group(subjects_dir, subjects_list, measure, atlas, hemi = "both", ...)
```

### Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subjects_list	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
measure	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
atlas	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKAtlas". Used to construct the name of the annotation file to be loaded.
hemi	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded. If set to 'both', combined data for 'lh' and 'rh' will be used.
...	parameters passed to <a href="#">qc.from.regionwise.df</a> .

**Value**

qc result as a hemilist, each entry contains a named list as returned by [qc.from.regionwise.df](#).

**See Also**

Other quality check functions: [qc.from.regionwise.df\(\)](#), [qc.from.segstats.table\(\)](#)

---

`qc.from.regionwise.df` *Perform data quality check based on a dataframe containing aggregated region-wise data.*

---

**Description**

Determine subjects that potentially failed segmentation, based on region-wise data. The data can be anything, but there must be one numerical value per subject per region.

**Usage**

```
qc.from.regionwise.df(
  rdf,
  z_threshold = 2.8,
  verbosity = 0L,
  num_bad_regions_allowed = 1L
)
```

**Arguments**

<code>rdf</code>	data.frame, the region data. The first column must contain the subject identifier, all other columns should contain numerical data for a single region. (Each row represents a subject.) This can be produced by calling <a href="#">group.agg.atlas.native</a> .
<code>z_threshold</code>	numerical, the cutoff value for considering a subject an outlier (in standard deviations).
<code>verbosity</code>	integer, controls the output to stdout. 0=off, 1=normal, 2=verbose.
<code>num_bad_regions_allowed</code>	integer, the number of regions in which subjects are allowed to be outliers without being reported as potentially failed segmentation

**Value**

named list with entries: `'failed_subjects'`: vector of character strings, the subject identifiers which potentially failed segmentation

**See Also**

Other quality check functions: [qc.for.group\(\)](#), [qc.from.segstats.table\(\)](#)

---

```
qc.from.segstats.tables
```

*Perform data quality check based on a segstats table.*

---

### Description

Determine subjects that potentially failed segmentation, based on segstats table data. The input table file must be a segmentation or parcellation table, generated by running the FreeSurfer tools 'aparcstats2table' or 'asegstats2table' for your subjects.

### Usage

```
qc.from.segstats.tables(filepath_lh, filepath_rh, ...)
```

### Arguments

filepath_lh	path to left hemisphere input file, a tab-separated file generated by running the FreeSurfer tools 'aparcstats2table' or 'asegstats2table'. The command line in the system shell would be something like 'aparcstats2table_bin -subjectsfile \$subjects_file -meas \$measure -hemi \$hemi -t \$aparc_output_table'.
filepath_rh	path to equivalent right hemisphere input file.
...	parameters passed to <a href="#">qc.from.regionwise.df</a> .

### Value

qc result as a hemilist, each entry contains a named list as returned by [qc.from.regionwise.df](#).

---

```
qc.vis.failcount.by.region
```

*Visualize the number of outlier subjects per region in your dataset.*

---

### Description

The function helps you to see which regions are affected the most by QC issues: for each region, it plots the number of subjects which are outliers in the region.

### Usage

```
qc.vis.failcount.by.region(
  qc_res,
  atlas,
  subjects_dir = fsaverage.path(),
  subject_id = "fsaverage",
  ...
)
```

**Arguments**

qc_res	hemilist of QC results, as returned by functions like <a href="#">qc.for.group</a> or <a href="#">qc.from.segstats.tables</a> .
atlas	string. The brain atlas to use. E.g., 'aparc' or 'aparc.a2009s'.
subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
...	extra parameters passed to <a href="#">vis.region.values.on.subject</a> . E.g., to change to interactive view, get a colorbar and better resolution, try: <code>draw_colorbar=T,rgloptions = rglot(),views='si'</code> .

**Note**

You can visualize this on any subject you like, 'fsaverage' is a typical choice. The atlas must be the one used during the QC step.

---

ras2vox_tkr	<i>The FreeSurfer default ras2vox_tkr matrix.</i>
-------------	---

---

**Description**

Applying this matrix to a FreeSurfer surface RAS coordinate (from a surface file like 'lh.white') will give you the voxel index (CRS) in a conformed FreeSurfer volume. The returned matrix is the inverse of the 'vox2ras\_tkr' matrix.

**Usage**

```
ras2vox_tkr()
```

**Value**

numeric 4x4 matrix, the FreeSurfer ras2vox\_tkr matrix.

**See Also**

Other surface and volume coordinates: [vox2ras\\_tkr\(\)](#)

**Examples**

```
# Compute the FreeSurfer CRS voxel index of surface RAS coordinate (0.0, 0.0, 0.0):
ras2vox_tkr() %*% c(0, 0, 0, 1);
# Show that the voxel at surface RAS corrd (0.0, 0.0, 0.0) is the one with CRS (128, 128, 128):
ras2vox_tkr() %*% c(0.0, 0.0, 0.0, 1);
```

---

read.colorcsv	<i>Read colors from CSV file.</i>
---------------	-----------------------------------

---

**Description**

Read colors from CSV file.

**Usage**

```
read.colorcsv(filepath)
```

**Arguments**

filepath          character string, path to a CSV file containing colors

**Value**

vector of hex color strings

---

read.md.demographics	<i>Read demographics file</i>
----------------------	-------------------------------

---

**Description**

Load a list of subjects and metadata from a demographics file, i.e., a tab-separated file containing an arbitrary number of columns, one of which must be the subject id.

**Usage**

```
read.md.demographics(  
  demographics_file,  
  column_names,  
  header = TRUE,  
  scale_and_center = FALSE,  
  sep = "",  
  report = FALSE,  
  stringsAsFactors = TRUE,  
  group_column_name = NULL  
)
```

## Arguments

demographics_file,	string. The path to the file.
column_names,	vector of strings. The column names to set in the returned dataframe. The length must match the number of columns in the file.
header,	logical. Whether the file starts with a header line.
scale_and_center,	logical. Whether to center and scale the data. Defaults to FALSE.
sep,	string. Separator passed to <a href="#">read.table</a> , defaults to tabulator.
report,	logical. Whether to write an overview, i.e., some descriptive statistics for each column, to STDOUT. Defaults to FALSE. See <a href="#">report.on.demographics</a> .
stringsAsFactors,	logical. Whether to convert strings in the input data to factors. Defaults to TRUE.
group_column_name,	string or NULL. If given, the column name of the group column. It must be a factor column with 2 levels. Enables group-comparison tests. Defaults to NULL.

## Value

a dataframe. The data in the file. String columns will be returned as factors, which you may want to adapt afterwards for the subject identifier column.

## See Also

Other metadata functions: [read.md.subjects\(\)](#)

## Examples

```
demographics_file =  
system.file("extdata", "demographics.tsv", package = "fsbrain", mustWork = TRUE);  
column_names = c("subject_id", "group", "age");  
demographics = read.md.demographics(demographics_file,  
column_names = column_names, report = FALSE);
```

---

read.md.subjects	<i>Read subjects file</i>
------------------	---------------------------

---

## Description

Load a list of subjects from a subjects file, i.e., a simple text file containing one subject name per line.

**Usage**

```
read.md.subjects(subjects_file, header = FALSE)
```

**Arguments**

subjects\_file character string, the path to the subjects file.  
 header logical, whether the file starts with a header line. Defaults to 'FALSE'.

**Value**

vector of strings, the subject identifiers.

**See Also**

Other metadata functions: [read.md.demographics\(\)](#)

**Examples**

```
subjects_file = system.file("extdata", "subjects.txt", package = "fsbrain", mustWork = TRUE);
subjects_list = read.md.subjects(subjects_file);
```

---

regions.to.ignore      *Give suggestions for regions to ignore for an atlas.*

---

**Description**

Give suggestions for regions to ignore for an atlas. These are regions for which many subjects do not have any vertices in them, or the Medial Wall and Unknown regions.

**Usage**

```
regions.to.ignore(atlas)
```

**Arguments**

atlas, string. The name of an atlas. Supported strings are 'aparc' and 'aparc.a2009s'.

**Value**

vector of strings, the region names.

**See Also**

Other atlas functions: [get.atlas.region.names\(\)](#), [group.agg.atlas.native\(\)](#), [group.agg.atlas.standard\(\)](#), [group.annot\(\)](#), [group.label.from.annot\(\)](#), [label.from.annotdata\(\)](#), [label.to.annot\(\)](#), [spread.values.over.annot\(\)](#), [spread.values.over.hemi\(\)](#), [spread.values.over.subject\(\)](#), [subject.annot\(\)](#), [subject.atlas.agg\(\)](#), [subject.label.from.annot\(\)](#), [subject.lobes\(\)](#)



## Examples

```
aparc_regions_ign = regions.to.ignore('aparc');
aparc_a2009s_regions_ign = regions.to.ignore('aparc.a2009s');
```

---

```
report.on.demographics
```

*Print a demographics report*

---

## Description

Print a demographics report

## Usage

```
report.on.demographics(  
  demographics_df,  
  group_column_name = NULL,  
  paired = FALSE  
)
```

## Arguments

demographics_df	a demographics data.frame, as returned by <a href="#">read.md.demographics</a> .
group_column_name,	string or NULL. If given, the column name of the group column. It must be a factor column with 2 levels. Enables group-comparison tests. Defaults to 'NULL'.
paired	Whether the data of the two groups is paired (repeated measurements). Only relevant if group_column_name is given and tests for group differences are included in the report. Defaults to 'FALSE'.

## Value

vector of character strings, the lines of the demographics report.

---

rglot	<i>Get rgl_options for testing.</i>
-------	-------------------------------------

---

**Description**

This function defines the figure size that is used during the unit tests.

**Usage**

```
rglot()
```

**Value**

named list, usable as 'rgl\_options' parameter for vis functions like [vis.subject.morph.native](#).

**Note**

This function is public so one can copy and paste unit test code into the R console, but you should not consider it part of the official functions and use it in your client code.

---

rglvoxels	<i>Draw 3D boxes at locations using rgl.</i>
-----------	--

---

**Description**

Draw 3D boxes at all given coordinates using rgl, analogous to [rgl.spheres](#). Constructs the coordinates for triangles making up the boxes, then uses [triangles3d](#) to render them.

**Usage**

```
rglvoxels(centers, r = 1, voxelcol = NULL, ...)
```

**Arguments**

centers	numerical matrix with 3 columns. Each column represents the x, y, z coordinates of a center at which to create a cube.
r	numerical vector or scalar, the cube edge length. This is the length of the axis-parallel edges of the cube. The vector must have length 1 (same edge length for all cubes), or the length must be identical to the number of rows in parameter 'centers'.
voxelcol	vector of rgb color strings for the individual voxels. Its length must be identical to <code>nrow(centers)</code> if given.
...	material properties, passed to <a href="#">triangles3d</a> . Example: <code>color = "#0000ff", lit=FALSE</code> .

**Value**

list of 'fs.coloredvoxels' instances, invisible. The function is called for the side effect of visualizing the data, and usually you can ignore the return value.

**Examples**

```
# Plot a 3D cloud of 500 red voxels:
centers = matrix(rnorm(500*3)*100, ncol=3);
rglvoxels(centers, voxelcol="red");
```

---

shift.hemis.apart	<i>Shift hemispheres apart.</i>
-------------------	---------------------------------

---

**Description**

Modify mesh coordinates of a hemilist of coloredmeshes to introduce a gap between the two hemispheres.

**Usage**

```
shift.hemis.apart(
  coloredmeshes_hl,
  shift_by = NULL,
  axis = 1L,
  hemi_order_on_axis = "lr",
  min_dist = 0
)
```

**Arguments**

coloredmeshes_hl	hemilist of coloredmeshes
shift_by	numerical vector of length 2, the amount by which to shift the hemis. The first value is for the left hemi, the second for the right hemi (values can be negative). Pass 'NULL' to determine the shift automatically from the mesh coordinates, and adapt 'hemi_order_on_axis' to define how that happens.
axis	positive integer, one of 1L, 2L or 3L. The axis on which to shift (x,y,z).
hemi_order_on_axis	character string, one of 'auto', 'auto_flipped', 'lr' or 'rl'. Defines how to determine the order of the hemis on the axes. This is ignored unless 'shift_by' is 'NULL'. The 'auto' setting assumes that the hemisphere with the smaller minimal vertex coordinate (on the given axis) comes first. Note that if the overlap (or shift) is extreme, this may not hold anymore. Therefore, the default value is 'lr', which works for FreeSurfer data. The 'auto_flipped' setting will always return the inverse of 'auto', so if 'auto' did not work, 'auto_flipped' will.

min\_dist            numerical scalar, the minimal distance of the hemispheres. Ignored unless 'shift\_by' is 'NULL'.

### Value

hemilist of coloredmeshes, the shifted meshes

---

spread.values.over.annot

*Spread a single value for a region to all region vertices.*

---

### Description

Given an annotation and a list of values (one per brain region), return data that has the values for each region mapped to all region vertices.

### Usage

```
spread.values.over.annot(
  annot,
  region_value_list,
  value_for_unlisted_regions = NaN,
  warn_on_unmatched_list_regions = FALSE,
  warn_on_unmatched_atlas_regions = FALSE
)
```

### Arguments

annot,            annotation. The result of calling fs.read.annot.

region\_value\_list,            named list of strings. Each name must be a region name from the annotation, and the value must be the value to spread to all region vertices.

value\_for\_unlisted\_regions,            numeric scalar. The value to assign to vertices which are part of atlas regions that are not listed in region\_value\_list. Defaults to NaN.

warn\_on\_unmatched\_list\_regions,            logical. Whether to print a warning when a region occurs in the region\_value\_list that is not part of the given atlas (and the value assigned to this region is thus ignored in the output file and data). Defaults to FALSE.

warn\_on\_unmatched\_atlas\_regions,            logical. Whether to print a warning when a region occurs in the atlas that is not part of the given region\_value\_list (and thus the vertices of the region will be assigned the value 'value\_for\_unlisted\_regions' in the output file and data). Defaults to FALSE.

**Value**

named list with following entries: "spread\_data": a vector of length n, where n is the number of vertices in the annotation. One could write this to an MGH or curv file for visualization. "regions\_not\_in\_annot": list of regions which are not in the annotation, but in the region\_value\_list. Their values were ignored.

**See Also**

Other atlas functions: [get.atlas.region.names\(\)](#), [group.agg.atlas.native\(\)](#), [group.agg.atlas.standard\(\)](#), [group.annot\(\)](#), [group.label.from.annot\(\)](#), [label.from.annotdata\(\)](#), [label.to.annot\(\)](#), [regions.to.ignore\(\)](#), [spread.values.over.hemi\(\)](#), [spread.values.over.subject\(\)](#), [subject.annot\(\)](#), [subject.atlas.agg\(\)](#), [subject.label.from.annot\(\)](#), [subject.lobes\(\)](#)

**Examples**

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
annot = subject.annot(subjects_dir, 'subject1', 'lh', 'aparc');
region_value_list = list("bankssts"=0.9, "precuneus"=0.7);
morph_like_data =
spread.values.over.annot(annot, region_value_list, value_for_unlisted_regions=0.0);
```

---

spread.values.over.hemi

*Spread the values in the region\_value\_list and return them for one hemisphere.*

---

**Description**

Given an atlas and a list that contains one value for each atlas region, create morphometry data in which all region vertices are assigned the value. Can be used to plot stuff like p-values or effect sizes onto brain regions.

