# Package 'neuroim'

January 7, 2016

**Imports** Matrix, yaImpute, Rcpp, iterators, abind, assertthat, readr, rgl

**LinkingTo** Rcpp

**License** GPL (>= 2)

**Maintainer** Bradley Buchsbaum <brad.buchsbaum@gmail.com>

**Type** Package

**Author** Bradley R. Buchsbaum

**Title** Data Structures and Handling for Neuroimaging Data

**Description** A collection of data structures that represent volumetric brain imaging data. The focus is on basic data handling for 3D and 4D neuroimaging data. In addition, there are function to read and write NIFTI files and limited support for reading AFNI files.

**Date** 2016-01-06

**Suggests** foreach, testthat, knitr

**Version** 0.0.6

**Depends** R (>= 3.0.0), stringr, hash, methods, stats, grDevices, grid, utils

**VignetteBuilder** knitr

**Collate** 'AFNI_IO.R' 'AllGeneric.R' 'AllClass.R' 'Axis.R' 'BinaryIO.R' 'BrainData.R' 'common.R' 'NIFTI_IO.R' 'BrainFileDescriptor.R' 'BrainMetaInfo.R' 'BrainRegion3D.R' 'BrainSlice.R' 'BrainSpace.R' 'BrainSurface.R' 'SparseBrainVector.R' 'BrainVector.R' 'BrainVolume.R' 'Display.R' 'FREESURFER_IO.R' 'IndexLookupVolume.R' 'Ops.R' 'RcppExports.R' 'conncomp.R' 'datadoc.R' 'neuroim.R'

**RoxygenNote** 5.0.1

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2016-01-07 23:59:04

# R topics documented:

---

`addDim` *Generic function to add a dimension to an object*

---

## Description

Generic function to add a dimension to an object

add dimension to `BrainSpace`

## Usage

```
addDim(x, n)

## S4 method for signature 'BrainSpace,numeric'
addDim(x, n)
```

## Arguments

| | |
|---|---|
| `x` | a dimensioned object |
| `n` | the size of the dimension to add |

## Examples

```
x = BrainSpace(c(10,10,10), c(1,1,1))
x1 <- addDim(x, 10)
ndim(x1) == 4
dim(x1)[4] == 10
```

## AFNIFileDescriptor-class *AFNIFileDescriptor*

### Description

This class supports the AFNI file format

## AFNIMetaInfo *AFNIMetaInfo*

### Description

Constructor for AFNIMetaInfo class

### Usage

```
AFNIMetaInfo(descriptor, afni_header)
```

### Arguments

| | |
|---|---|
| descriptor | an instance of class AFNIFileDescriptor |
| afni_header | a list returned by readAFNIHeader |

### Value

an instance of class AFNIMetaInfo

## as *conversion from DenseBrainVolume to array*

### Description

conversion from DenseBrainVolume to array

conversion from SparseBrainVolume to array

conversion from SparseBrainVolume to numeric

conversion from BrainVolume to LogicalBrainVolume

conversion from DenseBrainVolume to LogicalBrainVolume

conversion from ClusteredBrainVolume to LogicalBrainVolume

conversion from BrainVolume to array

---

`as.array,BrainData-method`
*convert* `BrainData` *instance to array*

---

## Description

convert `BrainData` instance to array

## Usage

```
## S4 method for signature 'BrainData'
as.array(x)
```

## Arguments

| | |
|---|---|
| `x` | the object |

---

`as.list,SparseBrainVector-method`
*as.list*

---

## Description

convert SparseBrainVector to list of `DenseBrainVolume`

convert a `BrainVector` to `list` of volumes.

## Usage

```
## S4 method for signature 'SparseBrainVector'
as.list(x)

## S4 method for signature 'BrainVector'
as.list(x)
```

## Arguments

| | |
|---|---|
| `x` | the object |

## as.logical,BrainVolume-method *as.logical*

### Description

Convert BrainVolume to linkS4class{LogicalBrainVolume}

### Usage

```
## S4 method for signature 'BrainVolume'
as.logical(x)
```

### Arguments

| | |
|---|---|
| x | the object |

### Details

the image values will be converted to using R base function as.logical and wrapped in LogicalBrainVolume

### Value

an instance of linkS4class{LogicalBrainVolume}

## as.mask *Convert to a LogicalBrainVolume*

### Description

Convert to a LogicalBrainVolume

### Usage

```
as.mask(x, indices)

## S4 method for signature 'BrainVolume,missing'
as.mask(x)

## S4 method for signature 'BrainVolume,numeric'
as.mask(x, indices)
```

### Arguments

| | |
|---|---|
| x | the object to binarize |
| indices | the indices to set to TRUE |

---

`as.matrix,BrainData-method`
*convert* `BrainData` *instance to matrix*

---

## Description

convert `BrainData` instance to matrix

## Usage

```
## S4 method for signature 'BrainData'
as.matrix(x)
```

## Arguments

| | |
|---|---|
| `x` | the object |

---

`as.matrix,SparseBrainVector-method`
*as.matrix*

---

## Description

convert SparseBrainVector to matrix

convert a `DenseBrainVector` to a matrix

## Usage

```
## S4 method for signature 'SparseBrainVector'
as.matrix(x)

## S4 method for signature 'DenseBrainVector'
as.matrix(x)
```

## Arguments

| | |
|---|---|
| `x` | the object |

## as.numeric,SparseBrainVolume-method

*Convert SparseBrainVolume to numeric*

### Description

Convert SparseBrainVolume to numeric

### Usage

```
## S4 method for signature 'SparseBrainVolume'
as.numeric(x)
```

### Arguments

| | |
|---|---|
| x | the object to convert |

## as.raster,Layer-method

*as.raster*

### Description

as.raster

### Usage

```
## S4 method for signature 'Layer'
as.raster(x, zpos)
```

### Arguments

| | |
|---|---|
| x | the layer to convert |
| zpos | the z coordinate in coordinate space |

as.sparse *Convert to from dense to sparse representation*

## Description

Convert to from dense to sparse representation

## Usage

```
as.sparse(x, mask, ...)

## S4 method for signature 'DenseBrainVector,LogicalBrainVolume'
as.sparse(x, mask)

## S4 method for signature 'DenseBrainVector,numeric'
as.sparse(x, mask)

## S4 method for signature 'DenseBrainVolume,LogicalBrainVolume'
as.sparse(x, mask)

## S4 method for signature 'DenseBrainVolume,numeric'
as.sparse(x, mask)
```

## Arguments

| | |
|---|---|
| x | the object to make sparse, e.g. DenseBrainVolume or DenseBrainVector |
| mask | the elements to retain |
| ... | additional arguments |

## Details

mask can be an integer vector of 1D indices or a mask volume of class LogicalBrainVolume

## Examples

```
bvol <- BrainVolume(array(runif(24*24*24), c(24,24,24)), BrainSpace(c(24,24,24), c(1,1,1)))
indmask <- sort(sample(1:(24*24*24), 100))
svol <- as.sparse(bvol, indmask)


mask <- LogicalBrainVolume(runif(length(indmask)), space=space(bvol), indices=indmask)
sum(mask) == 100
```

## as.vector,BrainData-method *convert* BrainData *instance to vector*

### Description

convert BrainData instance to vector

### Usage

```
## S4 method for signature 'BrainData'
as.vector(x)
```

### Arguments

| | |
|---|---|
| x | the object |

## axes *Generic getter function to extract image axes*

### Description

Generic getter function to extract image axes

### Usage

```
axes(x)

## S4 method for signature 'BrainSpace'
axes(x)

## S4 method for signature 'BrainData'
axes(x)
```

### Arguments

| | |
|---|---|
| x | an object with a set of axes |

`AxisSet-class` *Base*

## Description

Virtual base class representing an ordered set of named axes.

## Slots

`ndim` the number of axes (or dimensions)

`AxisSet1D-class` *AxisSet1D*

## Description

A one-dimensional axis set

## Slots

`i` the first axis

`AxisSet2D-class` *AxisSet2D*

## Description

A two-dimensional axis set

## Slots

`j` the second axis

`AxisSet3D-class` *AxisSet3D*

## Description

A three-dimensional axis set

## Slots

`k` the third axis

## AxisSet4D-class *AxisSet4D*

### Description

A four-dimensional axis set

### Slots

`l` the fourth axis

## AxisSet5D-class *AxisSet5D*

### Description

A five-dimensional axis set

### Slots

`m` the fifth axis

## axisToIndex *Generic function to convert 1-dimensional real axis coordinates along a single axis dimension to an 1D index along the same axis*

### Description

Generic function to convert 1-dimensional real axis coordinates along a single axis dimension to an 1D index along the same axis

### Usage

```
axisToIndex(x, real, dimNum)

## S4 method for signature 'BrainSpace,numeric,numeric'
axisToIndex(x, real, dimNum)
```

### Arguments

| | |
|---|---|
| `x` | the object |
| `real` | the axis coordinates |
| `dimNum` | the dimension number of the axis (e.g. 1, 2, 3) |

### Value

a vector of axis indices

## Base-class *Base*

### Description

Generic S4 Base class

## BaseMetaInfo-class *BaseMetaInfo*

### Description

This is a base class to represent meta information

## BaseSource-class *BaseSource*

### Description

This is a base class to represent a data source

### Slots

`metaInfo` meta information for the data source

## BinaryReader *BinaryReader*

### Description

Constructor for `BinaryReader` class

### Usage

```
BinaryReader(input, byteOffset, dataType, bytesPerElement,
  endian = .Platform$endian)
```

### Arguments

| | |
|---|---|
| `input` | file name to read from or else a `connection` object |
| `byteOffset` | the number of bytes to skip at the start of input |
| `dataType` | R data type of binary elements |
| `bytesPerElement` | number of bytes in each data element (e.g. 4 or 8 for floating point numbers) |
| `endian` | endianness of binary input connection |

## BinaryReader-class *BinaryReader*

### Description

This class supports reading of bulk binary data from a connection

### Slots

`input` the binary input connection

`byteOffset` the number of bytes to skip at the start of input

`dataType` the dataType of the binary Elements

`bytesPerElement` number of bytes in each data element (e.g. 4 or 8 for floating point numbers)

`endian` endianness of binary input connection

## BinaryWriter-class *BinaryWriter*

### Description

This class supports writing of bulk binary data to a connection

Constructor for `BinaryWriter` class

### Usage

```
BinaryWriter(output, byteOffset, dataType, bytesPerElement,
  endian = .Platform$endian)
```

### Arguments

| | |
|---|---|
| `output` | file name to write to or else a `connection` object |
| `byteOffset` | the number of bytes to skip at the start of output |
| `dataType` | R data type of binary elements |
| `bytesPerElement` | number of bytes in each data element (e.g. 4 or 8 for floating point numbers) |
| `endian` | endianness of binary output connection |

### Slots

`output` the binary output connection

`byteOffset` the number of bytes to skip at the start of input

`dataType` the dataType of the binary Elements

`bytesPerElement` number of bytes in each data element (e.g. 4 or 8 for floating point numbers)

`endian` endianness of binary output connection

---

`BootstrapSearchlight` *Create a searchlight iterator that samples regions from within a mask. Searchlight centers are sampled *without* replacement, but the same voxel can belong to multiple searchlight samples. It is in the latter sense that this is a bootstrap resampling scheme.*

---

## Description

Create a searchlight iterator that samples regions from within a mask. Searchlight centers are sampled *without* replacement, but the same voxel can belong to multiple searchlight samples. It is in the latter sense that this is a bootstrap resampling scheme.

## Usage

```
BootstrapSearchlight(mask, radius, iter = 100)
```

## Arguments

| | |
|---|---|
| `mask` | an image volume containing valid central voxels for roving searchlight |
| `radius` | in mm of spherical searchlight |
| `iter` | the total number of searchlights to sample (default is 100) |

---

`bounds` *Generic function to extract the spatial bounds (origin + dim * spacing) of an image param x the object*

---

## Description

Generic function to extract the spatial bounds (origin + dim * spacing) of an image param x the object

bounds

## Usage

```
bounds(x)

## S4 method for signature 'BrainSpace'
bounds(x)

## S4 method for signature 'BrainData'
bounds(x)
```

## Arguments

| | |
|---|---|
| `x` | the object with bounds property |

## Value

a matrix where each row contains the min (column 1) and max (column 2) bounds of the image dimension from 1 to ndim(image).

## Examples

```
bspace <- BrainSpace(c(10,10,10), c(2,2,2))
b <- bounds(bspace)
nrow(b) == ndim(bspace)
ncol(b) == 2
```

---

# BrainBucket-class *BrainBucket*

---

## Description

a four-dimensional image that conists of a sequence of labeled image volumes backed by a list

Constructor function for BrainBucket class

## Usage

```
BrainBucket(volumeList)
```

## Arguments

| | |
|---|---|
| volumeList | a named list of BrainVolume instances |

## Value

an instance of class BrainBucket

## Slots

source the data source for the bucket volumes

labels the names of the sub-volumes contained in the bucket

data a list of BrainVolume instances with names corresponding to volume labels

## Examples

```
vol1 <- BrainVolume(rnorm(24*24*24), BrainSpace(c(24,24,24), c(1,1,1)))
vol2 <- BrainVolume(rnorm(24*24*24), BrainSpace(c(24,24,24), c(1,1,1)))
vol3 <- BrainVolume(rnorm(24*24*24), BrainSpace(c(24,24,24), c(1,1,1)))
vlist <- list(vol1,vol2,vol3)
names(vlist) <- paste0("V", 1:3)
bucket <- BrainBucket(vlist)
all.equal(dim(bucket[[1]]), dim(vol1))
```

## BrainBucketSource-class *BrainBucketSource*

### Description

A class that is used to produce a `BrainBucket` instance

Constructor function for `BrainBucketSource` class

### Usage

```
BrainBucketSource(fileName, pattern = NULL, indices = NULL)
```

### Arguments

| | |
|---|---|
| `fileName` | the name of the bucket file |
| `pattern` | optional regular expression used to filter the sub-volumes using associated labels |
| `indices` | optional set of sub-volume indices to load |

### Slots

`sourceList` a list of sources for the bucket sub-volumes

`cache` a cache used to store data in memory

## BrainData-class *BrainData*

### Description

Base class for brain image data

### Slots

`source` an instance of class `BaseSource` to store the source of the data

`space` an instance of class `BrainSpace` to represent the geometry of the data space

`BrainFileDescriptor-class` *BrainFileDescriptor*

## Description

This class represents a neuroimaging file format

## Slots

`fileFormat` the name of the file format (e.g. NIfTI)

`headerEncoding` the file encoding of the header file (e.g. 'raw' for binary, 'gzip' for gz compressed')

`headerExtension` the file extension for the header file (e.g. 'nii' for NIfTI single files)

`dataEncoding` the file encoding for the data file

`dataExtension` the file extension for the data file (e.g. 'nii' for NIfTI single files)

`BrainFileSource-class` *BrainFileSource Base class for representing a data source for images. The purpose of this class is to provide a layer in between low level IO and image loading functionality.*

## Description

BrainFileSource

Base class for representing a data source for images. The purpose of this class is to provide a layer in between low level IO and image loading functionality.

## Slots

`metaInfo` meta information for the data source

`BrainMetaInfo-class` *BrainMetaInfo This class contains meta information from an image*

## Description

BrainMetaInfo

This class contains meta information from an image

This class contains meta information for an image

## Usage

```
BrainMetaInfo(Dim, spacing, origin = rep(0, length(spacing)),
  dataType = "FLOAT", label = "",
  spatialAxes = OrientationList3D$AXIAL_LPI, additionalAxes = NullAxis)
```

## Arguments

| | |
|---|---|
| `Dim` | image dimensions |
| `spacing` | voxel dimensions |
| `origin` | coordinate origin |
| `dataType` | the type of the data (e.g. "FLOAT") |
| `label` | name(s) of images |
| `spatialAxes` | image axes for spatial dimensions (x,y,z) |
| `additionalAxes` | axes for dimensions > 3 (e.g. time, color band, direction) |

## Value

an instance of class `BrainMetaInfo`

## Slots

`dataType` the data type code, e.g. FLOAT

`Dim` image dimensions

`spatialAxes` image axes for spatial dimensions (x,y,z)

`additionalAxes` axes for dimensions > 3 (e.g. time, color band, direction)

`spacing` voxel dimensions

`origin` coordinate origin

`label` name(s) of images

## BrainSlice *BrainSlice constructor*

### Description

BrainSlice constructor

### Usage

```
BrainSlice(data, space, indices = NULL)
```

### Arguments

| | |
|---|---|
| data | data vector or matrix |
| space | an instance of class BrainSpace |
| indices | linear indices corresponding used if data is a 1D vector. |

### Examples

```
bspace <- BrainSpace(c(64,64), spacing=c(1,1))
dat <- array(rnorm(64*64), c(64,64))
bslice <- BrainSlice(dat,bspace)
print(bslice)
```

## BrainSlice-class *BrainSlice*

### Description

Two-dimensional brain image

## BrainSource-class *BrainSource*

### Description

Base class for representing a data source for images. The purpose of this class is to provide a layer in between low level IO and image loading functionality.

### Slots

metaInfo meta information for the data source

BrainSpace *Constructor function for* BrainSpace *class*

## Description

Constructor function for BrainSpace class

## Usage

```
BrainSpace(Dim, spacing = NULL, origin = NULL, axes = NULL,
  trans = NULL)
```

## Arguments

| | |
|---|---|
| Dim | a vector describing the dimensions of the spatial grid |
| spacing | the real-valued voxel dimensions (usually in millimeters) |
| origin | the coordinate origin of the image space |
| axes | the image axes ordering (default is based on the NIFTI standard, Left-Posterior-Inferior) |
| trans | a matrix representing the coordinate transformation associated with the image space (default is based on the NIFTI standard, Left-Posterior-Inferior) |

## Value

an instance of class BrainSpace

## Note

one should rarely need to create a new BrainSpace instance, as it will almost always be created automatically using information stored in an image header. Also, If one already has an existing image object, its BrainSpace instance can be easily extracted with the space method.

## Examples

```
bspace <- BrainSpace(c(64,64,64), origin=c(0,0,0), spacing=c(2,2,2))
print(bspace)
origin(bspace)
axes(bspace)
trans(bspace)
```

## BrainSpace-class *BrainSpace*

### Description

This class represents the geometry of a brain image

### Slots

`Dim` the grid dimensions of the image

`origin` the coordinates of the spatial origin

`spacing` the dimensions (in mm) of the grid units (voxels)

`axes` the set of named spatial axes

`trans` an affine transformation matrix that moves from grid -> real world coordinates

`inverseTrans` an inverse matrix that moves from real world -> grid coordinates

## BrainSurface-class *BrainSurface*

### Description

a three-dimensional surface consisting of a set of triangle vertices with one value per vertex.

### Slots

`source` the data source for the surface

`mesh` the underlying `mesh3d` object

`data` the vector of data value at each vertex of the mesh

## BrainSurfaceSource-class *BrainSurfaceSource*

### Description

A class that is used to produce a `BrainSurface` instance

Constructor for BrainSurfaceSource

### Usage

```
BrainSurfaceSource(surfaceName, surfaceDataName, index = 1)
```

### Arguments

| | |
|---|---|
| surfaceName | the name of the file containing the surface geometry. |
| surfaceDataName | the name of the file containing the data values to be mapped to the surface. |
| index | the integer offset into the surface data matrix |

### Slots

metaInfo a SurfaceGeometryMetaInfo instance

dataMetaInfo a SurfaceDataMetaInfo instance

index the index offset into the surface data matrix

---

## BrainSurfaceVector-class

*BrainSurfaceVector*

---

### Description

a three-dimensional surface consisting of a set of triangle vertices with multiple values per vertex.

### Slots

source the data source for the surface

mesh the underlying mesh3d object

mat a matrix of values where each column contains a vector of values over the surface nodes.

---

## BrainSurfaceVectorSource-class

*BrainSurfaceVectorSource*

---

### Description

A class that is used to produce a BrainSurfaceVectorSource instance

### Slots

indices the index vector of the volumes to be loaded

## BrainVector-class *BrainVector*

### Description

Four-dimensional brain image

constructor function for virtual class BrainVector

### Usage

```
BrainVector(data, space = NULL, mask = NULL, source = NULL, label = "")
```

### Arguments

| | |
|---|---|
| data | the image data which can be a matrix, a 4d array, or a list of BrainVolumes. If the latter, the geometric space of the data BrainSpace will be inferred from the constituent volumes, which must all be identical. |
| space | a BrainSpace object. Does not ned to be included if data argument is a list of BrainVolumes |
| mask | an optional array of type logical |
| source | an optional BrainSource object |
| label | a label of type character |

### Value

a concrete instance of BrainVector class. If mask is provided then SparseBrainVector, otherwise DenseBrainVector

## BrainVectorSource *BrainVectorSource*

### Description

Construct a BrainVectorSource object

### Usage

```
BrainVectorSource(fileName, indices = NULL, mask = NULL)
```

### Arguments

| | |
|---|---|
| fileName | name of the 4-dimensional image file |
| indices | the subset of integer volume indices to load – if NULL then all volumes will be loaded |
| mask | image volume indicating the subset of voxels that will be loaded. If provided, function returns SparseBrainVectorSource |

### Details

If a mask is supplied then it should be a LogicalBrainVolume or BrainVolume instance. If the latter, then the mask will be defined by nonzero elements of the volume.