**Usage**

```
spread.values.over.hemi(
  subjects_dir,
  subject_id,
  hemi,
  atlas,
  region_value_list,
  value_for_unlisted_regions = NA
)
```

**Arguments**

subjects_dir,	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id,	string. The subject identifier
hemi,	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
atlas,	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKAtlas". Used to construct the name of the annotation file to be loaded.
region_value_list,	named list. A list in which the names are atlas regions, and the values are the value to write to all vertices of that region. You can pass an unnamed list or vector, but then the length must exactly match the number of regions in the atlas, and the order must match the annotation file of the subject and hemisphere. Use with care, and keep in mind that some subjects do not have all regions.
value_for_unlisted_regions,	numeric scalar. The value to assign to vertices which are part of atlas regions that are not listed in region_value_list. Defaults to NaN.

**Value**

numeric vector containing the data.

**See Also**

Other atlas functions: [get.atlas.region.names\(\)](#), [group.agg.atlas.native\(\)](#), [group.agg.atlas.standard\(\)](#), [group.annot\(\)](#), [group.label.from.annot\(\)](#), [label.from.annotdata\(\)](#), [label.to.annot\(\)](#), [regions.to.ignore\(\)](#), [spread.values.over.annot\(\)](#), [spread.values.over.subject\(\)](#), [subject.annot\(\)](#), [subject.atlas.agg\(\)](#), [subject.label.from.annot\(\)](#), [subject.lobes\(\)](#)

**Examples**

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
region_value_list = list("bankssts"=0.9, "precuneus"=0.7);
morph_like_data =
  spread.values.over.hemi(subjects_dir, 'subject1', 'lh', 'aparc', region_value_list);
```

---

spread.values.over.subject

*Spread the values in the region\_value\_list and return them for one hemisphere.*

---

**Description**

Given an atlas and a list that contains one value for each atlas region, create morphometry data in which all region vertices are assigned the value. Can be used to plot stuff like p-values or effect sizes onto brain regions.

**Usage**

```
spread.values.over.subject(
    subjects_dir,
    subject_id,
    atlas,
    lh_region_value_list,
    rh_region_value_list,
    value_for_unlisted_regions = NaN
)
```

**Arguments**

`subjects_dir`, string. The FreeSurfer SUBJECTS\_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.

`subject_id`, string. The subject identifier

`atlas`, string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKAtlas". Used to construct the name of the annotation file to be loaded.

`lh_region_value_list`,  
named list. A list in which the names are atlas regions, and the values are the value to write to all vertices of that region. Applied to the left hemisphere.

`rh_region_value_list`,  
named list. A list in which the names are atlas regions, and the values are the value to write to all vertices of that region. Applied to the right hemisphere.

`value_for_unlisted_regions`,  
numeric scalar. The value to assign to vertices which are part of atlas regions that are not listed in `region_value_list`. Defaults to NaN.

**Value**

named list with entries 'lh' and 'rh'. Each value is a numeric vector containing the data for the respective hemisphere.

**See Also**

Other atlas functions: [get.atlas.region.names\(\)](#), [group.agg.atlas.native\(\)](#), [group.agg.atlas.standard\(\)](#), [group.annot\(\)](#), [group.label.from.annot\(\)](#), [label.from.annotdata\(\)](#), [label.to.annot\(\)](#), [regions.to.ignore\(\)](#), [spread.values.over.annot\(\)](#), [spread.values.over.hemi\(\)](#), [subject.annot\(\)](#), [subject.atlas.agg\(\)](#), [subject.label.from.annot\(\)](#), [subject.lobes\(\)](#)

## Examples

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
lh_region_value_list = list("bankssts"=0.9, "precuneus"=0.7);
rh_region_value_list = list("bankssts"=0.5);
morph_like_data =
  spread.values.over.subject(subjects_dir, 'subject1', 'aparc',
    lh_region_value_list, rh_region_value_list);
```

---

subject.annot	<i>Load an annotation for a subject.</i>
---------------	--

---

## Description

Load a brain surface annotation, i.e., a cortical parcellation based on an atlas, for a subject.

## Usage

```
subject.annot(subjects_dir, subject_id, hemi, atlas)
```

## Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier
hemi	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
atlas	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.

## Value

the annotation, as returned by `read.fs.annot`. It is a named list, entries are: "vertices" vector of n vertex indices, starting with 0. "label\_codes": vector of n integers, each entry is a color code, i.e., a value from the 5th column in the table structure included in the "colortable" entry (see below). "label\_names": the n brain structure names for the vertices, already retrieved from the colortable using the code. "hex\_colors\_rgb": Vector of hex color for each vertex. The "colortable" is another named list with 3 entries: "num\_entries": int, number of brain structures. "struct\_names": vector of strings, the brain structure names. "table": numeric matrix with num\_entries rows and 5 columns. The 5 columns are: 1 = color red channel, 2=color blue channel, 3=color green channel, 4=color alpha channel, 5=unique color code. "colortable\_df": The same information as a dataframe. Contains the extra columns "hex\_color\_string\_rgb" and "hex\_color\_string\_rgba" that hold the color as an RGB(A) hex string, like "#rrggbaa".



**See Also**

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.annot()`, `group.label.from.annot()`, `label.from.annotdata()`, `label.to.annot()`, `regions.to.ignore()`, `spread.values.over.annot()`, `spread.values.over.hemi()`, `spread.values.over.subject`, `subject.atlas.agg()`, `subject.label.from.annot()`, `subject.lobes()`

**Examples**

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
annot_lh = subject.annot(subjects_dir, "subject1", "lh", "aparc");
```

---

<code>subject.atlas.agg</code>	<i>Aggregate morphometry data over brain atlas regions for a subject.</i>
--------------------------------	---

---

**Description**

Aggregate morphometry data over brain atlas regions, e.g., compute the mean thickness value over all regions in an atlas.

**Usage**

```
subject.atlas.agg(
  vertex_morph_data,
  vertex_label_names,
  agg_fun = mean,
  requested_label_names = c()
)
```

**Arguments**

`vertex_morph_data`,  
 numeric vector. The morphometry data, one value per vertex. The morphometry data are typically loaded from an MGZ or curv format file with the `read.fs.curv` or `read.fs.mgh` functions.

`vertex_label_names`,  
 string vector. The region names for the vertices, one string per vertex. The region names are typically loaded from an annotation file with the `read.fs.annot` function.

`agg_fun`,  
 function. An R function that aggregates data, typically `max`, `mean`, `min` or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to `mean`.

requested\_label\_names,

string vector. The label (or region) names that you want to occur in the output. If not specified, all region names which occur in the data are used. If given, and one of the requested names does NOT occur in the data, it will occur in the output with aggregation value NaN. If given, and one of the names from the data does NOT occur in the requested list, it will NOT occur in the output. So if you specify this, the output dataframe will contain a row for a region if and only if it is in the requested list.

### Value

dataframe with aggregated values for all regions, with 2 columns and n rows, where n is the number of effective regions. The columns are: "region": string, contains the region name. "aggregated": numeric, contains the result of applying agg\_fun to the morphometry data in that region.

### See Also

Other aggregation functions: [group.agg.atlas.native\(\)](#), [group.agg.atlas.standard\(\)](#), [group.morph.agg.standard\(\)](#)

Other atlas functions: [get.atlas.region.names\(\)](#), [group.agg.atlas.native\(\)](#), [group.agg.atlas.standard\(\)](#), [group.annot\(\)](#), [group.label.from.annot\(\)](#), [label.from.annotdata\(\)](#), [label.to.annot\(\)](#), [regions.to.ignore\(\)](#), [spread.values.over.annot\(\)](#), [spread.values.over.hemi\(\)](#), [spread.values.over.subject\(\)](#), [subject.annot\(\)](#), [subject.label.from.annot\(\)](#), [subject.lobes\(\)](#)

### Examples

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
morph_data = subject.morph.native(subjects_dir, 'subject1', 'thickness', 'lh');
annot = subject.annot(subjects_dir, 'subject1', 'lh', 'aparc');
agg = subject.atlas.agg(morph_data, annot$label_names);
```

---

subject.filepath.morph.native

*Construct filepath of native space morphometry data file.*

---

### Description

Construct filepath of native space morphometry data file.

### Usage

```
subject.filepath.morph.native(
  subjects_dir,
  subject_id,
  measure,
```

```

    hemi,
    format = "curv",
    warn_if_nonexistent = FALSE,
    error_if_nonexistent = FALSE
)

```

### Arguments

`subjects_dir` string. The FreeSurfer SUBJECTS\_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.

`subject_id` string. The subject identifier

`measure` string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.

`hemi` string, one of 'lh' or 'rh'. The hemisphere name.

`format`, string. One of 'mgh', 'mgz', 'curv'. Defaults to 'curv'.

`warn_if_nonexistent` logical. Whether to print a warning if the file does not exist or cannot be accessed. Defaults to FALSE.

`error_if_nonexistent` logical. Whether to raise an error if the file does not exist or cannot be accessed. Defaults to FALSE.

### Value

string, the file path.

---

subject.filepath.morph.standard  
*Construct filepath of standard space morphometry data file.*

---

### Description

Construct filepath of standard space morphometry data file.

### Usage

```

subject.filepath.morph.standard(
  subjects_dir,
  subject_id,
  measure,
  hemi,
  fwhm = "10",
  template_subject = "fsaverage",
  format = "auto",
  warn_if_nonexistent = FALSE,
  error_if_nonexistent = FALSE
)

```

**Arguments**

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier
measure	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
hemi	string, one of 'lh' or 'rh'. The hemisphere name.
fwhm	string. Smoothing as string, e.g. '10' or '25'. Defaults to '10'.
template_subject	string. Template subject name, defaults to 'fsaverage'.
format	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
warn_if_nonexistent	logical. Whether to print a warning if the file does not exist or cannot be accessed. Defaults to FALSE.
error_if_nonexistent	logical. Whether to raise an error if the file does not exist or cannot be accessed. Defaults to FALSE.

**Value**

string, the file path.

---

subject.label	<i>Retrieve label data for a single subject.</i>
---------------	--

---

**Description**

Load a label (like 'label/lh.cortex.label') for a subject from disk. Uses knowledge about the FreeSurfer directory structure to load the correct file.

**Usage**

```
subject.label(
  subjects_dir,
  subject_id,
  label,
  hemi,
  return_one_based_indices = TRUE
)
```

## Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier
label	string. Name of the label file, without the hemi part. You can include the '.label' suffix. E.g., 'cortex.label' for '?h.cortex.label'. You can also pass just the label (e.g., 'cortex'): if the string does not end with the suffix '.label', that suffix gets added automatically.
hemi	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded. For 'both', see the information on the return value.
return_one_based_indices	logical. Whether the indices should be 1-based. Indices are stored zero-based in the file, but R uses 1-based indices. Defaults to TRUE, which means that 1 will be added to all indices read from the file before returning them.

## Value

integer vector with label data: the list of vertex indices in the label. See 'return\_one\_based\_indices' for important information. If parameter 'hemi' is set to 'both', a named list with entries 'lh' and 'rh' is returned, and the values of are the respective labels.

## See Also

Other label data functions: [group.label\(\)](#), [labeldata.from.mask\(\)](#), [mask.from.labeldata.for.hemi\(\)](#)

## Examples

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
cortex_lh = subject.label(subjects_dir, "subject1", "cortex.label", "lh");
```

---

subject.label.from.annot

*Extract a region from an atlas annotation as a label for a subject.*

---

## Description

The returned label can be used to mask morphometry data, e.g., to set the values of a certain region to NaN or to extract only values from a certain region.

**Usage**

```
subject.label.from.annot(
  subjects_dir,
  subject_id,
  hemi,
  atlas,
  region,
  return_one_based_indices = TRUE,
  invert = FALSE,
  error_on_invalid_region = TRUE
)
```

**Arguments**

`subjects_dir`, string. The FreeSurfer SUBJECTS\_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.

`subject_id`, string. The subject identifier.

`hemi`, string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.

`atlas`, string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.

`region`, string. A valid region name for the annotation, i.e., one of the regions of the atlas.

`return_one_based_indices`, logical. Whether the indices should be 1-based. Indices are stored zero-based in label files, but R uses 1-based indices. Defaults to TRUE.

`invert`, logical. If TRUE, return the indices of all vertices which are NOT part of the region. Defaults to FALSE.

`error_on_invalid_region`, logical. Whether to throw an error if the given region does not appear in the region list of the annotation. If set to FALSE, this will be ignored and an empty vertex list will be returned. Defaults to TRUE.

**Value**

integer vector with label data: the list of vertex indices in the label.

**See Also**

Other atlas functions: [get.atlas.region.names\(\)](#), [group.agg.atlas.native\(\)](#), [group.agg.atlas.standard\(\)](#), [group.annot\(\)](#), [group.label.from.annot\(\)](#), [label.from.annotdata\(\)](#), [label.to.annot\(\)](#), [regions.to.ignore\(\)](#), [spread.values.over.annot\(\)](#), [spread.values.over.hemi\(\)](#), [spread.values.over.subject](#), [subject.annot\(\)](#), [subject.atlas.agg\(\)](#), [subject.lobes\(\)](#)

---

subject.lobes	<i>Load labels representing brain lobes.</i>
---------------	--

---

### Description

This gives you labels that represent brain lobes for a subject. The lobe definition is based on the Desikan-Killiany atlas (Desikan \*et al.\*, 2010) as suggested on the FreeSurfer website at <https://surfer.nmr.mgh.harvard.edu/fswiki/CorticalParcellation>.

### Usage

```
subject.lobes(
  subjects_dir,
  subject_id,
  hemi = "both",
  include_cingulate = TRUE,
  as_annot = FALSE
)
```

### Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier
hemi	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the surface file to be loaded. For 'both', see the information on the return value.
include_cingulate	logical, whether to include the vertices of the cingulate in the lobes
as_annot	return a hemilist of annotations instead of the return value described in the *value* section

### Value

hemilist of integer vectors, the vectors represent vertex indices of the hemispheres, and each vertex is assigned one of the following values: '0'=no\_lobe, '1'=frontal, '2'=parietal, '3'=temporal, '4'=occipital.

### See Also

Other atlas functions: [get.atlas.region.names\(\)](#), [group.agg.atlas.native\(\)](#), [group.agg.atlas.standard\(\)](#), [group.annot\(\)](#), [group.label.from.annot\(\)](#), [label.from.annotdata\(\)](#), [label.to.annot\(\)](#), [regions.to.ignore\(\)](#), [spread.values.over.annot\(\)](#), [spread.values.over.hemi\(\)](#), [spread.values.over.subject\(\)](#), [subject.annot\(\)](#), [subject.atlas.agg\(\)](#), [subject.label.from.annot\(\)](#)

Other label functions: [apply.label.to.morphdata\(\)](#), [apply.labeldata.to.morphdata\(\)](#), [subject.mask\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.subject.label\(\)](#)

---

subject.mask                      *Compute a mask for a subject.*

---

### Description

Compute a binary vertex mask for the surface vertices of a subject. By defaults, the medial wall is masked.

### Usage

```
subject.mask(
    subjects_dir,
    subject_id,
    hemi = "both",
    from_label = "cortex",
    surf_num_verts = "white",
    invert_mask = TRUE
)
```

### Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier
hemi	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
from_label	string, the label file to use. Defaults to 'cortex', which will result in a mask of the medial wall versus cortex vertices.
surf_num_verts	string or integer. If an integer, interpreted as the number of vertices in the respective surface (lh or rh). If a character string, interpreted as a surface name, (e.g., 'white' or 'pial'), and the respective surface will be loaded to determine the number of vertices in it. If parameter 'hemi' is set to 'both' and you supply the vertex count as an integer, this can be a vector of length 2 if the surfaces have different vertex counts (the first entry for 'lh', the second for 'rh').
invert_mask	logical, whether to invert the mask. E.g., when the mask is loaded from the cortex labels, if this is set to FALSE, the cortex would be masked (set to 0 in the final mask). If you want <b>everything but the cortex</b> to be masked (set to 0), you should set this to 'TRUE'. Defaults to 'TRUE'.

### Value

the mask, a logical vector with the length of the vertices in the surface. If parameter 'hemi' is set to 'both', a named list with entries 'lh' and 'rh' is returned, and the values of are the respective masks.



**See Also**

Other label functions: [apply.label.to.morphdata\(\)](#), [apply.labeldata.to.morphdata\(\)](#), [subject.lobes\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.subject.label\(\)](#)

**Examples**

```
# Generate a binary mask of the medial wall. Wall vertices will
# be set to 0, cortex vertices will be set to 1.
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
mask = subject.mask(subjects_dir, "subject1");
# Print some information on the mask:
#cat(sprintf("lh: %d verts, %d in cortex, %d medial wall.\n", length(mask$lh),
# sum(mask$lh), (length(mask$lh)- sum(mask$lh))))
# Output: lh: 149244 verts, 140891 in cortex, 8353 medial wall.
# Now visualize the mask to illustrate that it is correct:
vis.mask.on.subject(subjects_dir, "subject1", mask$lh, mask$rh);
```

---

subject.morph.native    *Retrieve native space morphometry data for a single subject.*

---

**Description**

Load native space morphometry data (like 'surf/lh.area') for a subject from disk. Uses knowledge about the FreeSurfer directory structure to load the correct file.

**Usage**

```
subject.morph.native(
  subjects_dir,
  subject_id,
  measure,
  hemi,
  format = "curv",
  cortex_only = FALSE,
  split_by_hemi = FALSE
)
```

**Arguments**

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier
measure	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.

hemi	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
format	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'curv'.
cortex_only	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>not</i> part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subject. Defaults to FALSE.
split_by_hemi	logical, whether the returned data should be encapsulated in a named list, where the names are from 'lh' and 'rh', and the values are the respective data.

**Value**

vector with native space morph data, as returned by `read.fs.morph`.

**See Also**

Other morphometry data functions: `apply.label.to.morphdata()`, `apply.labeldata.to.morphdata()`, `group.morph.native()`, `group.morph.standard()`, `subject.morph.standard()`

**Examples**

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");

# Load the full data:
thickness_lh = subject.morph.native(subjects_dir, "subject1", "thickness", "lh");
mean(thickness_lh); # prints 2.437466

# Load the data again, but this time exclude the medial wall:
thickness_lh_cortex = subject.morph.native(subjects_dir, "subject1", "thickness",
  "lh", cortex_only=TRUE);
mean(thickness_lh_cortex, na.rm=TRUE); # prints 2.544132
vis.data.on.subject(subjects_dir, "subject1", thickness_lh_cortex, NULL);
```

---

subject.morph.standard

*Retrieve standard space morphometry data for a single subject.*

---

**Description**

Load standard space morphometry data (like 'surf/lh.area.fwhm10.fsaverage.mgh') for a subject from disk. Uses knowledge about the FreeSurfer directory structure to load the correct file.

**Usage**

```

subject.morph.standard(
  subjects_dir,
  subject_id,
  measure,
  hemi,
  fwhm = "10",
  template_subject = "fsaverage",
  format = "mgh",
  cortex_only = FALSE,
  split_by_hemi = FALSE
)

```

**Arguments**

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier
measure	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
hemi	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
fwhm	string. Smoothing as string, e.g. '10' or '25'.
template_subject	string. Template subject name, defaults to 'fsaverage'.
format	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
cortex_only	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>*not*</i> part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the template subject. Defaults to FALSE.
split_by_hemi	logical, whether the returned data should be encapsulated in a named list, where the names are from 'lh' and 'rh', and the values are the respective data.

**Value**

vector with standard space morph data

**See Also**

Other morphometry data functions: [apply.label.to.morphdata\(\)](#), [apply.labeldata.to.morphdata\(\)](#), [group.morph.native\(\)](#), [group.morph.standard\(\)](#), [subject.morph.native\(\)](#)

## Examples

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
thickness_lh = subject.morph.standard(subjects_dir, "subject1", "thickness", "lh", fwhm='10');
```

---

subject.surface      *Load a surface for a subject.*

---

## Description

Load a brain surface mesh for a subject.

## Usage

```
subject.surface(subjects_dir, subject_id, surface, hemi)
```

## Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier
surface	string. The surface name. E.g., "white", or "pial". Used to construct the name of the surface file to be loaded.
hemi	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the surface file to be loaded. For 'both', see the information on the return value.

## Value

the 'fs.surface' instance, as returned by [read.fs.surface](#). If parameter 'hemi' is set to 'both', a named list with entries 'lh' and 'rh' is returned, and the values of are the respective surfaces. The mesh data structure used in 'fs.surface' is a \*face index set\*.

## See Also

Other surface mesh functions: [face.edges\(\)](#), [label.border\(\)](#), [mesh.vertex.included.faces\(\)](#), [mesh.vertex.neighbors\(\)](#), [vis.path.along.verts\(\)](#)

## Examples

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
lh_white = subject.surface(subjects_dir, "subject1", "white", "lh");
```

---

subject.volume      *Read a brain volume.*

---

### Description

Load a brain volume (like 'mri/brain.mgz') for a subject from disk. Uses knowledge about the FreeSurfer directory structure to load the correct file.

### Usage

```
subject.volume(
    subjects_dir,
    subject_id,
    volume,
    format = "auto",
    drop_empty_dims = TRUE,
    with_header = FALSE,
    mri_subdir = NULL
)
```

### Arguments

subjects_dir	character string, the FreeSurfer 'SUBJECTS_DIR', i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	character string, the subject identifier.
volume	character string, name of the volume file without file extension. Examples: 'brain' or 'aseg'.
format	string. One of 'mgh', 'mgz', 'AUTO'. If left at the default value 'AUTO', the function will look for files with extensions 'mgh' and 'mgz' (in that order) and use the first one that exists.
drop_empty_dims	logical, whether to drop empty dimensions of the returned data. Passed to <a href="#">read.fs.mgh</a> .
with_header	logical. Whether to return the header as well. If TRUE, return a named list with entries "data" and "header". The latter is another named list which contains the header data. These header entries exist: "dtype": int, one of: 0=MRI_UCHAR; 1=MRI_INT; 3=MRI_FLOAT; 4=MRI_SHORT. "voldim": integer vector. The volume (=data) dimensions. E.g., c(256, 256, 256, 1). These header entries may exist: "vox2ras_matrix" (exists if "ras_good_flag" is 1), "mr_params" (exists if "has_mr_params" is 1). Passed to <a href="#">read.fs.mgh</a> .
mri_subdir	character string or NULL, the subdir to use within the 'mri' directory. Defaults to 'NULL', which means to read directly from the 'mri' dir. You could use this to read volumes from the 'mri/orig/' directory by setting it to 'orig'.

### Value

numerical array, the voxel data. If 'with\_header', the full volume datastructure (see above).

**Examples**

```

fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
brain = subject.volume(subjects_dir, 'subject1', 'brain', with_header = TRUE);
# Use the vox2ras matrix from the header to compute RAS coordinates at CRS voxel (0, 0, 0):
brain$header$vox2ras_matrix %*% c(0,0,0,1);

```

---

vdata.split.by.hemi     *Split morph data vector at hemisphere boundary.*

---

**Description**

Given a single vector of per-vertex data for a mesh, split it at the hemi boundary. This is achieved by loading the respective surface and checking the number of vertices for the 2 hemispheres.

**Usage**

```
vdata.split.by.hemi(subjects_dir, subject_id, vdata, surface = "white")
```

**Arguments**

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier
vdata	numerical vector of data for both hemispheres, one value per vertex
surface	the surface to load to determine the vertex counts

**Value**

a hemilist, each entry contains the data part of the respective hemi.

**Note**

Instead of calling this function to split the data, you could use the 'split\_by\_hemi' parameter of [subject.morph.native](#).

---

vis.color.on.subject    *Visualize pre-defined vertex colors on a subject.*

---

## Description

Visualize pre-defined vertex colors on a subject.

## Usage

```
vis.color.on.subject(
  subjects_dir,
  vis_subject_id,
  color_lh,
  color_rh,
  surface = "white",
  views = c("t4"),
  rgloptions = list(),
  rglactions = list()
)
```

## Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
vis_subject_id	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
color_lh	vector of colors to visualize on the left hemisphere surface.
color_rh	vector of colors to visualize on the right hemisphere surface.
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to <a href="#">par3d</a> . Example: rgloptions = list("windowRect"=c(50,50,1000,1000)).
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05,0.95)).

## Value

list of coloredmeshes. The coloredmeshes used for the visualization.

**See Also**

Other visualization functions: `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.labeldata.on.subject()`, `vis.mask.on.subject()`, `vis.region.values.on.subject()`, `vis.subject.annot()`, `vis.subject.label()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`, `vis.symmetric.data.on.subject()`, `vislayout.from.coloredmeshes()`

**Examples**

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
color_lh = '#ff0000';
num_verts_subject1_rh = 153333;
color_rh = rep('#333333', num_verts_subject1_rh);
color_rh[1:30000] = '#00ff00';
color_rh[30001:60000] = '#ff0000';
color_rh[60001:90000] = '#0000ff';
color_rh[90001:120000] = '#ffff00';
color_rh[120001:150000] = '#00ffff';
vis.color.on.subject(subjects_dir, 'subject1', color_lh, color_rh);
```

---

vis.coloredmeshes	<i>Visualize a list of colored meshes in a single scene.</i>
-------------------	--

---

**Description**

Visualize a list of colored meshes in a single scene.

**Usage**

```
vis.coloredmeshes(
  coloredmeshes,
  background = "white",
  skip_all_na = TRUE,
  style = "default",
  rgloptions = list(),
  rglactions = list(),
  draw_colorbar = FALSE
)
```

**Arguments**

coloredmeshes	list of coloredmesh. A coloredmesh is a named list as returned by the <code>coloredmesh.from.*</code> functions. It has the entries 'mesh' of type <code>tmesh3d</code> , a 'col', which is a color specification for such a mesh.
background	string, background color passed to <code>rgl::bg3d()</code>



skip_all_na	logical, whether to skip (i.e., not render) meshes in the list that have the property 'render' set to FALSE. Defaults to TRUE. Practically, this means that a hemisphere for which the data was not given is not rendered, instead of being rendered in a single color.
style,	a named list of style parameters or a string specifying an available style by name (e.g., 'shiny'). Defaults to 'default', the default style.
rgloptions	option list passed to <a href="#">par3d</a> . Example: <code>rgloptions = list("windowRect"=c(50,50,1000,1000));</code>
rglactions	named list. A list in which the names are from a set of pre-defined actions. Defaults to the empty list.
draw_colorbar	logical. Whether to draw a colorbar. WARNING: Will only show up if there is enough space in the plot area and does not resize properly. Defaults to FALSE. See <a href="#">coloredmesh.plot.colorbar.separate</a> for an alternative.

**Value**

the list of visualized coloredmeshes

**Note**

To change or adapt the colorbar, you should use the `makecmap_options` parameter when constructing them in a `vis` function. See the example.

**Examples**

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
cm = vis.subject.morph.native(subjects_dir, 'subject1', 'thickness',
  makecmap_options=list('n'=100));
# You could mess with the meshes here.
vis.coloredmeshes(cm);
```

---

`vis.colortable.legend` *Create a separate legend plot for a colortable or an annotation.*

---

**Description**

This plots a legend for a colortable or an atlas (annotation), showing the region names and their assigned colors. This function creates a new plot.