### Value

a instance deriving from BrainVectorSource

---

## BrainVectorSource-class *BrainVectorSource*

---

### Description

A class that is used to produce a BrainVectorSource instance

### Slots

indices the index vector of the volumes to be loaded

---

## BrainVolume *BrainVolume*

---

### Description

Construct a BrainVolume instance, using default (dense) implementation

### Usage

```
BrainVolume(data, space, source = NULL, label = "", indices = NULL)
```

### Arguments

| | |
|---|---|
| data | a three-dimensional array |
| space | an instance of class BrainSpace |
| source | an instance of class BrainSource |
| label | a character string to identify volume |
| indices | an 1D vector that gives the linear indices of the associated data vector |

### Value

a DenseBrainVolume instance

## Examples

```
bspace <- BrainSpace(c(64,64,64), spacing=c(1,1,1))
dat <- array(rnorm(64*64*64), c(64,64,64))
bvol <- BrainVolume(dat,bspace, label="test")
print(bvol)
```

---

# BrainVolume-class *Base class for image representing 3D volumetric data.*

---

## Description

Base class for image representing 3D volumetric data.

---

# BrainVolumeSource-class *BrainVolume BrainVolumeSource A class is used to produce a* BrainVolume *instance*

---

## Description

BrainVolume BrainVolumeSource

A class is used to produce a BrainVolume instance

Constructor for BrainVolumeSource

## Usage

```
BrainVolumeSource(input, index = 1)
```

## Arguments

| | |
|---|---|
| input | the input file name |
| index | the image subvolume index |

## Slots

index the index of the volume to be read – must be of length 1.

close,BinaryReader-method *close*

## Description

close

## Usage

```
## S4 method for signature 'BinaryReader'
close(con)

## S4 method for signature 'BinaryWriter'
close(con)
```

## Arguments

| | |
|---|---|
| con | the object to close |

clusterCenters *clusterCenters*

## Description

clusterCenters

## Usage

```
clusterCenters(x, features, FUN)

## S4 method for signature 'ClusteredBrainVolume,matrix,missing'
clusterCenters(x, features)
```

## Arguments

| | |
|---|---|
| x | the object to extract cluster centers from |
| features | additional features |
| FUN | a user-supplied function |

ClusteredBrainVolume-class
*ClusteredBrainVolume*

## Description

Three-dimensional brain image that is divided into N disjoint partitions

Construct a ClusteredBrainVolume instance

## Usage

```
ClusteredBrainVolume(mask, clusters, labelMap = NULL, source = NULL,
  label = "")
```

## Arguments

| | |
|---|---|
| mask | an instance of class LogicalBrainVolume |
| clusters | a vector of clusters ids with length equal to number of nonzero voxels in mask mask |
| labelMap | an optional list that maps from cluster id to a cluster label, e.g. (1 -> "FFA", 2 -> "PPA") |
| source | an optional instance of class BrainSource |
| label | an optional character string used to label of the volume |

## Value

ClusteredBrainVolume instance

## Examples

```
bspace <- BrainSpace(c(16,16,16), spacing=c(1,1,1))
grid <- indexToGrid(bspace, 1:(16*16*16))
kres <- kmeans(grid, centers=10)
mask <- BrainVolume(rep(1, 16^3),bspace)
clusvol <- ClusteredBrainVolume(mask, kres$cluster)
```

## ColumnReader *ColumnReader*

### Description

Constructor for `ColumnReader` class

### Usage

```
ColumnReader(nrow, ncol, reader)
```

### Arguments

| | |
|---|---|
| `nrow` | the number of rows |
| `ncol` | the number of columns |
| `reader` | a function that takes a set of column indices and returns a `matrix` |

## ColumnReader-class *ColumnReader*

### Description

This class supports reading of data froma matrix-like stroage format

### Slots

`nrow` the number of rows

`ncol` the number of columns

`reader` a function that takes a set of column indices and returns a `matrix`

## concat *Concatenate two objects*

### Description

Concatenate two objects

## Usage

```
concat(x, y, ...)

## S4 method for signature 'SparseBrainVector,SparseBrainVector'
concat(x, y, ...)

## S4 method for signature 'BrainVector,BrainVolume'
concat(x, y, ...)

## S4 method for signature 'BrainVolume,BrainVector'
concat(x, y, ...)

## S4 method for signature 'BrainVector,BrainVector'
concat(x, y, ...)

## S4 method for signature 'DenseBrainVolume,DenseBrainVolume'
concat(x, y, ...)
```

## Arguments

| | |
|---|---|
| x | the first object, typically BrainVolume or BrainVector |
| y | the second object, typically BrainVolume or BrainVector |
| ... | additional objects |

## Details

The x and y images must have compatible dimensions. a BrainVolume can be concatenated to BrainVector, and vice versa. See examples.

## Note

dimensions of x and y must be equal

## Examples

```
bv1 <- BrainVolume(rep(1,1000), BrainSpace(c(10,10,10), c(1,1,1)))
bv2 <- BrainVolume(rep(2,1000), BrainSpace(c(10,10,10), c(1,1,1)))
bv3 <- concat(bv1,bv2)
inherits(bv3, "BrainVector")

bv4 <- concat(bv3, bv1)
dim(bv4)[4] == 3
bv5 <- concat(bv1, bv3)
dim(bv4)[4] == 3

bv6 <- concat(bv4,bv5)
dim(bv6)[4] == 6
```

## connComp *Find connected components*

### Description

Find connected components

find connected components in BrainVolume

### Usage

```
connComp(x, ...)

## S4 method for signature 'BrainVolume'
connComp(x, threshold = 0, clusterTable = TRUE,
  localMaxima = TRUE, localMaximaDistance = 15)
```

### Arguments

| | |
|---|---|
| x | the image object |
| ... | additonal arguments |
| threshold | threshold defining lower intensity bound for image mask |
| clusterTable | return clusterTable |
| localMaxima | return table of local maxima |
| localMaximaDistance | the distance used to define minum distance between local maxima |

## connComp3D *Extract connected components from a 3D mask*

### Description

Extract connected components from a 3D mask

### Usage

```
connComp3D(mask)
```

### Arguments

| | |
|---|---|
| mask | a 3D binary array |

### Value

a two-element list of the connected components (cluster `index` and cluster `size`) The first element `index` is a 3D array containing the cluster index of the connected component for each voxel. The second element `size` is a 3D array consisting of the size of the connected component inhabited by each voxel.

## coords *Extract coordinates*

### Description

Extract coordinates

coords

### Usage

```
coords(x, ...)

## S4 method for signature 'ROIVolume'
coords(x)

## S4 method for signature 'SparseBrainVector'
coords(x, i)

## S4 method for signature 'IndexLookupVolume'
coords(x, i)
```

### Arguments

| | |
|---|---|
| x | the object to extract coordinates from |
| ... | additional arguments |
| i | the index in to the lookup volume |

## coordToGrid *Generic function to convert N-dimensional real world coordinates to grid coordinates*

### Description

Generic function to convert N-dimensional real world coordinates to grid coordinates

### Usage

```
coordToGrid(x, coords)

## S4 method for signature 'BrainSpace,matrix'
coordToGrid(x, coords)

## S4 method for signature 'BrainSpace,numeric'
coordToGrid(x, coords)

## S4 method for signature 'BrainVolume,matrix'
coordToGrid(x, coords)
```

### Arguments

| | |
|---|---|
| x | the object |
| coords | a matrix of real world coordinates |

### Value

a matrix of grid coordinates

---

## coordToIndex *Generic function to convert N-dimensional real world coordinates to 1D indices*

---

### Description

Generic function to convert N-dimensional real world coordinates to 1D indices

### Usage

```
coordToIndex(x, coords)

## S4 method for signature 'BrainSpace,matrix'
coordToIndex(x, coords)

## S4 method for signature 'BrainSpace,numeric'
coordToIndex(x, coords)

## S4 method for signature 'BrainVolume,matrix'
coordToIndex(x, coords)
```

### Arguments

| | |
|---|---|
| x | the object |
| coords | a matrix of real world coordinates |

### Value

a vector of indices

---

`dataFile` *Generic function to get the name of the data file, given a file name and a* `BrainFileDescriptor` *instance.*

---

## Description

Generic function to get the name of the data file, given a file name and a `BrainFileDescriptor` instance.

## Usage

```
dataFile(x, fileName)

## S4 method for signature 'BrainFileDescriptor,character'
dataFile(x, fileName)
```

## Arguments

| | |
|---|---|
| `x` | descriptor instance |
| `fileName` | file name to be stripped of its extension |

## Value

the correct header name

---

`dataFileMatches` *Generic function to test whether a file name conforms to the given a* `BrainFileDescriptor` *instance. Will test for match to data file only*

---

## Description

Generic function to test whether a file name conforms to the given a `BrainFileDescriptor` instance. Will test for match to data file only

## Usage

```
dataFileMatches(x, fileName)

## S4 method for signature 'BrainFileDescriptor,character'
dataFileMatches(x, fileName)
```

## Arguments

| | |
|---|---|
| `x` | object for which the file name is to matched to |
| `fileName` | file name to be matched |

## Value

TRUE for match, FALSE otherwise

---

`dataReader` *Generic function to create data reader*

---

## Description

Generic function to create data reader

## Usage

```
dataReader(x, offset)

## S4 method for signature 'NIfTIMetaInfo'
dataReader(x, offset = 0)

## S4 method for signature 'AFNIMetaInfo'
dataReader(x, offset = 0)

## S4 method for signature 'NIMLSurfaceDataMetaInfo'
dataReader(x)
```

## Arguments

| | |
|---|---|
| `x` | an object specifying the infromation required to produce the reader |
| `offset` | the byte offset (number of bytes to skip before reading) |

---

`DenseBrainVector-class` *DenseBrainVector*

---

## Description

Four-dimensional brain image, backed by an array

constructor function for class `DenseBrainVector`

## Usage

```
DenseBrainVector(data, space, source = NULL, label = "")
```

### Arguments

| | |
|---|---|
| data | a 4-dimensional array or a 2-dimension matrix that is either nvoxels by ntime-points or ntime-points by nvoxels |
| space | a BrainSpace object |
| source | an optional BrainSource object |
| label | a label of type character |

### Value

DenseBrainVector instance

---

## DenseBrainVolume-class *DenseBrainVolume*

---

### Description

Three-dimensional brain image, backed by an array

Construct a DenseBrainVolume instance

### Usage

```
DenseBrainVolume(data, space, source = NULL, label = "", indices = NULL)
```

### Arguments

| | |
|---|---|
| data | a three-dimensional array |
| space | an instance of class BrainSpace |
| source | an instance of class BrainSource |
| label | a character string |
| indices | an optional 1-d index vector |

### Value

DenseBrainVolume instance

## dim,BrainData-method *dim of* BrainData *object*

### Description

dim of BrainData object

### Usage

```
## S4 method for signature 'BrainData'
dim(x)
```

### Arguments

| | |
|---|---|
| x | the object |

## dim,BrainSpace-method *dim*

### Description

dim

### Usage

```
## S4 method for signature 'BrainSpace'
dim(x)
```

### Arguments

| | |
|---|---|
| x | the object |

## dim,FileMetaInfo-method *dim of* FileMetaInfo

### Description

dim of FileMetaInfo

### Usage

```
## S4 method for signature 'FileMetaInfo'
dim(x)
```

### Arguments

| | |
|---|---|
| x | the object |

## dropDim *Generic function to drop a dimension from an object*

### Description

Generic function to drop a dimension from an object

### Usage

```
dropDim(x, dimnum)

## S4 method for signature 'AxisSet2D,numeric'
dropDim(x, dimnum)

## S4 method for signature 'AxisSet2D,missing'
dropDim(x, dimnum)

## S4 method for signature 'AxisSet3D,numeric'
dropDim(x, dimnum)

## S4 method for signature 'AxisSet3D,missing'
dropDim(x, dimnum)

## S4 method for signature 'BrainSpace,numeric'
dropDim(x, dimnum)

## S4 method for signature 'BrainSpace,missing'
dropDim(x)
```

### Arguments

| | |
|---|---|
| x | a dimensioned object |
| dimnum | the index of the dimension to drop |

### Examples

```
x = BrainSpace(c(10,10,10), c(1,1,1))
x1 <- dropDim(x)
ndim(x1) == 2
dim(x1)[2] == 10
```

---

## eachSeries *Generic functions to apply a function to each series of a 4D image That is, if the 4th dimension is 'time' each series is a 1D time series.*

---

### Description

Generic functions to apply a function to each series of a 4D image That is, if the 4th dimension is 'time' each series is a 1D time series.

### Usage

```
eachSeries(x, FUN, withIndex, ...)

## S4 method for signature 'SparseBrainVector,`function`,logical'
eachSeries(x, FUN,
  withIndex = FALSE, ...)

## S4 method for signature 'BrainVector,`function`,missing'
eachSeries(x, FUN,
  withIndex = FALSE, ...)
```

### Arguments

| | |
|---|---|
| x | a four dimensional image |
| FUN | a function taking one or two arguments (depending on the value of withIndex |
| withIndex | whether the index of the series is supplied as the second argument to the function |
| ... | additional arguments |

### Details

when x is a SparseBrainVector eachSeries only iterates over nonzero series.

### Examples

```
bvec <- BrainVector(array(rnorm(24*24*24*24), c(24,24,24,24)), BrainSpace(c(24,24,24,24), c(1,1,1)))
res <- eachSeries(bvec, mean)
length(res) == 24*24*24
```

## eachSlice — *Generic functions to apply a function to each (2D) slice of an image*

### Description

Generic functions to apply a function to each (2D) slice of an image

### Usage

```
eachSlice(x, FUN, withIndex, ...)

## S4 method for signature 'BrainVolume,`function`,missing'
eachSlice(x, FUN)

## S4 method for signature 'BrainVolume,`function`,logical'
eachSlice(x, FUN, withIndex)
```

### Arguments

| | |
|---|---|
| x | the object |
| FUN | a function taking one or two arguments (depending on the value of withIndex |
| withIndex | whether the index of the slice is supplied as the second argument to the function |
| ... | additional arguments |

## eachVolume — *Generic function to apply a function to each volume of a four-dimensional image*

### Description

Generic function to apply a function to each volume of a four-dimensional image

### Usage

```
eachVolume(x, FUN, withIndex, mask, ...)

## S4 method for signature 'SparseBrainVector,`function`,logical,missing'
eachVolume(x, FUN,
  withIndex = FALSE, mask, ...)

## S4 method for signature 'SparseBrainVector,`function`,missing,missing'
eachVolume(x, FUN,
  withIndex, mask, ...)
```

```
  ## S4 method for signature
## 'SparseBrainVector,`function`,missing,LogicalBrainVolume'
eachVolume(x,
  FUN, withIndex, mask, ...)

## S4 method for signature 'BrainVector,`function`,missing,missing'
eachVolume(x, FUN, withIndex,
  mask, ...)

## S4 method for signature 'BrainVector,`function`,missing,BrainVolume'
eachVolume(x, FUN,
  withIndex, mask, ...)

## S4 method for signature 'BrainVector,`function`,missing,missing'
eachVolume(x, FUN, withIndex,
  mask, ...)

## S4 method for signature 'BrainBucket,`function`,missing,missing'
eachVolume(x, FUN, withIndex,
  mask, ...)

## S4 method for signature 'BrainBucket,`function`,logical,ANY'
eachVolume(x, FUN, withIndex,
  mask, ...)

## S4 method for signature 'BrainVector,`function`,logical,ANY'
eachVolume(x, FUN, withIndex,
  mask, ...)
```

## Arguments

| | |
|---|---|
| x | four-dimensional image, e.g. of class BrainVector |
| FUN | a function taking one or two arguments (depending on the value of withIndex) |
| withIndex | whether the index of the volume supplied as the second argument to the function |
| mask | an image mask indicating subset of volume elements to apply function over |
| ... | additional arguments |

## Value

a list of results of apply FUN to each volume.

## Examples

```
bvec <- BrainVector(array(rnorm(24*24*24*24), c(24,24,24,24)), BrainSpace(c(24,24,24,24), c(1,1,1)))
res <- eachVolume(bvec, mean)

res <- eachVolume(bvec, function(x,i) median(x), withIndex=TRUE)
```

`fileMatches` *Generic function to test whether a file name conforms to the given* `BrainFileDescriptor` *instance. Will test for match to either header file or data file*

## Description

Generic function to test whether a file name conforms to the given `BrainFileDescriptor` instance. Will test for match to either header file or data file

## Usage

```
fileMatches(x, fileName)

## S4 method for signature 'BrainFileDescriptor,character'
fileMatches(x, fileName)
```

## Arguments

| | |
|---|---|
| `x` | object for which the file name is to matched to |
| `fileName` | file name to be matched |

## Value

TRUE for match, FALSE otherwise

`FileMetaInfo-class` *FileMetaInfo*

## Description

This class contains meta information from an image data file

This class contains meta information for a NIfTI image file

This class contains meta information for a AFNI image file

## Slots

`headerFile` name of the file containing meta information

`dataFile` name of the file containing data

`fileDescriptor` descriptor of image file format

`endian` byte order of data ('little' or 'big')

`dataOffset` the number of bytes preceding the start of image data in data file

`bytesPerElement` number of bytes per element

`intercept` constant value added to image – multiple values allowed (must equal numer of sub-images)

`slope` image multiplier – multiple values allowed (must equal numer of sub-images)

`header` a list of format specific attributes

`nifti_header` a `list` of attributes specific to the NIfTI file format

`afni_header` a list of attributes specific to the AFNI file format

`afni_header` a `list` of attributes specific to the AFNI file format

---

`fill` *Generic function to map values from one set to another using a user-supplied lookup table*

---

## Description

Generic function to map values from one set to another using a user-supplied lookup table

## Usage

```
fill(x, lookup)

## S4 method for signature 'BrainVolume,list'
fill(x, lookup)

## S4 method for signature 'BrainVolume,matrix'
fill(x, lookup)
```

## Arguments

| | |
|---|---|
| `x` | the object to map values from |
| `lookup` | the lookup table. The first column is the "key" the second column is the "value". |

## Value

a new object where the original values have been filled in with the values in the lookup table

## Examples

```
x <- BrainSpace(c(10,10,10), c(1,1,1))
vol <- BrainVolume(sample(1:10, 10*10*10, replace=TRUE), x)

## lookup table is list
lookup <- lapply(1:10, function(i) i*10)
ovol <- fill(vol, lookup)

## lookup table is matrix. First column is key, second column is value
names(lookup) <- 1:length(lookup)
lookup.mat <- cbind(as.numeric(names(lookup)), unlist(lookup))
```

```
ovol2 <- fill(vol, lookup.mat)
all.equal(as.vector(ovol2), as.vector(ovol))
```

---

`FreesurferAsciiSurfaceFileDescriptor-class` *FresurferAsciiSurfaceFileDescriptor*

---

## Description

This class supports the FreesurferAsciiSurfaceFileDescriptor file format for surface geometry

---

`FreesurferSurfaceGeometryMetaInfo-class` *FreeSurferSurfaceGeometryMetaInfo This class contains meta information for brain surface geometry*

---

## Description

FreeSurferSurfaceGeometryMetaInfo

This class contains meta information for brain surface geometry

---

`gridToCoord` *Generic function to convert N-dimensional grid coordinate coordinates to real world coordinates Generic function to convert N-dimensional grid coordinates to real world coordinates*

---

## Description

Generic function to convert N-dimensional grid coordinate coordinates to real world coordinates Generic function to convert N-dimensional grid coordinates to real world coordinates

## Usage

```
gridToCoord(x, coords)

## S4 method for signature 'BrainSpace,matrix'
gridToCoord(x, coords)

## S4 method for signature 'BrainVolume,matrix'
gridToCoord(x, coords)
```

### Arguments

| | |
|---|---|
| x | the object |
| coords | a matrix of grid coordinates |

### Value

a matrix of real coordinates

---

## gridToIndex *Generic function to convert N-dimensional grid coordinate to 1D indices*

---

### Description

Generic function to convert N-dimensional grid coordinate to 1D indices

### Usage

```
gridToIndex(x, coords)

## S4 method for signature 'BrainSlice,matrix'
gridToIndex(x, coords)

## S4 method for signature 'BrainSpace,matrix'
gridToIndex(x, coords)

## S4 method for signature 'BrainSpace,numeric'
gridToIndex(x, coords)

## S4 method for signature 'BrainVolume,matrix'
gridToIndex(x, coords)

## S4 method for signature 'BrainVolume,numeric'
gridToIndex(x, coords)
```

### Arguments

| | |
|---|---|
| x | the object, typically a BrainVolume or BrainSpace instance. |
| coords | a matrix where each row is a coordinate or a vector of length equal to ndim(x) |

### Value

a vector of indices

---

`headerFile` *Generic function to get the name of the header file, given a file name and a* `BrainFileDescriptor` *instance.*

---

## Description

Generic function to get the name of the header file, given a file name and a `BrainFileDescriptor` instance.