**Usage**

```
vis.colortable.legend(colortable, ncols = 1L, plot_struct_index = TRUE)
```

**Arguments**

colortable	dataframe, a colortable as returned by <code>read.fs.colortable</code> or the inner 'colortable_df' returned by <code>subject.annot</code> . One can also pass an annotation (*fs.annot* instance).
ncols	positive integer, the number of columns to use when plotting
plot_struct_index	logical, whether to plot the region index from tge 'struct_index' field. If there is no such field, this is silently ignored.

**Note**

This function plots one or more legends (see [legend](#)). You may have to adapt the device size before calling this function if you inted to plot a large colortable.

**Examples**

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
annot = subject.annot(subjects_dir, 'subject1', 'lh', 'aparc');
vis.colortable.legend(annot$colortable_df, ncols=3);
```

---

vis.data.on.fsaverage *Visualize arbitrary data on the fsaverage template subject, if available.*

---

**Description**

Creates a surface mesh, applies a colormap transform the morphometry data values into colors, and renders the resulting colored mesh in an interactive window. If hemi is 'both', the data is rendered for the whole brain. This function tries to automatically retrieve the subjects\_dir of the fsaverage template subject by checking the environment variables SUBJECTS\_DIR and FREESURFER\_HOME for the subject. The subject is required for its surfaces, which are not shipped with this package for licensing reasons.

**Usage**

```
vis.data.on.fsaverage(
  subjects_dir = NULL,
  vis_subject_id = "fsaverage",
  morph_data_lh,
  morph_data_rh,
  surface = "white",
  colormap = NULL,
  views = c("t4"),
  rgloptions = rglot(),
```

```

    rglactions = list(),
    draw_colorbar = FALSE,
    makecmap_options = mkco.seq(),
    bg = NULL
)

```

### Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
vis_subject_id	string. The subject identifier from which to obtain the surface for data visualization. Defaults to 'fsaverage'.
morph_data_lh	numeric vector or character string or NULL, the data to visualize on the left hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of morph_data_lh or morph_data_rh is allowed to be NULL.
morph_data_rh	numeric vector or character string or NULL, the data to visualize on the right hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of morph_data_lh or morph_data_rh is allowed to be NULL.
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
colormap	a colormap function. <b>**DEPRECATED**</b> : use parameter 'makecmap_options' instead, like this: makecmap_options=list('colFn'=terrain.colors).
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to <a href="#">par3d</a> . Example: rgloptions = list("windowRect"=c(50,50,1000,1000)).
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05,0.95)).
draw_colorbar	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See <a href="#">coloredmesh.plot.colorbar.separate</a> for an alternative.
makecmap_options	named list of parameters to pass to <a href="#">makecmap</a> . Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.

bg a background definition. Can be a surface color layer or a character string, see [collayer.bg](#) for valid strings.

### Value

list of coloredmeshes. The coloredmeshes used for the visualization.

### See Also

Other visualization functions: [vis.color.on.subject\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Other morphometry visualization functions: [vis.data.on.subject\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.symmetric.data.on.subject\(\)](#)

---

vis.data.on.subject     *Visualize arbitrary data on the surface of any subject.*

---

### Description

Creates a surface mesh, applies a colormap transform the morphometry data values into colors, and renders the resulting colored mesh in an interactive window. If hemi is 'both', the data is rendered for the whole brain.

### Usage

```
vis.data.on.subject(
  subjects_dir,
  vis_subject_id,
  morph_data_lh,
  morph_data_rh,
  surface = "white",
  colormap = NULL,
  views = c("t4"),
  rgloptions = rglot(),
  rglactions = list(),
  draw_colorbar = FALSE,
  makecmap_options = mkco.seq(),
  bg = NULL
)
```

### Arguments

subjects\_dir string. The FreeSurfer SUBJECTS\_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.

vis_subject_id	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
morph_data_lh	numeric vector or character string or NULL, the data to visualize on the left hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of morph_data_lh or morph_data_rh is allowed to be NULL.
morph_data_rh	numeric vector or character string or NULL, the data to visualize on the right hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of morph_data_lh or morph_data_rh is allowed to be NULL.
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
colormap	a colormap function. <b>**DEPRECATED**</b> : use parameter 'makecmap_options' instead, like this: makecmap_options=list('colFn'=terrain.colors).
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to <a href="#">par3d</a> . Example: rgloptions = list("windowRect"=c(50,50,1000,1000)).
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05,0.95)).
draw_colorbar	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See <a href="#">coloredmesh.plot.colorbar.separate</a> for an alternative.
makecmap_options	named list of parameters to pass to <a href="#">makecmap</a> . Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
bg	a background definition. Can be a surface color layer or a character string, see <a href="#">collayer.bg</a> for valid strings.

**Value**

list of coloredmeshes. The coloredmeshes used for the visualization.

**See Also**

Other visualization functions: [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#),

```
vis.subject.morph.native(), vis.subject.morph.standard(), vis.symmetric.data.on.subject(),
vislayout.from.coloredmeshes()
```

Other morphometry visualization functions: [vis.data.on.fsaverage\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.symmetric.data.on.subject\(\)](#)

## Examples

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
morph_data_lh = subject.morph.native(subjects_dir, 'subject1', 'thickness', 'lh');
morph_data_rh = NULL;
vis.data.on.subject(subjects_dir, 'subject1', morph_data_lh, morph_data_rh);
```

---

vis.fs.surface

*Visualize fs.surface mesh*

---

## Description

Visualize fs.surface mesh

## Usage

```
vis.fs.surface(
  fs_surface,
  col = "white",
  per_vertex_data = NULL,
  hemi = "lh",
  makecmap_options = mkco.seq(),
  ...
)
```

## Arguments

fs_surface	an fs.surface instance, as returned by function like <a href="#">subject.surface</a> or <a href="#">read.fs.surface</a> . If a character string, it is assumed to be the full path of a surface file, and the respective file is loaded with <a href="#">read.fs.surface</a> .
col	vector of colors, the per-vertex-colors. Defaults to white.
per_vertex_data	numerical vector, per-vertex data. If given, takes precedence over 'col'. Used to color the mesh using the colormap options in parameter 'makecmap_options'. If a character string, it is assumed to be the full path of a morphometry data file, and the respective file is loaded with <a href="#">read.fs.morph</a> .
hemi	character string, one of 'lh' or 'rh'. This may be used by visualization functions to decide whether or not to show this mesh in a certain view.

makecmap\_options  
 named list of parameters to pass to [makecmap](#). Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.  
 ...  
 extra parameters to pass to [vis.coloredmeshes](#).

---

 vis.labeldata.on.subject

*Visualize a label on the surface of a subject.*

---

## Description

Visualizes a label. Note that a label is just a set of vertices, and that you can use this function to visualize sets of vertices, e.g., to see where on the mesh a certain vertex lies. It may be helpful the visualize the vertex with its neighbors, because otherwise it may be too small to spot. Use the function [fsbrain::mesh.vertex.neighbors] to get them. It is advisable to set the view to the interactive 'si' mode and use the 'inflated' surface to identify single vertices.

## Usage

```
vis.labeldata.on.subject(  
  subjects_dir,  
  vis_subject_id,  
  lh_labeldata,  
  rh_labeldata,  
  surface = "white",  
  colormap = NULL,  
  views = c("t4"),  
  rgloptions = list(),  
  rglactions = list(),  
  draw_colorbar = FALSE,  
  makecmap_options = list(colFn = label.colFn.inv)  
)
```

## Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
vis_subject_id	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
lh_labeldata	integer vector of vertex indices for the left hemisphere
rh_labeldata	integer vector of vertex indices for the right hemisphere
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
colormap	a colormap function. <b>**DEPRECATED**</b> : use parameter 'makecmap_options' instead, like this: makecmap_options=list('colFn'=terrain.colors).

views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to <a href="#">par3d</a> . Example: <code>rgloptions = list("windowRect"=c(50,50,1000,1000))</code> .
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05,0.95))</code> .
draw_colorbar	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See <a href="#">coloredmesh.plot.colorbar.separate</a> for an alternative.
makecmap_options	named list of parameters to pass to <a href="#">makecmap</a> . Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.

**Value**

list of coloredmeshes. The coloredmeshes used for the visualization.

**Note**

Drawing a colorbar for label data makes limited sense, use a legend instead. The colorbar can give a rough overview of the relative number of label and non-label vertices though, so it is possible to request one.

**See Also**

Other label functions: [apply.label.to.morphdata\(\)](#), [apply.labeldata.to.morphdata\(\)](#), [subject.lobes\(\)](#), [subject.mask\(\)](#), [vis.subject.label\(\)](#)

Other visualization functions: [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

**Examples**

```
fsbrain::download_optional_data();

# Define the data to use:
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
lh_labeldata = c(1000, 1001, 1002); # only the vertices, will be tiny.
subject_id = 'subject1';
surface = 'white'; # Should use 'inflated', but we do not currently
                # ship it for the example subject to reduce download size.
```



```
# For the right hemi, extend them to neighborhood for better visibility:
rh_labeldata = c(500, 5000);
rh_surface = subject.surface(subjects_dir, subject_id, surface, 'rh');
rh_labeldata_neighborhood = mesh.vertex.neighbors(rh_surface, rh_labeldata);
vis.labeldata.on.subject(subjects_dir, subject_id, lh_labeldata,
  rh_labeldata_neighborhood$vertices, surface=surface, views=c('si'));
```

---

vis.mask.on.subject     *Visualize a vertex mask on the surface of a subject.*

---

## Description

A mask is a logical vector that contains one value per vertex. You can create it manually, or use functions like [mask.from.labeldata.for.hemi](#) to create and modify it. Check the example for this function.

## Usage

```
vis.mask.on.subject(
  subjects_dir,
  vis_subject_id,
  mask_lh,
  mask_rh,
  surface = "white",
  colormap = NULL,
  views = c("t4"),
  rgloptions = list(),
  rglactions = list(),
  draw_colorbar = FALSE,
  makecmap_options = list(colFn = label.colFn.inv)
)
```

## Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
vis_subject_id	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
mask_lh	logical vector or NULL, the mask to visualize on the left hemisphere surface. Must have the same length as the lh surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of mask_lh or mask_rh is allowed to be NULL.

mask_rh	logical vector or NULL, the mask to visualize on the right hemisphere surface. Must have the same length as the rh surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of mask_lh or mask_rh is allowed to be NULL.
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
colormap	a colormap function. <b>**DEPRECATED**</b> : use parameter 'makecmap_options' instead, like this: makecmap_options=list('colFn'=terrain.colors).
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to <a href="#">par3d</a> . Example: rgloptions = list("windowRect"=c(50,50,1000,1000)).
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05,0.95)).
draw_colorbar	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See <a href="#">coloredmesh.plot.colorbar.separate</a> for an alternative.
makecmap_options	named list of parameters to pass to <a href="#">makecmap</a> . Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.

## Value

list of coloredmeshes. The coloredmeshes used for the visualization.

## Note

Drawing a colorbar for label data makes limited sense, use a legend instead. The colorbar can give a rough overview of the relative number of label and non-label vertices though, so it is possible to request one.

## See Also

Other mask functions: [coloredmesh.from.mask\(\)](#), [mask.from.labeldata.for.hemi\(\)](#)

Other visualization functions: [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

**Examples**

```

fsbrain::download_optional_data();

# Define the data to use:
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
subject_id = 'subject1';
surface = 'white';
hemi = 'both';
atlas = 'aparc';
region = 'bankssts';

# Create a mask from a region of an annotation:
lh_annot = subject.annot(subjects_dir, subject_id, 'lh', atlas);
rh_annot = subject.annot(subjects_dir, subject_id, 'rh', atlas);
lh_label = label.from.annotdata(lh_annot, region);
rh_label = label.from.annotdata(rh_annot, region);
lh_mask = mask.from.labeldata.for.hemi(lh_label, length(lh_annot$vertices));
rh_mask = mask.from.labeldata.for.hemi(rh_label, length(rh_annot$vertices));

# Edit the mask: add the vertices from another region to it:
region2 = 'medialorbitofrontal';
lh_label2 = label.from.annotdata(lh_annot, region2);
rh_label2 = label.from.annotdata(rh_annot, region2);
lh_mask2 = mask.from.labeldata.for.hemi(lh_label2, length(lh_annot$vertices),
existing_mask = lh_mask);
rh_mask2 = mask.from.labeldata.for.hemi(rh_label2, length(rh_annot$vertices),
existing_mask = rh_mask);
# Visualize the mask:
vis.mask.on.subject(subjects_dir, subject_id, lh_mask2, rh_mask2);

```

---

vis.path.along.verts *Draw a 3D line from vertex to vertex*

---

**Description**

To get a nice path along the surface, pass the vertex indices along a geodesic path. Note: You can first open an interactive brain view ('view='si') with a vis\* function like [vis.subject.morph.native](#), then run this function to draw into the active plot.

**Usage**

```
vis.path.along.verts(surface_vertices, path_vertex_indices)
```

**Arguments**

- surface\_vertices  
float matrix of size (n, 3), the surface vertex coordinates, as returned as part of `subject.surface` or `read.fs.surface`, in the member "vertices".
- path\_vertex\_indices  
vector of vertex indices, the path. You will need to have it computed already. (This function does **not** compute geodesic paths. You can use it to visualize such a path though.)

**See Also**

Other surface mesh functions: `face.edges()`, `label.border()`, `mesh.vertex.included.faces()`, `mesh.vertex.neighbors()`, `subject.surface()`

---

vis.region.values.on.subject

*Visualize arbitrary data, one value per atlas region, on the surface of any subject (including template subjects).*

---

**Description**

This function can be used for rendering a single value (color) for all vertices of an atlas region. The typical usecase is the visualization of results of atlas-based analyses, e.g., p-value, means or other aggregated values over all vertices of a region.

**Usage**

```
vis.region.values.on.subject(
  subjects_dir,
  subject_id,
  atlas,
  lh_region_value_list,
  rh_region_value_list,
  surface = "white",
  colormap = NULL,
  views = c("t4"),
  rgloptions = list(),
  rglactions = list(),
  value_for_unlisted_regions = NA,
  draw_colorbar = FALSE,
  makecmap_options = mkco.heat(),
  bg = NULL
)
```

**Arguments**

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
atlas	string. The brain atlas to use. E.g., 'aparc' or 'aparc.a2009s'.
lh_region_value_list	named list. A list for the left hemisphere in which the names are atlas regions, and the values are the value to write to all vertices of that region. You can pass an unnamed list, but then the its length must exactly match the number of atlas regions. The order of values must also match the order of regions in the annotation, of course. The resulting mapping will be printed so you can check it.
rh_region_value_list	named list. A list for the right hemisphere in which the names are atlas regions, and the values are the value to write to all vertices of that region.
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
colormap	a colormap function. <b>**DEPRECATED**</b> : use parameter 'makecmap_options' instead, like this: makecmap_options=list('colFn'=terrain.colors).
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to <a href="#">par3d</a> . Example: rgloptions = list("windowRect"=c(50,50,1000,1000)).
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05,0.95)).
value_for_unlisted_regions	numerical scalar or 'NaN', the value to assign to regions which do not occur in the region_value_lists. Defaults to 'NaN'.
draw_colorbar	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See <a href="#">coloredmesh.plot.colorbar.separate</a> for an alternative.
makecmap_options	named list of parameters to pass to <a href="#">makecmap</a> . Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
bg	a background definition. Can be a surface color layer or a character string, see <a href="#">collayer.bg</a> for valid strings.

**Value**

list of coloredmeshes. The coloredmeshes used for the visualization.

**See Also**

Other visualization functions: `vis.color.on.subject()`, `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.labeldata.on.subject()`, `vis.mask.on.subject()`, `vis.subject.annot()`, `vis.subject.label()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`, `vis.symmetric.data.on.subject()`, `vislayout.from.coloredmeshes()`

Other region-based visualization functions: `vis.subject.annot()`

**Examples**

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
atlas = 'aparc'; # Desikan atlas
# For the left hemisphere, we just assign a subset of the
# atlas regions. The others will get the default value.
lh_region_value_list = list("bankssts"=0.9, "precuneus"=0.7, "postcentral"=0.8, "lingual"=0.6);
# For the right hemisphere, we retrieve the full list of regions for
# the atlas, and assign random values to all of them.
atlas_region_names = get.atlas.region.names(atlas, template_subjects_dir = subjects_dir,
  template_subject='subject1');
rh_region_value_list = rnorm(length(atlas_region_names), 3.0, 1.0);
names(rh_region_value_list) = atlas_region_names;
vis.region.values.on.subject(subjects_dir, 'subject1', atlas,
  lh_region_value_list, rh_region_value_list);
```

---

`vis.subject.annot`

*Visualize an annotation for a subject.*

---

**Description**

Creates a surface mesh, loads the colors from the annotation, and renders the resulting colored mesh in an interactive window. If hemi is 'both', the data is rendered for the whole brain.

**Usage**

```
vis.subject.annot(
  subjects_dir,
  subject_id,
  atlas,
  hemi = "both",
  surface = "white",
  views = c("t4"),
  rgloptions = list(),
  rglactions = list(),
  outline = FALSE
)
```

**Arguments**

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
atlas	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded. Can also be a hemilist of already loaded annotations.
hemi	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to <a href="#">par3d</a> . Example: <code>rgloptions = list("windowRect"=c(50,50,1000,1000))</code> .
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05,0.95))</code> .
outline	logical, whether to draw an outline only instead of filling the regions. Defaults to 'FALSE'. Instead of passing 'TRUE', one can also pass a list of extra parameters to pass to <a href="#">annot.outline</a> , e.g., <code>outline=list('outline_color'='#000000')</code> . Using this increases computation time dramatically, sorry for the performance.

**Value**

list of coloredmeshes. The coloredmeshes used for the visualization.

**See Also**

Other visualization functions: [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Other region-based visualization functions: [vis.region.values.on.subject\(\)](#)

**Examples**

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
vis.subject.annot(subjects_dir, 'subject1', 'aparc', 'both');
```

---

vis.subject.label      *Visualize a binary label for a subject.*

---

### Description

Visualize a label for a subject. A label is just a logical vector with one entry for each vertex in the mesh. Each vertex may additionally be associated with a scalar value, but this function ignored that.

### Usage

```
vis.subject.label(
    subjects_dir,
    subject_id,
    label,
    hemi,
    surface = "white",
    colormap = NULL,
    views = c("t4"),
    rgloptions = rglot(),
    rglactions = list(),
    draw_colorbar = FALSE,
    makecmap_options = list(colFn = label.colFn.inv, col.na = "#FFFFFF00"),
    map_to_NA = 0L,
    bg = NULL
)
```

### Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
label	string. Name of the label file, without the hemi part (if any), but including the '.label' suffix. E.g., 'cortex.label' for '?h.cortex.label'.
hemi	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
colormap	a colormap function. <b>**DEPRECATED**</b> : use parameter 'makecmap_options' instead, like this: makecmap_options=list('colFn'=terrain.colors).
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to <a href="#">par3d</a> . Example: rgloptions = list("windowRect"=c(50, 50, 1000, 1000)).



rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png"=~~/fsbrain.png", "clip_data"=c(0.05,0.95)).</code>
draw_colorbar	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See <a href="#">coloredmesh.plot.colorbar.separate</a> for an alternative.
makecmap_options	named list of parameters to pass to <a href="#">makecmap</a> . Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
map_to_NA	the value or value range that should <b>not</b> be considered part of the label, and should thus be plotted as background color. Only used if 'bg' is not 'NULL'. If a single value, only exactly this value is used (typically 0). If two values, they are interpreted as a range, and a values between them are mapped to NA. If you prefer to map the data to NA yourself before using this function, pass 'NULL'.
bg	a background definition. Can be a surface color layer or a character string, see <a href="#">collayer.bg</a> for valid strings.

**Value**

list of coloredmeshes. The coloredmeshes used for the visualization.

**Note**

Drawing a colorbar for label data makes limited sense, use a legend instead. The colorbar can give a rough overview of the relative number of label and non-label vertices though, so it is possible to request one.

**See Also**

Other visualization functions: [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Other label functions: [apply.label.to.morphdata\(\)](#), [apply.labeldata.to.morphdata\(\)](#), [subject.lobes\(\)](#), [subject.mask\(\)](#), [vis.labeldata.on.subject\(\)](#)

**Examples**

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
subject_id = 'subject1';
surface = 'white';
hemi = 'both';
```

```
label = 'cortex.label';
vis.subject.label(subjects_dir, subject_id, label, hemi, views="si");
```

---

```
vis.subject.morph.native
```

*Visualize native space morphometry data for a subject.*

---

## Description

Creates a surface mesh, applies a colormap transform the morphometry data values into colors, and renders the resulting colored mesh in an interactive window. If hemi is 'both', the data is rendered for the whole brain.

## Usage

```
vis.subject.morph.native(
    subjects_dir,
    subject_id,
    measure,
    hemi = "both",
    surface = "white",
    colormap = NULL,
    views = c("t4"),
    rgloptions = rglot(),
    rglactions = list(),
    draw_colorbar = FALSE,
    cortex_only = FALSE,
    style = "default",
    makecmap_options = mkco.seq(),
    bg = NULL
)
```

## Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
measure	string. The morphometry data to use. E.g., 'area' or 'thickness'. Pass NULL to render just the surface in white, without any data.
hemi	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".

colormap	a colormap function. <b>**DEPRECATED**</b> : use parameter 'makecmap_options' instead, like this: <code>makecmap_options=list('colFn'=terrain.colors)</code> .
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to <a href="#">par3d</a> . Example: <code>rgloptions = list("windowRect"=c(50,50,1000,1000))</code> .
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05,0.95))</code> .
draw_colorbar	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See <a href="#">coloredmesh.plot.colorbar.separate</a> for an alternative.
cortex_only	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>not</i> part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subject. Defaults to FALSE.
style	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.
makecmap_options	named list of parameters to pass to <a href="#">makecmap</a> . Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
bg	a background definition. Can be a surface color layer or a character string, see <a href="#">collayer.bg</a> for valid strings.

**Value**

list of coloredmeshes. The coloredmeshes used for the visualization.

**See Also**

Other visualization functions: [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Other morphometry visualization functions: [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.symmetric.data.on.subject\(\)](#)

**Examples**

```
fsbrain::download_optional_data();
```

```
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
vis.subject.morph.native(subjects_dir, 'subject1', 'thickness', 'lh', views=c("t9"));
```

---

```
vis.subject.morph.standard
```

*Visualize native space morphometry data for a subject or a group.*

---

### Description

Renders standard space morphometry data for a single subject, or the group mean for a group of subjects. The default template subject is fsaverage.

### Usage

```
vis.subject.morph.standard(
  subjects_dir,
  subject_id,
  measure,
  hemi = "both",
  fwhm = "10",
  surface = "white",
  template_subject = "fsaverage",
  template_subjects_dir = NULL,
  colormap = NULL,
  views = c("t4"),
  rgloptions = rglot(),
  rglactions = list(),
  draw_colorbar = FALSE,
  cortex_only = FALSE,
  makecmap_options = mkco.seq(),
  bg = NULL
)
```

### Arguments

<code>subjects_dir</code>	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
<code>subject_id</code>	character string or vector of character strings, the subject or subjects. For a single subjects, its data will be plotted. If a group of subjects is given instead, at each vertex the mean value over all the subjects will be plotted.
<code>measure</code>	string. The morphometry data to use. E.g., 'area' or 'thickness'. Pass NULL to render just the surface in white, without any data.
<code>hemi</code>	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.

fwhm	string, smoothing setting (full width at half maximum of the kernel). The smoothing part of the filename, typically something like '0', '5', '10', ..., or '25'.
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
template_subject	The template subject used. This will be used as part of the filename, and its surfaces are loaded for data visualization. Defaults to 'fsaverage'.
template_subjects_dir	The template subjects dir. If NULL, the value of the parameter 'subjects_dir' is used. If you have FreeSurfer installed and configured, and are using the standard fsaverage subject, try passing the result of calling 'file.path(Sys.getenv('FREESURFER_HOME'), 'subjects')'.
colormap	a colormap function. <b>**DEPRECATED**</b> : use parameter 'makecmap_options' instead, like this: <code>makecmap_options=list('colFn'=terrain.colors)</code> .
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to <a href="#">par3d</a> . Example: <code>rgloptions = list("windowRect"=c(50,50,1000,1000))</code> .
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05,0.95))</code> .
draw_colorbar	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See <a href="#">coloredmesh.plot.colorbar.separate</a> for an alternative.
cortex_only	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>not</i> part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subject. Defaults to FALSE.
makecmap_options	named list of parameters to pass to <a href="#">makecmap</a> . Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
bg	a background definition. Can be a surface color layer or a character string, see <a href="#">collayer.bg</a> for valid strings.

**Value**

list of coloredmeshes. The coloredmeshes used for the visualization.

**See Also**

Other visualization functions: [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Other morphometry visualization functions: [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.subject.morph.native\(\)](#), [vis.symmetric.data.on.subject\(\)](#)

**Examples**

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
fsaverage_dir = file.path(Sys.getenv('FREESURFER_HOME'), 'subjects');
if(dir.exists(fsaverage_dir)) {
  vis.subject.morph.standard(subjects_dir, 'subject1', 'thickness', 'lh',
    '10', template_subjects_dir=fsaverage_dir);
}
# The last command will load the file
# *<subjects_dir>/subject1/surf/lh.thickness.fwhm10.fsaverage.mgh* and
# visualize the data on *$FREESURFER_HOME/subjects/fsaverage/surf/lh.white*.
```

---

```
vis.symmetric.data.on.subject
```

*Visualize clusters or activation data on the surface of any subject.*

---

**Description**

This function is intended to plot symmetric data around zero (like positive and negative activation data, signed p-values, etc.) on a subject's surface. It is a thin wrapper around [vis.data.on.subject](#).

**Usage**

```
vis.symmetric.data.on.subject(
  subjects_dir,
  vis_subject_id,
  morph_data_lh,
  morph_data_rh,
  surface = "white",
  colormap = NULL,
  views = c("t4"),
  rgloptions = list(),
  rglactions = list(),
  draw_colorbar = FALSE,
  makecmap_options = list(colFn = cm.div(), symm = TRUE, col.na = "#FFFFFF00", n = 100),
```

```

    map_to_NA = c(0),
    bg = NULL
)

```

### Arguments

- subjects\_dir** string. The FreeSurfer SUBJECTS\_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
- vis\_subject\_id** string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
- morph\_data\_lh** numeric vector or character string or NULL, the data to visualize on the left hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis\_subject\_id has vertices. If NULL, this surface will not be rendered. Only one of morph\_data\_lh or morph\_data\_rh is allowed to be NULL.
- morph\_data\_rh** numeric vector or character string or NULL, the data to visualize on the right hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis\_subject\_id has vertices. If NULL, this surface will not be rendered. Only one of morph\_data\_lh or morph\_data\_rh is allowed to be NULL.
- surface** string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
- colormap** a colormap function. **\*\*DEPRECATED\*\***: use parameter 'makecmap\_options' instead, like this: makecmap\_options=list('colFn'=terrain.colors).
- views** list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
- rgloptions** option list passed to [par3d](#). Example: rgloptions = list("windowRect"=c(50,50,1000,1000)).
- rglactions** named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: rglactions = list("snapshot\_png"="~/fsbrain.png", "clip\_data"=c(0.05,0.95)).
- draw\_colorbar** logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See [coloredmesh.plot.colorbar.separate](#) for an alternative.
- makecmap\_options** named list of parameters to pass to [makecmap](#). Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
- map\_to\_NA** the value or value range that should **\*\*not\*\*** be considered a cluster, and should thus be plotted as background color. These values will be set to NA, leading

to transparent rendering, so the background will be visible instead. If a single value, only exactly this value is used (typically 0). If two values, they are interpreted as a range, and a values between them are mapped to NA. If you prefer to map the data to NA yourself before using this function or do not want to use a , pass 'NULL'.

**bg** a background definition. Can be a surface color layer or a character string, see [collayer.bg](#) for valid strings.

### Value

list of coloredmeshes. The coloredmeshes used for the visualization.