## Usage

```
headerFile(x, fileName)

## S4 method for signature 'BrainFileDescriptor,character'
headerFile(x, fileName)
```

## Arguments

| | |
|---|---|
| `x` | descriptor instance |
| `fileName` | file name to be stripped of its extension |

## Value

the correct header name

---

`headerFileMatches` *Generic function to test whether a file name conforms to the given* `BrainFileDescriptor` *instance. Will test for match to header file only*

---

## Description

Generic function to test whether a file name conforms to the given `BrainFileDescriptor` instance. Will test for match to header file only

## Usage

```
headerFileMatches(x, fileName)

## S4 method for signature 'BrainFileDescriptor,character'
headerFileMatches(x, fileName)
```

## Arguments

| | |
|---|---|
| `x` | object for which the file name is to matched to |
| `fileName` | file name to be matched |

## Value

TRUE for match, FALSE otherwise

---

`image,BrainVolume-method`
*image*

---

## Description

image

image

## Usage

```
## S4 method for signature 'BrainVolume'
image(x, slice, col = gray((0:255)/255, alpha = 1),
  zero.col = "#000000", axis = 3, ...)

## S4 method for signature 'Overlay'
image(x, zpos, axis = 3)

## S4 method for signature 'Layer'
image(x, zpos, axis = 3)
```

## Arguments

| | |
|---|---|
| `x` | the object to display |
| `slice` | the voxel index of the slice to display |
| `col` | a color map |
| `zero.col` | the color to use when the value is 0 (e.g background color) |
| `axis` | the axis index |
| `...` | extra arguments to passed to `grid.raster` |
| `zpos` | the z coordinate |

imageGrid *imageGrid*

## Description

Display a set of images slices in a 2D montage

## Usage

```
imageGrid(layer, gridDim = c(3, 3), zstart, zend, panelSize = 3,
  panelUnit = "inches", interpolate = FALSE, fontCol = "red")
```

## Arguments

| | |
|---|---|
| layer | the layer to display |
| gridDim | the dimensions of the 2D grid montage |
| zstart | the z coordinate of the first slice |
| zend | the z coordinate of the last slice |
| panelSize | the size of each panel in the montage (default unit is inches) |
| panelUnit | the unit for the panel size (default is inches) |
| interpolate | whether to interpolate pixel values |
| fontCol | color of labels indicating slice level |

IndexLookupVolume-class *IndexLookupVolume*

## Description

Three-dimensional brain image that can be used as a map between 1D grid indices and a table of values Currently used in the SparseBrainVector class.

IndexLookupVolume

## Usage

```
IndexLookupVolume(space, indices)
```

## Arguments

| | |
|---|---|
| space | a BrainSpace object |
| indices | the set of 1-d indices defining the lookup map |

indexToCoord *Generic function to convert 1D indices to N-dimensional real world coordinates*

## Description

Generic function to convert 1D indices to N-dimensional real world coordinates

## Usage

```
indexToCoord(x, idx)

## S4 method for signature 'BrainSpace,index'
indexToCoord(x, idx)

## S4 method for signature 'BrainVolume,index'
indexToCoord(x, idx)
```

## Arguments

| | |
|---|---|
| x | the object |
| idx | the 1D indices |

## Value

a matrix of real coordinates

## Examples

```
bvol <- BrainVolume(array(0, c(10,10,10)), BrainSpace(c(10,10,10), c(1,1,1)))
idx <- 1:10
g <- indexToCoord(bvol, idx)
idx2 <- coordToIndex(bvol, g)
all.equal(idx, idx2)
```

indexToGrid *Generic function to convert 1D indices to N-dimensional grid coordinates*

## Description

Generic function to convert 1D indices to N-dimensional grid coordinates

### Usage

```
indexToGrid(x, idx)

## S4 method for signature 'BrainSlice,index'
indexToGrid(x, idx)

## S4 method for signature 'BrainSpace,index'
indexToGrid(x, idx)

## S4 method for signature 'BrainVector,index'
indexToGrid(x, idx)

## S4 method for signature 'BrainVolume,index'
indexToGrid(x, idx)
```

### Arguments

| | |
|---|---|
| x | the object |
| idx | the 1D vector of indices |

### Value

a matrix of grid coordinates

### Examples

```
bvol <- BrainVolume(array(0, c(10,10,10)), BrainSpace(c(10,10,10), c(1,1,1)))
idx <- 1:10
g <- indexToGrid(bvol, idx)
bvol[g]
```

---

## indices *Extract indices*

---

### Description

Extract indices

indices

### Usage

```
indices(x)

## S4 method for signature 'ROIVolume'
indices(x)
```

```
## S4 method for signature 'SparseBrainVector'
indices(x)

## S4 method for signature 'IndexLookupVolume'
indices(x)
```

## Arguments

| | |
|---|---|
| x | the object to extract indices |

---

`inverseTrans` *Generic getter to extract inverse image coordinate transformation*

---

## Description

Generic getter to extract inverse image coordinate transformation

## Usage

```
inverseTrans(x)

## S4 method for signature 'BrainSpace'
inverseTrans(x)

## S4 method for signature 'BrainData'
inverseTrans(x)
```

## Arguments

| | |
|---|---|
| x | an object |

## Examples

```
bspace <- BrainSpace(c(10,10,10), c(2,2,2))
itrans <- inverseTrans(bspace)
identical(trans(bspace) %*% inverseTrans(bspace), diag(4))
```

## Kernel *Create a Kernel object*

### Description

Create a Kernel object

### Usage

```
Kernel(kerndim, vdim, FUN = dnorm, ...)
```

### Arguments

| | |
|---|---|
| kerndim | the dimensions in voxels of the kernel |
| vdim | the dimensions of the voxels in real units |
| FUN | the kernel function taking as its first argument representing the distance from the center of the kernel |
| ... | additional parameters to the kernel FUN |

## Kernel-class *Kernel*

### Description

A class representing an image kernel

### Slots

`width` the width in voxels of the kernel

`weights` the kernel weights

`voxels` the relative voxel coordinates of the kernel

`coords` the relative real coordinates of the kernel

## Layer *Layer*

### Description

create a Layer object

### Usage

```
Layer(vol, colorMap = gray((0:255)/255, alpha = 1), thresh = c(0, 0),
  axis = 3, zero.col = "#000000", alpha = 1)
```

### Arguments

| | |
|---|---|
| vol | volume instance of BrainVolume |
| colorMap | a lookup table defining mapping from image intensity values to colors. |
| thresh | a range (min,max) defining the threshold window for determining image opacity. |
| axis | the axis index of the axis perpendicular to the xy plane (options: 1,2,3; default is 3) |
| zero.col | the color used when the value is zero. |
| alpha | transparency multiplier, vlaue between 0 and 1. |

### Value

an object of class Layer

## Layer-class *Layer*

### Description

A class used for displaying 2D images with color maps

### Slots

vol the BrainVolume that provides the data for the layer.

colorMap a character vector of colors in hexadecimal rgb format. Can be generated by calls to rainbow, heat.colors, topo.colors, terrain.colors or similar functions.

thresh cut-off value above which vlaues will be made transparent.

axis the axis index of perpendicular to the xy plane (option: 1,2,3; default is 3)

zero.col the color pixels with intensity of zero. This value overrides the color from the slot colorMap

alpha the transparency of the layer

`length,ROIVolume-method`

*Get length of* `BrainVector`. *This is the numbe rof volumes in the volume vector (e.g. the 4th image dimension)*

## Description

Get length of `BrainVector`. This is the numbe rof volumes in the volume vector (e.g. the 4th image dimension)

## Usage

```
## S4 method for signature 'ROIVolume'
length(x)

## S4 method for signature 'BrainVector'
length(x)
```

## Arguments

| | |
|---|---|
| `x` | the object to get `length` |

`loadBucket` *loadBucket*

## Description

load a BrainBucket object from file

## Usage

```
loadBucket(fileName, pattern = NULL, indices = NULL)
```

## Arguments

| | |
|---|---|
| `fileName` | the name of the file to load |
| `pattern` | optional regular expression used to filter the sub-volumes using associated labels |
| `indices` | optional set of sub-volume indices to load |

`loadData` *Generic function to load data from a data source*

## Description

Generic function to load data from a data source

load a BrainSurface

loadData

Load data from a `BrainBucketSource`

load a BrainVolume

## Usage

```
loadData(x, ...)

## S4 method for signature 'BrainSurfaceSource'
loadData(x)

## S4 method for signature 'SparseBrainVectorSource'
loadData(x)

## S4 method for signature 'BrainVectorSource'
loadData(x, mmap = FALSE)

## S4 method for signature 'BrainBucketSource'
loadData(x, key)

## S4 method for signature 'BrainVolumeSource'
loadData(x)
```

## Arguments

| | |
|---|---|
| `x` | a data source |
| `...` | additional arguments |
| `mmap` | use memory-mapped file |
| `key` | the name or index of the bucket to load |

## Value

an instance of class `BrainVector`

an instance of class `BrainVolume`

## loadFSSurface *load Freesurfer ascii surface*

### Description

load Freesurfer ascii surface

### Usage

```
loadFSSurface(mesh)
```

### Arguments

| | |
|---|---|
| mesh | file name of mesh to read in. |

## loadSurface *load an surface from a set of files*

### Description

load an surface from a set of files

### Usage

```
loadSurface(surfaceName, surfaceDataName)
```

### Arguments

| | |
|---|---|
| surfaceName | the name of the file containing the surface geometry. |
| surfaceDataName | the name of the file containing the values to be mapped to the surface. |

### Value

an instance of the class BrainSurface

## loadVector *loadVector*

### Description

load an image volume from a file

### Usage

```
loadVector(fileName, indices = NULL, mask = NULL)
```

### Arguments

| | |
|---|---|
| fileName | the name of the file to load |
| indices | the indices of the sub-volumes to load (e.g. if the file is 4-dimensional) |
| mask | a mask defining the spatial elements to load |

### Value

an BrainVector object

## loadVolume *Load an image volume from a file*

### Description

Load an image volume from a file

### Usage

```
loadVolume(fileName, index = 1)
```

### Arguments

| | |
|---|---|
| fileName | the name of the file to load |
| index | the index of the volume (e.g. if the file is 4-dimensional) |

### Value

an instance of the class DenseBrainVolume

### Examples

```
fname <- system.file("extdata", "global_mask.nii", package="neuroim")
x <- loadVolume(fname)
print(dim(x))
space(x)
```

## loadVolumeList *loadVolList*

### Description

load a list of image volumes and return a BrainVector instance

### Usage

```
loadVolumeList(fileNames, mask = NULL)
```

### Arguments

| | |
|---|---|
| fileNames | a list of files to load |
| mask | an optional mask indicating subset of voxels to load |

### Value

an instance of class BrainVector

## LogicalBrainVolume-class *LogicalBrainVolume*

### Description

Three-dimensional brain image where all values are either TRUE or FALSE

Construct a LogicalBrainVolume instance

### Usage

```
LogicalBrainVolume(data, space, source = NULL, label = "", indices = NULL)
```

### Arguments

| | |
|---|---|
| data | a three-dimensional array, a 1D vector with length equal to prod(dim(space)), or a set of indices where elements are TRUE |
| space | an instance of class BrainSpace |
| source | an instance of class BrainSource |
| label | a character string |
| indices | an optional 1-d index vector |

### Value

LogicalBrainVolume instance

## lookup *Index Lookup operation*

### Description

Index Lookup operation

lookup

### Usage

```
lookup(x, i, ...)

## S4 method for signature 'SparseBrainVector,numeric'
lookup(x, i)

## S4 method for signature 'IndexLookupVolume,numeric'
lookup(x, i)
```

### Arguments

| | |
|---|---|
| x | the object to query |
| i | the index to lookup |
| ... | additional arguments |

## makeVector *makeVector*

### Description

Construct a BrainVector instance, using default (dense) implementation

### Usage

```
makeVector(data, refdata, source = NULL, label = "")
```

### Arguments

| | |
|---|---|
| data | a four-dimensional array |
| refdata | an instance of class BrainVector or BrainVolume containing the reference space for the new vector. |
| source | an instance of class BrainSource |
| label | a character string |

### Value

DenseBrainVector instance

makeVolume *makeVolume*

## Description

Construct a BrainVolume instance, using default (dense) implementation

## Usage

```
makeVolume(data = NULL, refvol, source = NULL, label = "",
  indices = NULL)
```

## Arguments

| | |
|---|---|
| data | an optional one- or three-dimensional vector or array |
| refvol | an instance of class BrainVolume containing the reference space for the new volume. |
| source | an optional instance of class BrainSource |
| label | an optional character string |
| indices | an optional 1d vector of indices in to the 3d space |

## Value

DenseBrainVolume instance

## Examples

```
bspace <- BrainSpace(c(64,64,64), spacing=c(1,1,1))
dat <- array(rnorm(64*64*64), c(64,64,64))
bvol <- BrainVolume(dat,bspace, label="test")
bvol2 <- makeVolume(dat, bvol)
all.equal(as.array(bvol),as.array(bvol2))
data <- 1:10
indices = seq(1,1000, length.out=10)
bvol3 <- makeVolume(data,bvol,indices=indices)
sum(bvol3) == sum(data)
```

## map *Generic function to apply a function to an object*

### Description

Generic function to apply a function to an object

apply a kernel function to a BrainVolume

### Usage

```
map(x, m, ...)

## S4 method for signature 'BrainVolume,Kernel'
map(x, m, mask = NULL)
```

### Arguments

| | |
|---|---|
| x | the object that is mapped |
| m | the mapping object |
| ... | additional arguments |
| mask | restrict application of kernel to maksed area |

## mapToColors *mapToColors*

### Description

map an matrix of intensity values to a matrix of color values.

### Usage

```
mapToColors(imslice, col = heat.colors(128, alpha = 1),
  zero.col = "#00000000", alpha = 1)
```

### Arguments

| | |
|---|---|
| imslice | an image matrix defining intensity values |
| col | a color map |
| zero.col | the background color. |
| alpha | transparency multiplier |

## matchAnatomy2D *given two named axes return AxisSet2D singleton*

### Description

given two named axes return AxisSet2D singleton

### Usage

```
matchAnatomy2D(axis1, axis2)
```

### Arguments

| | |
|---|---|
| axis1 | the first axis |
| axis2 | the second axis |

## matchAnatomy3D *given three named axes return AxisSet3D singleton*

### Description

given three named axes return AxisSet3D singleton

### Usage

```
matchAnatomy3D(axis1, axis2, axis3)
```

### Arguments

| | |
|---|---|
| axis1 | the first axis |
| axis2 | the second axis |
| axis3 | the third axis |

## matrixToVolumeList — *matrixToVolumeList converts a matrix to a list of BrainVolumes with values filled at grid coordinates determined by the* vox *argument.*

### Description

matrixToVolumeList converts a matrix to a list of BrainVolumes with values filled at grid coordinates determined by the vox argument.

### Usage

```
matrixToVolumeList(voxmat, mat, mask, default = NA)
```

### Arguments

| | |
|---|---|
| voxmat | an N by 3 matrix of voxel coordinates |
| mat | an N by M matrix of values where M is the number of volumes to create (e.g. one volume per column in mat) |
| mask | a reference volume defining the geometry of the output volumes. This can either be of type BrainSpace or BrainVolume |
| default | the value that will be used for voxels not contained within voxmat (defualt is NA) |

### Value

a list of BrainVolume instances, one for each column of mat

## mergePartitions — *mergePartitions*

### Description

mergePartitions

merge partititons in a ClusteredBrainVolume

### Usage

```
mergePartitions(x, K, features, ...)

## S4 method for signature 'ClusteredBrainVolume,numeric,matrix'
mergePartitions(x, K, features)
```

### Arguments

| | |
|---|---|
| x | the object to merge |
| K | the number of merged partitions |
| features | the features used to define the partition |
| ... | additional arguments |

---

## MNI_SPACE_1MM *MNI SPACE 1MM*

---

### Description

This is a BrainSpace object encoding the geometry of the MNI_1MM template.

### Usage

```
data(MNI_SPACE_1MM)
```

### Format

a BrainSpace instance

---

## NamedAxis-class *NamedAxis*

---

### Description

This class represents an axis with a name attribute

### Slots

axis the name of the axis

direction of axis (-1,+1)

## names,BrainBucketSource-method *names*

### Description

names

### Usage

```
## S4 method for signature 'BrainBucketSource'
names(x)

## S4 method for signature 'BrainBucket'
names(x)
```

### Arguments

| | |
|---|---|
| x | the object to get names of |

## ndim *Generic function to extract the number of dimensions of an object*

### Description

Generic function to extract the number of dimensions of an object

### Usage

```
ndim(x, ...)

## S4 method for signature 'AxisSet'
ndim(x, ...)

## S4 method for signature 'BrainData'
ndim(x)

## S4 method for signature 'BrainSpace'
ndim(x)
```

### Arguments

| | |
|---|---|
| x | n-dimensional object |
| ... | additional arguments |

## Examples

```
x = BrainSpace(c(10,10,10), c(1,1,1))
ndim(x) == 3
x = BrainSpace(c(10,10,10,3), c(1,1,1,1))
ndim(x) == 4
```

---

`neuroim` *neuroim*

---

## Description

Data structures for analysis of neuroimaging data.

## Details

none

---

`NIfTIFileDescriptor-class` *NIfTIFileDescriptor*

---

## Description

This class supports the NIfTI file format

---

`NIfTIMetaInfo` *Constructor for* `NIfTIMetaInfo` *class*

---

## Description

Constructor for `NIfTIMetaInfo` class

## Usage

```
NIfTIMetaInfo(descriptor, nifti_header)
```

## Arguments

| | |
|---|---|
| `descriptor` | an instance of class `NIfTIFileDescriptor` |
| `nifti_header` | a `list` returned by `readNIftiHeader` |

## Value

an instance of class `NIfTIMetaInfo`

---

`NIMLSurfaceDataMetaInfo` *Constructor for* `NIMLSurfaceDataMetaInfo` *class*

---

## Description

Constructor for `NIMLSurfaceDataMetaInfo` class

## Usage

```
NIMLSurfaceDataMetaInfo(descriptor, header)
```

## Arguments

| | |
|---|---|
| `descriptor` | the file descriptor |
| `header` | a `list` containing header information |

---

`NIMLSurfaceDataMetaInfo-class` *NIMLSurfaceDataMetaInfo This class contains meta information for surface-based data for the NIML data format*

---

## Description

NIMLSurfaceDataMetaInfo

This class contains meta information for surface-based data for the NIML data format

## Slots

`data` the numeric data matrix of surface values (rows = nodes, columns=surface vectors)

`nodeIndices` the indices of the nodes for mapping to associated surface geometry.