### See Also

Other visualization functions: [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Other morphometry visualization functions: [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#)

### Examples

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
morph_data_lh = subject.morph.native(subjects_dir, 'subject1', 'thickness', 'lh');
morph_data_rh = NULL;
vis.symmetric.data.on.subject(subjects_dir, 'subject1', morph_data_lh, morph_data_rh);
```

---

vislayout.from.coloredmeshes

*Visualize coloredmeshes from several angles and combine the images into a new figure.*

---

### Description

Create a tight layout view of coloredmeshes from several angles. Creates separate 'sd\_<angle>' images, then crops and finally merges them into a single output image with image magick. The 'coloredmeshes' to pass to this function are usually obtained by running any 'vis\*' function (like [vis.subject.morph.native](#), [vis.subject.morph.standard](#), [vis.subject.label](#), [vis.subject.annot](#), and others). That means you can use this function to visualize all kinds of data, e.g., morphometry data in native and standard space, labels, and brain atlases.



**Usage**

```
vislayout.from.coloredmeshes(
    coloredmeshes,
    view_angles = get.view.angle.names(angle_set = "t4"),
    rgloptions = list(),
    rglactions = list(),
    style = "default",
    output_img = "fsbrain_arranged.png",
    silent = FALSE,
    grid_like = TRUE
)
```

**Arguments**

coloredmeshes,	list of coloredmesh. A coloredmesh is a named list as returned by the ‘coloredmesh.from*’ functions (like <a href="#">coloredmesh.from.morph.native</a> ). It has the entries ‘mesh’ of type <code>tmesh3d</code> , a ‘col’, which is a color specification for such a mesh. The ‘vis*’ functions (like <a href="#">vis.subject.morph.native</a> ) all return a list of coloredmeshes.
view_angles	list of strings. See <a href="#">get.view.angle.names</a> for all valid strings.
rgloptions	option list passed to <a href="#">par3d</a> . Example: <code>rgloptions = list("windowRect"=c(50,50,1000,1000))</code> .
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action.
style	character string, a rendering style, e.g., ‘default’, ‘shiny’ or ‘semitransparent’. Alternatively, a named list of style parameters (see <a href="#">material3d</a> ), e.g., <code>list("shininess"=50,specular=...)</code> . Use the magic word ‘from_mesh’ to use the ‘style’ field of each coloredmesh instead of a single, global style. In that case, you will have to make sure your meshes have such a field, if not, the style ‘default’ is used as a fallback for those which don’t.
output_img	string, path to the output file. Defaults to "fsbrain_arranged.png"
silent	logical, whether to suppress all messages
grid_like	logical, whether to arrange the images in a grid-like fashion. If FALSE, they will all be merged horizontally. Passed to <a href="#">arrange.brainview.images</a> .

**Value**

list of coloredmeshes. The coloredmeshes used for the visualization.

**See Also**

Other visualization functions: [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.symmetric.data.on.subject\(\)](#)

**Examples**

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
# Use any vis function to get coloredmeshes. You can visualize morphometry,
# labels, an atlas, whatever. You can suppress the view unless you need it.
coloredmeshes = vis.subject.morph.native(subjects_dir, "subject1", "thickness",
  cortex_only=TRUE, rglactions=list("clip_data"=c(0.05, 0.95)),
  views=NULL);
# The meshes contain the surface, data, and color information and can be
# visualized. You could adapt the rendering style while doing so:
vislayout.from.coloredmeshes(coloredmeshes, style='shiny');
# You could change the rendering style on a per-mesh basis.
coloredmeshes[[1]]$style = list("shininess"=50, alpha=0.5);
vislayout.from.coloredmeshes(coloredmeshes, style='from_mesh');

## End(Not run)
```

---

vol.boundary.box	<i>Compute 3D bounding box of a volume.</i>
------------------	---

---

**Description**

Compute the axis-aligned foreground bounding box of a 3D volume, i.e. the inner foreground area that must be retained if you want to remove all background from the corners of the volume. The foreground is determined by thresholding, such that all values greater than 0 are considered foreground. See [vol.boundary.mask](#) for details.

**Usage**

```
vol.boundary.box(volume, threshold = 0L, apply = FALSE)
```

**Arguments**

volume	a 3D image volume
threshold	numerical, the threshold intensity used to separate background and foreground. All voxels with intensity values greater than this value will be considered ‘foreground’ voxels.
apply	logical, whether to directly apply the bounding box and return the resulting volume instead.

**Value**

named list with 2 entries: ‘from’ is an integer vector of length 3, defining the minimal (x,y,z) foreground indices. ‘to’ is an integer vector of length 3, defining the maximal (x,y,z) foreground indices.

**See Also**

Other volume utility: [vol.imagestack\(\)](#), [vol.merge\(\)](#), [vol.overlay.colors.from.activation\(\)](#), [vol.planes\(\)](#), [vol.slice\(\)](#)

---

```
vol.boundary.box.apply
```

*Apply a boundary box to a volume, returning the inner volume part*

---

**Description**

Apply a boundary box to a volume, returning the inner volume part

**Usage**

```
vol.boundary.box.apply(volume, bbox)
```

**Arguments**

volume	a 3D image volume
bbox	the boundary box

**Value**

a 3D image volume, the inner volume part, resulting from the application of the boundary box

---

```
vol.hull
```

*Retain only the outer hull voxels of the foreground.*

---

**Description**

Filters the \*foreground\* voxel in the volume by keeping only an outer border of voxels, and setting the inner core voxels to 'NA'. This is a utility function for voxel-based visualization. The goal is to remove the inner voxels, which will not be visible anyways, and thus to dramatically reduce the number of triangles that will need to be computed for the mesh.

**Usage**

```
vol.hull(volume, thickness = 1L, axes = c(2L))
```

**Arguments**

volume	numeric 3d array, must contain foreground voxel and background voxels. The latter must have value 'NA'. This function assumes that a solid foreground object surrounded by background exists in the volume.
thickness	integer, the width of the border in voxels, i.e., how many of the voxels in each upright column to keep at the top and at the bottom.
axes	integer vector, the axes to use. Valid values in the vector are 1L, 2L and 3L. You will have to use all 3 axes if you do not want any holes in the object. (Obviously, having noise around the object can still lead to holes.)

**Value**

numeric 3d array, a filtered version of the input. It contains at least as many 'NA' voxels as the input. If the function had any effect, it contains a lot more 'NA' values. The other values and the volume dimensions are left unchanged.

---

vol.imagestack	<i>Turn volume into an ImageMagick image stack.</i>
----------------	---

---

**Description**

Create an image from each slice along the axis, then stack those into an ImageMagick image stack.

**Usage**

```
vol.imagestack(volume, axis = 1L, intensity_scale = 255)
```

**Arguments**

volume	a 3D image volume. Can be numeric, or something that can be read directly by <code>magick::image_read</code> in 2D matrices (slices along the axis), e.g., a 3D array of color strings. If a 2D matrix is passed, the resulting stack will contain a single image.
axis	positive integer in range 1L..3L or an axis name, the axis to use.
intensity_scale	integer, value by which to scale the intensities in the volume to the range '[0, 1]'. Only used for numeric volumes. Set to NULL for data that can be read directly by <code>magick::image_read</code> , and to 1 for intensity data that requires no scaling. Defaults to 255, which is suitable for 8 bit image data.

**Value**

a vectorized ImageMagick image, containing one subimage per slice. This can be interpreted as an animation or whatever.

**See Also**

Other volume utility: [vol.boundary.box\(\)](#), [vol.merge\(\)](#), [vol.overlay.colors.from.activation\(\)](#), [vol.planes\(\)](#), [vol.slice\(\)](#)

---

vol.intensity.to.color

*Convert integer intensity image to RGB color string form.*

---

**Description**

Convert a gray-scale image defined by intensity values in range '[0, 1]' to an image with identical dimensions that contains an R color string (like '#222222') at each position. The color strings are computed from the intensities, by taking the intensity value as the value for all three RGB channels. I.e., the output is still gray-scale, but defined in RGB space. To make it clear, this function does **not** apply a colormap. It only changes the representation of the data, not the resulting colors.

**Usage**

```
vol.intensity.to.color(volume, scale = NULL)
```

**Arguments**

volume	numeric array, typically a 3D image with intensities in range '[0, 1]'. This function now also supports numeric matrices (2D images, slices) and numeric vectors (1D).
scale	numeric or character string, a scaling to apply to the values. Defaults to NULL, which means <i>no scaling</i> and requires the values in 'volume' to be in range '[0, 1]'. You can pass a number like 255 or the string 'normalize' to scale based on the data. You can pass the string 'normalize_if_needed' to scale only if the data is <i>outside</i> the range '[0, 1]', so that data in range '[0.3, 0.5]' would <b>not</b> be rescaled to '[0, 1]'.

**Value**

array (or matrix, or vector) of RGB color strings. All of them will represent gray values.

**Examples**

```
vol.intensity.to.color(c(0.0, 0.5, 1.0));  
# output: "#000000" "#808080" "#FFFFFF"  
vol.intensity.to.color(c(20, 186, 240), scale="normalize");  
vol.intensity.to.color(c(20, 186, 240), scale=255);  
vol.intensity.to.color(c(0.0, 0.5, 0.8), scale="normalize");  
vol.intensity.to.color(c(0.0, 0.5, 0.8), scale="normalize_if_needed");
```

vol.mask.from.segmentation

*Extract subset from a volume by value.*

---

### Description

Extract subset from a volume by value, set all other voxel values to 'NA'. Typically used to extract a brain structure (like corpus callosum) from a volume segmentation (like the 'mri/aseg.mgz' file of a subject). You should consider passing the volume and the include values as integers.

### Usage

```
vol.mask.from.segmentation(volume, include_values)
```

### Arguments

volume	numeric 3D array
include_values	numerical vector, the intensity values which qualify a voxel to be part of the result (without being set to NA)

### Value

numerical array with same dimensions as the input volume. All values which are not part of 'include\_values' replaced with 'NA'.

---

vol.merge

*Merge background volume and overlay to new colors.*

---

### Description

Merge background volume and overlay to new colors.

### Usage

```
vol.merge(  
  volume,  
  overlay_colors,  
  bbox_threshold = 0L,  
  forced_overlay_color = NULL  
)
```

**Arguments**

volume	3D array, can be numeric (gray-scale intensity values) or color strings. If numeric, the intensity values must be in range '[0, 1]'.
overlay_colors	3D array of color strings, values which are not part of the overlay (and should display background in the result) must have 'NA' instead of a color string. Must have same dimensions as the 'volume'.
bbox_threshold	numerical, the threshold intensity used to separate background and foreground. All voxels with intensity values greater than this value in the background 'volume' will be considered 'foreground' voxels. Background-only slices at the borders of the volume will be discarded. Pass 'NULL' to use the full image without applying any bounding box.
forced_overlay_color	NULL or an rgb color string, like '#FF0000' for red. If NULL, the activation colors will be used as foreground colors. Otherwise, the given color will be for all foreground vertices.

**Value**

3D array of color strings, the merged colors

**See Also**

Other volume utility: [vol.boundary.box\(\)](#), [vol.imagestack\(\)](#), [vol.overlay.colors.from.activation\(\)](#), [vol.planes\(\)](#), [vol.slice\(\)](#)

---

vol.overlay.colors.from.activation

*Generate colors for a 3D volume, based on the activation data and a colormap.*

---

**Description**

Applies the colormap function to the data, then sets the alpha value (transparency) to full in all areas without any activation. Feel free to clip data or whatever before passing it, so that all your no-activation data has the same value.

**Usage**

```
vol.overlay.colors.from.activation(
  volume,
  colormap_fn = squash::blueorange,
  no_act_source_value = 0
)
```

**Arguments**

**volume** a 3D array, the activation data (or p-values, effect sizes, or whatever)  
**colormap\_fn** function, a colormap function  
**no\_act\_source\_value** numerical scalar, the value from the data in 'volume' that means no activation. The output colors for this value will be set to 'NA'.

**Value**

a 3D matrix of color strings, with the same dimensions as the input volume

**See Also**

Other volume utility: [vol.boundary.box\(\)](#), [vol.imagestack\(\)](#), [vol.merge\(\)](#), [vol.planes\(\)](#), [vol.slice\(\)](#)

---

vol.overlay.colors.from.colortable

*Compute voxel colors based on colortable.*

---

**Description**

Use the intensity values of the voxels in volume and lookup the respective colors in a colortable.

**Usage**

```
vol.overlay.colors.from.colortable(
  volume,
  colortable,
  ignored_struct_indices = c(),
  ignored_struct_names = c("unknown", "Unknown")
)
```

**Arguments**

**volume** numeric 3D array, the values should be integers present in the 'struct\_index' column of the colortable. All other values will be assigned 'NA' as a color.  
**colortable** a colortable, as returned by [read.fs.colortable](#).  
**ignored\_struct\_indices** integer vector, 'struct\_index' entries in the colortable that should be ignored  
**ignored\_struct\_names** vector of character strings, 'struct\_name' entries in the colortable that should be ignored. Can be combined with 'ignored\_struct\_indices'.

**Value**

character string 3D array, the colors. Voxels in the volume which were not matched by the colortable are set to 'NA' in it.



---

vol.planes	<i>Translate names and indices of planes.</i>
------------	---

---

**Description**

Translate names and indices of 3D image planes. The names only make sense if the data in the volume is in the default FreeSurfer conformed orientation.

**Usage**

```
vol.planes(plane = NULL)
```

**Arguments**

plane            NULL, a plane index, or a plane name.

**Value**

if 'plane' is NULL, all available planes and their indices as a named list. If 'plane' is an integer (a plane index), its name. If 'plane' is an characters string (a plane name), its index.

**See Also**

Other volume utility: [vol.boundary.box\(\)](#), [vol.imagestack\(\)](#), [vol.merge\(\)](#), [vol.overlay.colors.from.activation](#)  
[vol.slice\(\)](#)

---

vol.slice	<i>Extract a slice of a 3D image stack.</i>
-----------	---

---

**Description**

Extracts one or more 2D slices from a 3D image (or a frame of a 4D image). To display the result, you can use [volvis.lightbox](#).

**Usage**

```
vol.slice(  
  volume,  
  slice_index = NULL,  
  frame = 1L,  
  axis = 1L,  
  rotation = 0L,  
  flip = NULL  
)
```

**Arguments**

volume	a 3D or 4D image volume. Note that empty dimensions will be dropped before any processing, and the remaining volume must have 3 or 4 dimensions.
slice_index	positive integer or vector of positive integers, the index into the slices (for the axis). A <i>*slice*</i> in the sense of this function is any 2D image plane extracted from the 3D volume (no matter the axis). If NULL, the slice in the middle of the volume is used. One can pass the magic character string 'all' to use all slice indices along the axis.
frame	positive integer, optional. The frame (time point) to use, only relevant for 4D volumes. The last (i.e. 4th) dimension is assumed to be the time dimension in that case.
axis	positive integer, the axis to use when indexing the slices. Defaults to 1.
rotation	integer, rotation in degrees. Defaults to 0 (no rotation). Must be a multiple of 90L if given.
flip	NULL or one of the character strings 'vertically' or 'horizontally'. Note that flipping <i>*horizontally*</i> means that the image will be mirrored along the central <i>*vertical*</i> axis. If 'NULL' is passed, nothing is flipped. Flipping occurs after rotation.

**Value**

slice data. If 'slice\_index' is a scalar, a numerical 2D matrix (a 2D image from the stack). Otherwise, a numerical 3D array that contains the selected 2D images.

**See Also**

Other volume utility: [vol.boundary.box\(\)](#), [vol.imagestack\(\)](#), [vol.merge\(\)](#), [vol.overlay.colors.from.activation](#)  
[vol.planes\(\)](#)

---

vol.vox.from.crs      *Compute R voxel index for FreeSurfer CRS voxel index.*

---

**Description**

Performs a vox2vos transform from FreeSurfer to R indices.

**Usage**

```
vol.vox.from.crs(fs_crs, add_affine = FALSE)
```

**Arguments**

fs_crs	integer vector of length 3, FreeSurfer indices for column, row, and slice (CRS).
add_affine	logical, whether to add 1 to the output vector as the 4th value

**Value**

the R indices into the volume data for the given FreeSurfer CRS indices

**Examples**

```
# Get voxel intensity data on the command line, based
# on the FreeSurfer (zero-based) CRS voxel indices:
# `mri_info --voxel 127 100 100 ~/data/tim_only/tim/mri/brain.mgz`
# (the result is: 106.0)
#
# That should be identical to:
# our_crs = vol.vox.from.crs(c(127,100,100), add_affine = FALSE);
# brain$data[our_crs[1], our_crs[2], our_crs[3]]; # gives 106
```

---

volvis.contour

*Visualize contour of a volume.*


---

**Description**

Compute a smoothed surface from the voxel intensities in the given volume and render it. Requires the ‘misc3d’ package to be installed, which is an optional dependency.

**Usage**

```
volvis.contour(volume, level = 80, show = TRUE, frame = 1L, color = "white")
```

**Arguments**

volume	a 3D brain volume
level	numeric, intensity threshold for the data. Voxels with intensity value smaller than ‘level’ will be ignored when creating the contour surface.
show	logical, whether to display the triangles. Defaults to ‘TRUE’.
frame	integer, the frame to show in case of a 4D input volume. Can also be the character string ‘all’ to draw the contents of all frames at once. Useful to plot white matter tracts from DTI data, where each tract is stored in a different frame.
color	the color to use when plotting. Can be a vector of colors when plotting all frames of a 4D image (one color per frame).

**Value**

the rendered triangles (a ‘Triangles3D’ instance) with coordinates in surface RAS space if any, ‘NULL’ otherwise. This will be a list if you pass a 4D volume and select ‘all’ frames.

## Examples

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
brain = subject.volume(subjects_dir, 'subject1', 'brain');
# Plot all voxels of the brain:
volvis.contour(brain);
```

---

```
volvis.lightbox
```

```
Draw a lightbox view from volume slices.
```

---

## Description

A lightbox is a single image that holds a set of subimages, arranged in a grid. The images can have a small border or spacing between them. Consecutive subimages will be appear the same row of the grid.

If `overlay_colors` are given, the volume will be used as the background, and it will only be visible where `overlay_colors` has transparency.

## Usage

```
volvis.lightbox(
  volume,
  slices = -5,
  axis = 1L,
  per_row = 5L,
  per_col = NULL,
  border_geometry = "5x5",
  background_color = "#000000",
  arrange_single_image = FALSE
)
```

## Arguments

<code>volume</code>	3D array, can be numeric (gray-scale intensity values) or color strings. If numeric, the intensity values must be in range <code>[0, 1]</code> .
<code>slices</code>	slice index definition. If a vector of integers, interpreted as slice indices. If a single negative interger <code>'-n'</code> , interpreted as every <code>'nth'</code> slice, starting at slice 1. The character string <code>'all'</code> or the value <code>'NULL'</code> will be interpreted as <code>*all slices*</code> .
<code>axis</code>	positive integer in range <code>1L..3L</code> , the axis to use.
<code>per_row</code>	positive integer, the number of subimages per row in the output image. If <code>'NULL'</code> , automatically computed from the number of slices and the <code>'per_col'</code> parameter.

per_col	positive integer, the number of subimages per column in the output image. If 'NULL', automatically computed from the number of slices and the 'per_row' parameter.
border_geometry	string, a geometry string passed to <code>magick::image_border</code> to define the borders to add to each image tile. The default value adds 5 pixels, both horizontally and vertically.
background_color	string, a valid ImageMagick color string such as "white" or "#000080". The color to use when extending images (e.g., when creating the border). Defaults to black.
arrange_single_image	logical, whether to apply the given arrangement (from parameters 'per_row' and 'per_column') even if a single slice (a 2D image) is passed as 'volume'. Defaults to FALSE, which prevents that background tiles are added to fill the row up to 'per_row' images. This also prevents the border from getting added to a single image, so all you see is the raw image. Set to 'TRUE' if you want to arrange even a single image in a row with a border.

---

volvis.voxels

*Voxel-based visualization of volume mask at surface RAS positions.*


---

## Description

Plots a 3D box at every *foreground* voxel in the given volume. All voxels which do not have their intensity value set to 'NA' are considered *foreground* voxels. The locations at which to plot the voxels is computed from the voxel CRS indices using the FreeSurfer `vox2ras_tkr` matrix. This means that the position of the rendered data fits to the surface coordinates (in files like 'surf/lh.white'), and that you can call this function while an active surface rendering window is open (e.g., from calling `vis.subject.morph.native`), to superimpose the surface and volume data. **\*\*On coloring the voxels\*\*** (using *rgl* materials\*): Note that you can call this function several times for the active plot, and color the voxels differently by passing different material properties in each call. Alternatively, check the 'voxelcol' parameter.

## Usage

```
volvis.voxels(volume, render_every = 1, voxelcol = NULL, ...)
```

## Arguments

volume	numeric 3d array, voxels which should not be plotted must have value 'NA'. Take care not to plot too many.
render_every	integer, how many to skip before rendering the next one (to improve performance and/or see deeper structures). Use higher values to see a less dense representation of your data that usually still allows you to see the general shape, but at lower computational burden. Set to 1 to render every (foreground) voxel.

voxelcol character string or a *\*voxel coloring\**. A *\*voxel coloring\** can be specified in three ways: 1) the string 'from\_intensity' will compute colors based on the intensity values of the foreground voxels in the volume, applying normalization of the intensity values if needed. 2) an array of RGB color strings: will be used to retrieve the colors for all foreground vertices, at their CRS indices. 3) A vector with length identical to the number of foreground voxels in the volume: will be applied directly. Obviously, you should not pass a color material parameter (see '...') when using this.

... material properties, passed to [triangles3d](#). Example: color = "#0000ff", lit=FALSE.

## Examples

```
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
brain = subject.volume(subjects_dir, 'subject1', 'brain');
# Plot all voxels of the brain:
brain[which(brain==0L, arr.ind = TRUE)] = NA; # mark background
brain = vol.hull(brain); # remove inner triangles
volvis.voxels(brain);
```

---

vox2ras\_tkr

*The FreeSurfer default vox2ras\_tkr matrix.*

---

## Description

Applying this matrix to a FreeSurfer CRS index of a conformed volume will give you the RAS coordinates of the voxel in surface coordinates, i.e., in the coordinates used in surface file like 'lh.white'. The central voxel is 127,127,127 when using zero-based indices (or 128,128,128 when using one-based indices), meaning its surface RAS coordinates are 0.0, 0.0, 0.0. The returned matrix is the inverse of the 'ras2vox\_tkr' matrix.

## Usage

```
vox2ras_tkr()
```

## Value

numeric 4x4 matrix, the FreeSurfer vox2ras\_tkr matrix.

## See Also

Other surface and volume coordinates: [ras2vox\\_tkr\(\)](#)

**Examples**

```
# Compute surface RAS coordinate of voxel with CRS (0L, 0L, 0L):
vox2ras_tkr() %*% c(0, 0, 0, 1);
# Show that voxel with CRS (128,128,128) is at the
# origin (0.0, 0.0, 0.0) of the surface RAS coordinate system:
(vox2ras_tkr() %*% c(128, 128, 128, 1))[1:3];
```

---

```
write.region.aggregated
```

*Write data aggregated over regions to morphometry file for group.*

---

**Description**

Given an atlas, a subjects list and a measure, aggregate the measure over each region (e.g., mean) and write an output morphometry file in which the value for all region vertices is set to the aggregated value.

**Usage**

```
write.region.aggregated(
  subjects_dir,
  subjects_list,
  measure,
  hemi,
  atlas,
  agg_fun = mean,
  outfile_morph_name = "",
  format = "mgz"
)
```

**Arguments**

`subjects_dir`, string. The FreeSurfer SUBJECTS\_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.

`subjects_list`, string vector. A vector of subject identifiers that match the directory names within `subjects_dir`.

`measure`, string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.

`hemi`, string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.

`atlas`, string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKAtlas". Used to construct the name of the annotation file to be loaded.

agg_fun,	function. An R function that aggregates data, typically max, mean, min or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to mean.
outfile_morph_name,	string. The measure part of the output file name. E.g., 'agg_thickness' will write the file '<subject>/surf/<hemi>.agg_thickness.mgh'. Defaults to 'agg_<measure>'.
format,	string. A morphometry file format. One of 'mgh', 'mgz' or 'curv.' The output file name extension will be set accordingly. Defaults to 'mgz'.

**See Also**

Other output functions: [write.region.values.fsaverage\(\)](#), [write.region.values\(\)](#)

---

write.region.values    *Write one value per atlas region for a subject.*

---

**Description**

Given an atlas and a list that contains one value for each atlas region, write a morphometry file in which all region vertices are assigned the value. Can be used to plot stuff like p-values or effect sizes onto brain regions.

**Usage**

```
write.region.values(
  subjects_dir,
  subject_id,
  hemi,
  atlas,
  region_value_list,
  outfile_morph_name,
  format = "mgz",
  do_write_file = TRUE,
  output_path = NULL,
  value_for_unlisted_regions = NaN
)
```

**Arguments**

subjects_dir,	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id,	string. The subject identifier
hemi,	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
atlas,	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.



region\_value\_list,  
     named list. A list in which the names are atlas regions, and the values are the value to write to all vertices of that region.

outfile\_morph\_name,  
     string. The measure part of the output file name. E.g., 'agg\_thickness' will write the file '<subject>/surf/<hemi>.agg\_thickness.mgh'.

format,  
     string. A morphometry file format. One of 'mgh', 'mgz' or 'curv.' The output file name extension will be set accordingly. Defaults to 'mgz'.

do\_write\_file,  
     logical. Whether to write the data to a file on the disk. If FALSE, the data are only returned (and the outfile\_morph\_name parameter gets ignored). Default to TRUE.

output\_path  
     string, path to the output directory. If omitted, defaults to the 'surf' directory of the subject (i.e., '<subjects\_dir>/<subject\_id>/surf/').

value\_for\_unlisted\_regions,  
     numeric scalar. The value to assign to vertices which are part of atlas regions that are not listed in region\_value\_list. Defaults to NaN.

**Value**

a named list with the following entries: "data": a vector containing the data. "file\_written": string, path to the file that was written, only exists if do\_write = TRUE.

**See Also**

Other output functions: [write.region.aggregated\(\)](#), [write.region.values.fsaverage\(\)](#)

**Examples**

```

fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
region_value_list = list("bankssts"=0.9, "precuneus"=0.7);
write.region.values(subjects_dir, 'subject1', 'lh', 'aparc',
  region_value_list, 'pvalues.mgz', do_write_file = FALSE);

```

---

write.region.values.fsaverage

*Write one value per atlas region for a template subject.*

---

**Description**

Given an atlas and a list that contains one value for each atlas region, write a morphometry file in which all region vertices are assigned the value. Can be used to plot stuff like p-values or effect sizes onto brain regions.

**Usage**