---

`NIMLSurfaceFileDescriptor-class` *NIMLSurfaceFileDescriptor*

---

## Description

This class supports the NIML file format for surface-based data

## NullMetaInfo-class *NullMetaInfo*

### Description

This is class is used to denote the absense of meta information

## numClusters *numClusters*

### Description

numClusters

get number of clusters in a ClusteredBrainVolume

### Usage

```
numClusters(x)

## S4 method for signature 'ClusteredBrainVolume'
numClusters(x)
```

### Arguments

| | |
|---|---|
| x | the object to extract number of clusters |

## origin *Generic getter to extract image origin*

### Description

Generic getter to extract image origin

### Usage

```
origin(x)

## S4 method for signature 'BrainSpace'
origin(x)

## S4 method for signature 'BrainData'
origin(x)
```

## Arguments

| | |
|---|---|
| x | an object with an origin |

## Examples

```
bspace <- BrainSpace(c(10,10,10), c(2,2,2))
origin(bspace)
```

---

`overlay` *overlay two objects*

---

## Description

overlay two objects

overlay

## Usage

```
overlay(x, y, ...)

## S4 method for signature 'Layer,Layer'
overlay(x, y)

## S4 method for signature 'Overlay,Layer'
e1 + e2

## S4 method for signature 'Layer,Layer'
e1 + e2
```

## Arguments

| | |
|---|---|
| x | the underlay object |
| y | the overlay object |
| ... | additional arguments for class-specific implementations |
| e1 | the left operand |
| e2 | the right operand |

partition *partition*

## Description

partition

partition a ClusteredBrainVolume into K spatial disjoint components for every existing partition in the volume

## Usage

```
partition(x, K, features, ...)

## S4 method for signature 'ClusteredBrainVolume,numeric,matrix'
partition(x, K, features,
  method = "kmeans")
```

## Arguments

| | |
|---|---|
| x | the object to partition |
| K | the number of partitions |
| features | the features used to define the partition |
| ... | additional arguments |
| method | clustering method |

permMat *Extract permutation matrix*

## Description

Extract permutation matrix

permMat

## Usage

```
permMat(x, ...)

## S4 method for signature 'AxisSet2D'
permMat(x, ...)
```

## Arguments

| | |
|---|---|
| x | the object |
| ... | additional arguments |

## pick *pick*

### Description

pick

### Usage

```
pick(x, mask, ...)
```

### Arguments

| | |
|---|---|
| x | the object to pick from |
| mask | a mask object |
| ... | addiitonal arguments |

## print *Generic function to print an object*

### Description

Generic function to print an object

### Usage

```
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | the object to print |
| ... | additional arguments |

---

`print,AxisSet2D-method`
*print a* `AxisSet2D` *instance*

---

## Description

print a `AxisSet2D` instance

## Usage

```
## S4 method for signature 'AxisSet2D'
print(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | the object |
| `...` | extra args |

---

`print,AxisSet3D-method`
*print a* `AxisSet3D` *instance*

---

## Description

print a `AxisSet3D` instance

## Usage

```
## S4 method for signature 'AxisSet3D'
print(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | the object |
| `...` | extra args |

## print,NamedAxis-method *print a* NamedAxis

### Description

print a NamedAxis

### Usage

```
## S4 method for signature 'NamedAxis'
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | the object |
| ... | extra arguments |

## RandomSearchlight *Create an Random Searchlight iterator*

### Description

Create an Random Searchlight iterator

### Usage

```
RandomSearchlight(mask, radius)
```

### Arguments

| | |
|---|---|
| mask | an image volume containing valid central voxels for roving searchlight |
| radius | in mm of spherical searchlight |

readAFNIHeader *readAFNIHeader*

## Description

readAFNIHeader

## Usage

```
readAFNIHeader(fileName)
```

## Arguments

| | |
|---|---|
| fileName | the name of the AFNI header file (ending in .HEAD) |

readColumns *Generic function to read a set of column vector from an input source (e.g.* ColumnReader*)*

## Description

Generic function to read a set of column vector from an input source (e.g. ColumnReader)

## Usage

```
readColumns(x, columnIndices)

## S4 method for signature 'ColumnReader,numeric'
readColumns(x, columnIndices)
```

## Arguments

| | |
|---|---|
| x | the input channel |
| columnIndices | the column indices |

## Value

a matrix consisting of the requested column vectors

## readElements *Generic function to read a sequence of elements from an input source*

### Description

Generic function to read a sequence of elements from an input source

readElements

### Usage

```
readElements(x, numElements)

## S4 method for signature 'BinaryReader,numeric'
readElements(x, numElements)
```

### Arguments

| | |
|---|---|
| x | the input channel |
| numElements | the number of elements to read |

### Value

the elements as a vector

## readHeader *read header information of an image file*

### Description

read header information of an image file

### Usage

```
readHeader(fileName)
```

### Arguments

| | |
|---|---|
| fileName | the name of the file to read |

### Value

an instance of class FileMetaInfo

## readMetaInfo *Generic function to read image meta info given a file and a* BrainFileDescriptor *instance.*

### Description

Generic function to read image meta info given a file and a BrainFileDescriptor instance.

### Usage

```
readMetaInfo(x, fileName)

## S4 method for signature 'NIfTIFileDescriptor'
readMetaInfo(x, fileName)

## S4 method for signature 'AFNIFileDescriptor'
readMetaInfo(x, fileName)

## S4 method for signature 'NIMLSurfaceFileDescriptor'
readMetaInfo(x, fileName)

## S4 method for signature 'FreesurferAsciiSurfaceFileDescriptor'
readMetaInfo(x, fileName)
```

### Arguments

| | |
|---|---|
| x | descriptor instance |
| fileName | file name contianing meta information |

## RegionCube *Create A Cuboid Region of Interest*

### Description

Create A Cuboid Region of Interest

### Usage

```
RegionCube(bvol, centroid, surround, fill = NULL, nonzero = FALSE)
```

### Arguments

| | |
|---|---|
| bvol | an BrainVolume or BrainSpace instance |
| centroid | the center of the cube in *voxel* coordinates |
| surround | the number of voxels on either side of the central voxel. A vector of length 3. |
| fill | optional value(s) to assign to data slot. |
| nonzero | keep only nonzero elements from bvol. If bvol is A BrainSpace then this argument is ignored. |

### Value

an instance of class ROIVolume

### Examples

```
sp1 <- BrainSpace(c(10,10,10), c(1,1,1))
cube <- RegionCube(sp1, c(5,5,5), 3)
vox <- coords(cube)
cube2 <- RegionCube(sp1, c(5,5,5), 3, fill=5)
```

---

## RegionSphere *Create A Spherical Region of Interest*

---

### Description

Create A Spherical Region of Interest

### Usage

```
RegionSphere(bvol, centroid, radius, fill = NULL, nonzero = FALSE)
```

### Arguments

| | |
|---|---|
| bvol | an BrainVolume or BrainSpace instance |
| centroid | the center of the sphere in voxel space |
| radius | the radius in real units (e.g. millimeters) of the spherical ROI |
| fill | optional value(s) to assign to data slot |
| nonzero | keep only nonzero elements from bvol |

### Value

an instance of class ROIVolume

## Examples

```
sp1 <- BrainSpace(c(10,10,10), c(1,1,1))
cube <- RegionSphere(sp1, c(5,5,5), 3.5)
vox = coords(cube)
```

---

RegionSquare — *Create a square region of interest where the z-dimension is fixed at one voxel coordinate.*

---

## Description

Create a square region of interest where the z-dimension is fixed at one voxel coordinate.

## Usage

```
RegionSquare(bvol, centroid, surround, fill = NULL, nonzero = FALSE,
  fixdim = 3)
```

## Arguments

| | |
|---|---|
| bvol | an BrainVolume or BrainSpace instance. |
| centroid | the center of the cube in *voxel* coordinates. |
| surround | the number of voxels on either side of the central voxel. |
| fill | optional value(s) to assign to data slot. |
| nonzero | keep only nonzero elements from bvol. If bvol is A BrainSpace then this argument is ignored. |
| fixdim | the fixed dimension is the third, or z, dimension. |

## Value

an instance of class ROIVolume.

## Examples

```
sp1 <- BrainSpace(c(10,10,10), c(1,1,1))
square <- RegionSquare(sp1, c(5,5,5), 1)
vox <- coords(square)
## a 3 X 3 X 1 grid
nrow(vox) == 9
```

## render *Render an image to create a drawable image.*

### Description

Render an image to create a drawable image.

### Usage

```
render(x, width, height, colmap, ...)

## S4 method for signature 'BrainSlice,numeric,numeric,character'
render(x, width, height,
  colmap, zero.col = "#000000FF", alpha = 1, units = "mm")
```

### Arguments

| | |
|---|---|
| x | the object, e.g. an instance of type BrainSlice |
| width | width of the rendered image |
| height | height of the rendered image |
| colmap | the colors used to map from values to RGBA colors. |
| ... | additional arguments |
| zero.col | color used when background intensity is 0. |
| alpha | transparency multiplier |
| units | grid unit type, e.g. "mm", "inches" |

## renderSlice *Render a slice at z coordinate*

### Description

Render a slice at z coordinate

### Usage

```
renderSlice(x, zpos, width, height, colmap, ...)

## S4 method for signature 'Overlay,numeric,numeric,numeric,missing'
renderSlice(x, zpos, width,
  height, zero.col = "#000000FF", units = "mm")

## S4 method for signature 'Layer,numeric,numeric,numeric,missing'
renderSlice(x, zpos, width,
  height, colmap, zero.col = "#000000FF", units = "mm")
```

### Arguments

| | |
|---|---|
| x | the object, e.g. an instance of type Layer or Overlay |
| zpos | the z coordinate to slice through. |
| width | width of the rendered image |
| height | height of the rendered image |
| colmap | the colors used to map from values to RGBA colors. |
| ... | additional arguments |
| zero.col | color used when background intensity is 0. |
| units | grid unit type, e.g. "mm", "inches" |

---

## ROIVolume *Create an instance of class ROIVolume*

---

### Description

Create an instance of class ROIVolume

### Usage

```
ROIVolume(vspace, coords, data = rep(length(indices), 1))
```

### Arguments

| | |
|---|---|
| vspace | the volume BrainSpace |
| coords | matrix of voxel coordinates |
| data | the data values |

### Value

an instance of class ROIVolume

---

## ROIVolume-class *ROIVolume*

---

### Description

A class that representing a volumetric region of interest (ROI).

### Slots

data the numeric data stored in the ROI

coords the voxel coordinates of the ROI

scaleSeries *Generic functions to scale (center and/or normalize by standard deviation) each series of a 4D image That is, if the 4th dimension is 'time' each series is a 1D time series.*

## Description

Generic functions to scale (center and/or normalize by standard deviation) each series of a 4D image That is, if the 4th dimension is 'time' each series is a 1D time series.