```
write.region.values.fsaverage(
  hemi,
  atlas,
  region_value_list,
  output_file,
  template_subject = "fsaverage",
  template_subjects_dir = NULL,
  show_freeview_tip = FALSE,
  value_for_unlisted_regions = NaN
)
```

**Arguments**

`hemi`, string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.

`atlas`, string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.

`region_value_list`, named list. A list in which the names are atlas regions, and the values are the value to write to all vertices of that region.

`output_file`, string or 'NULL'. Path of the output file, including file name and extension. The format is determined from the (absence of a) file extension. If NULL, no file will be written.

`template_subject` string, template subject name. Defaults to 'fsaverage'.

`template_subjects_dir` string, the path to the subjects directory containing the template subject directory. If this is 'NULL', the function will try to find it using the environment, see the function [find.subjectsdir.of](#) for details. Defaults to NULL.

`show_freeview_tip` logical, whether to print the freeview command on howto use the overlay to the console. (Only happens if the `output_file` is not 'NULL'.)

`value_for_unlisted_regions`, numeric scalar. The value to assign to vertices which are part of atlas regions that are not listed in `region_value_list`. Defaults to NaN.

**Value**

a named list with the following entries: "data": a vector containing the data. "file\_written": string, path to the file that was written, only exists if `do_write = TRUE`.

**See Also**

Other output functions: [write.region.aggregated\(\)](#), [write.region.values\(\)](#)

# Index

- alphablend, [5](#), [38](#)
- annot.outline, [6](#), [15](#), [127](#)
- apply.label.to.morphdata, [7](#), [8](#), [62](#), [63](#),  
[103](#), [105–107](#), [120](#), [129](#)
- apply.labeldata.to.morphdata, [7](#), [8](#), [62](#),  
[63](#), [103](#), [105–107](#), [120](#), [129](#)
- apply.transform, [8](#)
- arrange.brainview.images, [9](#), [137](#)
  
- brainviews, [10](#)
  
- c, [68](#)
- clip.data, [11](#), [25](#), [27](#)
- cm.cbry, [12](#)
- cm.div, [12](#)
- cm.heat, [13](#)
- cm.qual, [13](#)
- cm.seq, [14](#)
- collayer.bg, [14](#), [16–21](#), [116](#), [117](#), [125](#), [129](#),  
[131](#), [133](#), [136](#)
- collayer.bg.atlas, [15](#), [15](#), [17–21](#)
- collayer.bg.meancurv, [15](#), [16](#), [16](#), [17–21](#)
- collayer.bg.sulc, [15–17](#), [17](#), [18–21](#)
- collayer.from.annot, [15–17](#), [18](#), [19–21](#)
- collayer.from.annotdata, [15–18](#), [18](#), [20](#),  
[21](#)
- collayer.from.mask.data, [15–19](#), [19](#), [20](#),  
[21](#)
- collayer.from.morphlike.data, [15–20](#), [20](#),  
[21](#)
- collayers.merge, [14–20](#), [21](#)
- coloredmesh.from.annot, [21](#), [23](#), [25–28](#), [32](#)
- coloredmesh.from.label, [22](#), [22](#), [25–28](#), [32](#)
- coloredmesh.from.mask, [22](#), [23](#), [24](#), [26–28](#),  
[32](#), [77](#), [122](#)
- coloredmesh.from.morph.native, [22](#), [23](#),  
[25](#), [25](#), [27](#), [28](#), [32](#), [137](#)
- coloredmesh.from.morph.standard, [22](#), [23](#),  
[25](#), [26](#), [26](#), [28](#), [32](#)
- coloredmesh.from.morphdata, [22](#), [23](#),  
[25–27](#), [28](#), [32](#)
- coloredmesh.from.preloaded.data, [29](#)
- coloredmesh.plot.colorbar.separate, [11](#),  
[29](#), [33](#), [34](#), [113](#), [115](#), [117](#), [120](#), [122](#),  
[125](#), [129](#), [131](#), [133](#), [135](#)
- coloredmeshes.from.color, [22](#), [23](#), [25–28](#),  
[31](#)
- colorlist.brain.clusters, [32](#)
- combine.colorbar.with.brainview.animation,  
[31](#), [33](#), [34](#)
- combine.colorbar.with.brainview.image,  
[31](#), [33](#), [34](#)
- cube3D.tris, [35](#), [36](#)
- cubes3D.tris, [36](#)
  
- delete\_all\_optional\_data, [37](#)
- desaturate, [5](#), [37](#)
- download\_fsaverage, [38](#), [41](#)
- download\_optional\_data, [39](#)
  
- export.coloredmesh.ply, [39](#)
  
- face.edges, [40](#), [72](#), [79](#), [108](#), [124](#)
- find.freesurferhome, [41](#)
- find.subjectsdir.of, [41](#), [154](#)
- fs.coloredmesh, [42](#)
- fs.home, [43](#)
- fsaverage.path, [43](#)
- fup, [44](#)
  
- get.atlas.region.names, [6](#), [44](#), [48](#), [50](#), [51](#),  
[56](#), [74](#), [75](#), [88](#), [93–95](#), [97](#), [98](#), [102](#),  
[103](#)
- get.view.angle.names, [10](#), [11](#), [45](#), [137](#)
- get\_optional\_data\_filepath, [47](#)
- getIn, [46](#)
- group.agg.atlas.native, [45](#), [47](#), [50](#), [51](#), [56](#),  
[61](#), [74](#), [75](#), [83](#), [88](#), [93–95](#), [97](#), [98](#),  
[102](#), [103](#)

- group.agg.atlas.standard, [45](#), [48](#), [49](#), [51](#), [56](#), [61](#), [74](#), [75](#), [88](#), [93–95](#), [97](#), [98](#), [102](#), [103](#)
- group.annot, [45](#), [48](#), [50](#), [51](#), [56](#), [74](#), [75](#), [88](#), [93–95](#), [97](#), [98](#), [102](#), [103](#)
- group.concat.measures.native, [52](#), [54](#)
- group.concat.measures.standard, [52](#), [53](#)
- group.label, [54](#), [76](#), [77](#), [101](#)
- group.label.from.annot, [45](#), [48](#), [50](#), [51](#), [55](#), [74](#), [75](#), [88](#), [93–95](#), [97](#), [98](#), [102](#), [103](#)
- group.morph.agg.native, [56](#), [59](#), [65](#), [67](#)
- group.morph.agg.standard, [58](#), [58](#), [65](#), [67](#)
- group.morph.agg.standard.vertex, [48](#), [50](#), [60](#), [98](#)
- group.morph.native, [7](#), [8](#), [61](#), [63](#), [106](#), [107](#)
- group.morph.standard, [7](#), [8](#), [62](#), [62](#), [106](#), [107](#)
- group.multimorph.agg.native, [58](#), [59](#), [64](#), [67](#)
- group.multimorph.agg.standard, [58](#), [59](#), [65](#), [65](#)
  
- hasIn, [67](#)
- hemilist.derive.hemi, [68](#)
- hemilist.get.combined.data, [68](#)
- hemilist.unwrap, [69](#)
- hemilist.wrap, [69](#)
  
- image.plot, [30](#)
- is.fs.coloredmesh, [70](#)
- is.fs.coloredvoxels, [70](#)
- is.hemilist, [71](#)
  
- jet, [23–25](#), [27](#), [28](#)
  
- label.border, [40](#), [71](#), [79](#), [108](#), [124](#)
- label.colFn, [72](#)
- label.colFn.inv, [73](#)
- label.from.annotdata, [45](#), [48](#), [50](#), [51](#), [56](#), [73](#), [75](#), [88](#), [93–95](#), [97](#), [98](#), [102](#), [103](#)
- label.to.annot, [45](#), [48](#), [50](#), [51](#), [56](#), [74](#), [74](#), [88](#), [93–95](#), [97](#), [98](#), [102](#), [103](#)
- labeldata.from.mask, [55](#), [75](#), [77](#), [101](#)
- legend, [114](#)
- list\_optional\_data, [76](#)
  
- makecmap, [19](#), [20](#), [23](#), [24](#), [26–29](#), [115](#), [117](#), [119](#), [120](#), [122](#), [125](#), [129](#), [131](#), [133](#), [135](#)
- mask.from.labeldata.for.hemi, [25](#), [55](#), [76](#), [77](#), [101](#), [121](#), [122](#)
  
- material3d, [137](#)
- max, [57](#), [59](#), [60](#), [64](#), [66](#)
- mean, [57](#), [59](#), [61](#), [65](#), [67](#)
- mesh.vertex.included.faces, [40](#), [72](#), [79](#), [108](#), [124](#)
- mesh.vertex.neighbors, [40](#), [72](#), [78](#), [108](#), [124](#)
- mkco.cluster, [79](#)
- mkco.div, [80](#)
- mkco.heat, [80](#)
- mkco.seq, [81](#)
  
- par3d, [10](#), [111](#), [113](#), [115](#), [117](#), [120](#), [122](#), [125](#), [127](#), [128](#), [131](#), [133](#), [135](#), [137](#)
- png, [30](#)
- print.fs.coloredmesh, [81](#)
- print.fs.coloredvoxels, [82](#)
  
- qc.for.group, [82](#), [83](#), [85](#)
- qc.from.regionwise.df, [82](#), [83](#), [83](#), [84](#)
- qc.from.segstats.table, [83](#)
- qc.from.segstats.tables, [84](#), [85](#)
- qc.vis.failcount.by.region, [84](#)
  
- ras2vox\_tkr, [85](#), [150](#)
- read.colorcsv, [86](#)
- read.fs.annot, [75](#), [96](#)
- read.fs.colortable, [114](#), [144](#)
- read.fs.mgh, [109](#)
- read.fs.morph, [106](#), [118](#)
- read.fs.surface, [6](#), [40](#), [71](#), [78](#), [108](#), [118](#), [124](#)
- read.md.demographics, [86](#), [88](#), [89](#)
- read.md.subjects, [87](#), [87](#)
- read.table, [87](#)
- regions.to.ignore, [45](#), [48](#), [50](#), [51](#), [56](#), [74](#), [75](#), [88](#), [93–95](#), [97](#), [98](#), [102](#), [103](#)
- report.on.demographics, [87](#), [89](#)
- rgb, [21](#)
- rgl.spheres, [90](#)
- rglot, [90](#)
- rglvoxels, [90](#)
  
- shift.hemis.apart, [91](#)
- spread.values.over.annot, [45](#), [48](#), [50](#), [51](#), [56](#), [74](#), [75](#), [88](#), [92](#), [94](#), [95](#), [97](#), [98](#), [102](#), [103](#)
- spread.values.over.hemi, [45](#), [48](#), [50](#), [51](#), [56](#), [74](#), [75](#), [88](#), [93](#), [93](#), [95](#), [97](#), [98](#), [102](#), [103](#)

- spread.values.over.subject, [45](#), [48](#), [50](#),  
[51](#), [56](#), [74](#), [75](#), [88](#), [93](#), [94](#), [94](#), [97](#), [98](#),  
[102](#), [103](#)
- subject.annot, [6](#), [19](#), [45](#), [48](#), [50](#), [51](#), [56](#), [74](#),  
[75](#), [88](#), [93–95](#), [96](#), [98](#), [102](#), [103](#), [114](#)
- subject.atlas.agg, [45](#), [48](#), [50](#), [51](#), [56](#), [61](#),  
[74](#), [75](#), [88](#), [93–95](#), [97](#), [97](#), [102](#), [103](#)
- subject.filepath.morph.native, [98](#)
- subject.filepath.morph.standard, [99](#)
- subject.label, [8](#), [55](#), [76](#), [77](#), [100](#)
- subject.label.from.annot, [45](#), [48](#), [50](#), [51](#),  
[56](#), [74](#), [75](#), [88](#), [93–95](#), [97](#), [98](#), [101](#),  
[103](#)
- subject.lobes, [7](#), [8](#), [45](#), [48](#), [50](#), [51](#), [56](#), [74](#),  
[75](#), [88](#), [93–95](#), [97](#), [98](#), [102](#), [103](#), [105](#),  
[120](#), [129](#)
- subject.mask, [7](#), [8](#), [103](#), [104](#), [120](#), [129](#)
- subject.morph.native, [7](#), [8](#), [62](#), [63](#), [105](#),  
[107](#), [110](#)
- subject.morph.standard, [7](#), [8](#), [62](#), [63](#), [106](#),  
[106](#)
- subject.surface, [6](#), [24](#), [40](#), [42](#), [71](#), [72](#), [78](#),  
[79](#), [108](#), [118](#), [124](#)
- subject.volume, [109](#)
- tmesh3d, [22–24](#), [26–28](#), [32](#), [42](#)
- toupper, [44](#)
- triangles3d, [35](#), [90](#), [150](#)
- vdata.split.by.hemi, [110](#)
- vis.color.on.subject, [15–20](#), [111](#), [116](#),  
[117](#), [120](#), [122](#), [126](#), [127](#), [129](#), [131](#),  
[134](#), [136](#), [137](#)
- vis.coloredmeshes, [112](#), [119](#)
- vis.colortable.legend, [113](#)
- vis.data.on.fsaverage, [112](#), [114](#), [117](#), [118](#),  
[120](#), [122](#), [126](#), [127](#), [129](#), [131](#), [134](#),  
[136](#), [137](#)
- vis.data.on.subject, [112](#), [116](#), [116](#), [120](#),  
[122](#), [126](#), [127](#), [129](#), [131](#), [134](#), [136](#),  
[137](#)
- vis.fs.surface, [118](#)
- vis.labeldata.on.subject, [7](#), [8](#), [103](#), [105](#),  
[112](#), [116](#), [117](#), [119](#), [122](#), [126](#), [127](#),  
[129](#), [131](#), [134](#), [136](#), [137](#)
- vis.mask.on.subject, [25](#), [77](#), [112](#), [116](#), [117](#),  
[120](#), [121](#), [126](#), [127](#), [129](#), [131](#), [134](#),  
[136](#), [137](#)
- vis.path.along.verts, [40](#), [72](#), [79](#), [108](#), [123](#)
- vis.region.values.on.subject, [85](#), [112](#),  
[116](#), [117](#), [120](#), [122](#), [124](#), [127](#), [129](#),  
[131](#), [134](#), [136](#), [137](#)
- vis.subject.annot, [112](#), [116](#), [117](#), [120](#), [122](#),  
[126](#), [126](#), [129](#), [131](#), [134](#), [136](#), [137](#)
- vis.subject.label, [7](#), [8](#), [103](#), [105](#), [112](#), [116](#),  
[117](#), [120](#), [122](#), [126](#), [127](#), [128](#), [131](#),  
[134](#), [136](#), [137](#)
- vis.subject.morph.native, [30](#), [39](#), [45](#),  
[79–81](#), [90](#), [112](#), [116](#), [118](#), [120](#), [122](#),  
[123](#), [126](#), [127](#), [129](#), [130](#), [134](#), [136](#),  
[137](#), [149](#)
- vis.subject.morph.standard, [112](#), [116](#),  
[118](#), [120](#), [122](#), [126](#), [127](#), [129](#), [131](#),  
[132](#), [136](#), [137](#)
- vis.symmetric.data.on.subject, [112](#), [116](#),  
[118](#), [120](#), [122](#), [126](#), [127](#), [129](#), [131](#),  
[134](#), [134](#), [137](#)
- vislayout.from.coloredmeshes, [45](#), [112](#),  
[116](#), [118](#), [120](#), [122](#), [126](#), [127](#), [129](#),  
[131](#), [134](#), [136](#), [136](#)
- vol.boundary.box, [138](#), [141](#), [143–146](#)
- vol.boundary.box.apply, [139](#)
- vol.boundary.mask, [138](#)
- vol.hull, [139](#)
- vol.imagestack, [139](#), [140](#), [143–146](#)
- vol.intensity.to.color, [141](#)
- vol.mask.from.segmentation, [142](#)
- vol.merge, [139](#), [141](#), [142](#), [144–146](#)
- vol.overlay.colors.from.activation,  
[139](#), [141](#), [143](#), [143](#), [145](#), [146](#)
- vol.overlay.colors.from.colortable,  
[144](#)
- vol.planes, [139](#), [141](#), [143](#), [144](#), [145](#), [146](#)
- vol.slice, [139](#), [141](#), [143–145](#), [145](#)
- vol.vox.from.crs, [146](#)
- volvis.contour, [147](#)
- volvis.lightbox, [145](#), [148](#)
- volvis.voxels, [149](#)
- vox2ras\_tkr, [85](#), [149](#), [150](#)
- which, [75](#)
- write.region.aggregated, [151](#), [153](#), [154](#)
- write.region.values, [152](#), [152](#), [154](#)
- write.region.values.fsaverage, [152](#), [153](#),  
[153](#)