## Usage

```
scaleSeries(x, center, scale)

## S4 method for signature 'BrainVector,logical,logical'
scaleSeries(x, center, scale)

## S4 method for signature 'BrainVector,missing,logical'
scaleSeries(x, center, scale)

## S4 method for signature 'BrainVector,missing,missing'
scaleSeries(x, center, scale)

## S4 method for signature 'BrainVector,logical,missing'
scaleSeries(x, center, scale)
```

## Arguments

| | |
|---|---|
| x | a four dimensional image |
| center | a logical value indicating whether series should be centered |
| scale | a logical value indicating whether series should be divided by standard deviation |

## Examples

```
bvec <- BrainVector(array(rnorm(24*24*24*24), c(24,24,24,24)), BrainSpace(c(24,24,24,24), c(1,1,1)))
res <- scaleSeries(bvec, TRUE, TRUE)
```

Searchlight *Create an exhaustive searchlight iterator*

## Description

Create an exhaustive searchlight iterator

### Usage

```
Searchlight(mask, radius)
```

### Arguments

| | |
|---|---|
| mask | an image volume containing valid central voxels for roving searchlight |
| radius | in mm of spherical searchlight |

---

## series *Extract vector series from object*

---

### Description

Extract vector series from object

### Usage

```
series(x, i, ...)

## S4 method for signature 'SparseBrainVector,matrix'
series(x, i)

## S4 method for signature 'SparseBrainVector,numeric'
series(x, i, j, k)

## S4 method for signature 'BrainVector,matrix'
series(x, i)

## S4 method for signature 'BrainVector,numeric'
series(x, i, j, k)
```

### Arguments

| | |
|---|---|
| x | the object |
| i | the series index |
| ... | additional arguments |
| j | index for 2nd dimension |
| k | index for 3rd dimension |

seriesIter *seriesIter*

## Description

Construct a series iterator

## Usage

```
seriesIter(x)

## S4 method for signature 'SparseBrainVector'
seriesIter(x)

## S4 method for signature 'BrainVector'
seriesIter(x)
```

## Arguments

| | |
|---|---|
| x | the object to be iterated over. This is typically an instance of class BrainVector |

## Value

an iter object from the iterators package.

## Methods (by class)

- SparseBrainVector: get a seriesIter for a SparseBrainVector instance
- BrainVector: get a series iterator for a BrainVector instance

## Examples

```
## create a BrainVector with 10X10X10X10, where the last dimension is
## by convention the fourth dimension.
bvec <- BrainVector(array(rnorm(10*10*10*10), rep(10,4)), BrainSpace(rep(10,4), c(1,1,1)))
iter <- seriesIter(bvec)

## compute mean of each series
library(foreach)
library(iterators)
foreach(i=iter, .combine=c) %do% { mean(i) }
iter <- seriesIter(bvec)

## combine all series into a matrix
foreach(i=iter, .combine=rbind) %do% { i }

## scale all series, add as columns in matrix.
foreach(i=seriesIter(bvec), .combine=cbind) %do% { scale(i) }
```

## show,AxisSet1D-method *show an* AxisSet1D

### Description

show an AxisSet1D

### Usage

```
## S4 method for signature 'AxisSet1D'
show(object)
```

### Arguments

| | |
|---|---|
| object | the object |

## show,AxisSet2D-method *show an* AxisSet2D

### Description

show an AxisSet2D

### Usage

```
## S4 method for signature 'AxisSet2D'
show(object)
```

### Arguments

| | |
|---|---|
| object | the object |

## show,AxisSet3D-method *show an* AxisSet3D

### Description

show an AxisSet3D

### Usage

```
## S4 method for signature 'AxisSet3D'
show(object)
```

### Arguments

| | |
|---|---|
| object | the object |

## show,AxisSet4D-method *show an* AxisSet4D

### Description

show an AxisSet4D

### Usage

```
## S4 method for signature 'AxisSet4D'
show(object)
```

### Arguments

| | |
|---|---|
| object | the object |

## show,BaseMetaInfo-method *show a* BaseMetaInfo

### Description

show a BaseMetaInfo

### Usage

```
## S4 method for signature 'BaseMetaInfo'
show(object)
```

### Arguments

| | |
|---|---|
| object | the object |

show,BrainSpace-method *show a* BrainSpace

## Description

show a BrainSpace

## Usage

```
## S4 method for signature 'BrainSpace'
show(object)
```

## Arguments

| | |
|---|---|
| object | the object |

show,BrainVector-method *show a* BrainVector

## Description

show a BrainVector

## Usage

```
## S4 method for signature 'BrainVector'
show(object)
```

## Arguments

| | |
|---|---|
| object | the object |

---

show,BrainVectorSource-method
*show a* BrainVectorSource

---

## Description

show a BrainVectorSource

## Usage

```
## S4 method for signature 'BrainVectorSource'
show(object)
```

## Arguments

| | |
|---|---|
| object | the object |

---

show,BrainVolume-method
*show a* BrainVolume

---

## Description

show a BrainVolume

## Usage

```
## S4 method for signature 'BrainVolume'
show(object)
```

## Arguments

| | |
|---|---|
| object | the object |

## show,FileMetaInfo-method *show a* FileMetaInfo

### Description

show a FileMetaInfo

### Usage

```
## S4 method for signature 'FileMetaInfo'
show(object)
```

### Arguments

| | |
|---|---|
| object | the object |

## show,NamedAxis-method *show an* NamedAxis

### Description

show an NamedAxis

### Usage

```
## S4 method for signature 'NamedAxis'
show(object)
```

### Arguments

| | |
|---|---|
| object | the object |

---

show,NullMetaInfo-method
*show a* NullMetaInfo

---

## Description

show a NullMetaInfo

## Usage

```
## S4 method for signature 'NullMetaInfo'
show(object)
```

## Arguments

| | |
|---|---|
| object | the object |

---

show,ROIVolume-method *show an* ROIVolime

---

## Description

show an ROIVolime

## Usage

```
## S4 method for signature 'ROIVolume'
show(object)
```

## Arguments

| | |
|---|---|
| object | the object |

---

show,SparseBrainVector-method
*show a* SparseBrainVector

---

## Description

show a SparseBrainVector

## Usage

```
## S4 method for signature 'SparseBrainVector'
show(object)
```

## Arguments

| | |
|---|---|
| object | the object |

---

show,SurfaceDataMetaInfo-method
*show an* SurfaceDataMetaInfo

---

## Description

show an SurfaceDataMetaInfo

## Usage

```
## S4 method for signature 'SurfaceDataMetaInfo'
show(object)
```

## Arguments

| | |
|---|---|
| object | the object |

## show,SurfaceGeometryMetaInfo-method *show an* SurfaceGeometryMetaInfo

### Description

show an SurfaceGeometryMetaInfo

### Usage

```
## S4 method for signature 'SurfaceGeometryMetaInfo'
show(object)
```

### Arguments

| | |
|---|---|
| object | the object |

## slice *Extract a 2D slice from an image volume*

### Description

Extract a 2D slice from an image volume

### Usage

```
slice(x, zlevel, along, orientation, ...)

## S4 method for signature 'BrainVolume,numeric,numeric,character'
slice(x, zlevel, along,
  orientation)
```

### Arguments

| | |
|---|---|
| x | the object |
| zlevel | coordinate (in voxel units) along the sliced axis |
| along | the axis along which to slice |
| orientation | the target orientation of the 2D slice |
| ... | additional arguments |

## sliceData *sliceData*

### Description

extract a 2D slice from a `BrainVolume` instance.

### Usage

```
sliceData(vol, slice, axis = 3)
```

### Arguments

| | |
|---|---|
| `vol` | an `BrainVolume` instance |
| `slice` | the integer index of the slice to cut. |
| `axis` | the axis number (1, 2, 3) defining fixed axis of the 2D slice. |

## space *Generic function to extract geometric properties of an image.*

### Description

Generic function to extract geometric properties of an image.

### Usage

```
space(x, ...)

## S4 method for signature 'BrainData'
space(x)

## S4 method for signature 'BrainSpace'
space(x)

## S4 method for signature 'IndexLookupVolume'
space(x)
```

### Arguments

| | |
|---|---|
| `x` | the object to query, e.g. an instance of `BrainVolume` or `BrainVector` |
| `...` | additional arguments |

### Value

an object representing the geometric space of the image of type `BrainSpace`

## Examples

```
x = BrainSpace(c(10,10,10), c(1,1,1))
vol <- BrainVolume(rnorm(10*10*10), x)
identical(x,space(vol))
```

---

spacing *Generic function to extract the voxel dimensions of an image*

---

## Description

Generic function to extract the voxel dimensions of an image

spacing

## Usage

```
spacing(x)

## S4 method for signature 'BrainData'
spacing(x)

## S4 method for signature 'BrainSpace'
spacing(x)
```

## Arguments

| | |
|---|---|
| x | the object |

## Value

a numeric vector

## Examples

```
bspace <- BrainSpace(c(10,10,10), c(2,2,2))
all.equal(spacing(bspace), c(2,2,2))
```

SparseBrainVector-class *SparseBrainVector*

## Description

a sparse four-dimensional brain image, backed by a `matrix`, where each column represents a vector spanning the fourth dimension (e.g. time)

constructs a SparseBrainVector object

## Usage

```
SparseBrainVector(data, space, mask, source = NULL, label = "")
```

## Arguments

| | |
|---|---|
| `data` | an array which can be a `matrix` or 4-D `array` |
| `space` | a BrainSpace instance |
| `mask` | a 3D `array` of type `logical` |
| `source` | the data source – an instance of class `BrainSource` |
| `label` | associated sub-image labels |

## Slots

`mask` the mask defining the sparse domain

`data` the matrix of series, where rows span across voxel space and columns span the fourth dimensions

`map` instance of class `IndexLookupVolume` is used to map between spatial and index/row coordinates

## Examples

```
bspace <- BrainSpace(c(10,10,10,100), c(1,1,1))
mask <- array(rnorm(10*10*10) > .5, c(10,10,10))
mat <- matrix(rnorm(sum(mask)), 100, sum(mask))
svec <- SparseBrainVector(mat, bspace,mask)
length(indices(svec)) == sum(mask)
```

## SparseBrainVectorSource-class

*SparseBrainVectorSource*

### Description

A class that is used to produce a SparseBrainVector instance

constructs a SparseBrainVectorSource object

### Usage

```
SparseBrainVectorSource(metaInfo, indices, mask)
```

### Arguments

| | |
|---|---|
| metaInfo | an object of class BrainMetaInfo |
| indices | a vector of 1D indices |
| mask | a 3D array of type logical |

### Slots

mask the subset of voxels that will be stored in memory

## SparseBrainVolume-class

*SparseBrainVolume*

### Description

Three-dimensional brain image, backed by a sparseVector for Matrix package

Construct a SparseBrainVolume instance

### Usage

```
SparseBrainVolume(data, space, indices = NULL, source = NULL, label = "")
```

### Arguments

| | |
|---|---|
| data | a numeric vector |
| space | an instance of class BrainSpace |
| indices | a index vector indicating the 1-d coordinates of the data values |
| source | an instance of class BrainSource |
| label | a character string |

## Details

Image data is backed by `Matrix::sparseVector`.

## Value

`SparseBrainVolume` instance

## Slots

`data` a `sparseVector` instance

## Examples

```
data <- 1:10
indices <- seq(1,1000, length.out=10)
bspace <- BrainSpace(c(64,64,64), spacing=c(1,1,1))
sparsevol <- SparseBrainVolume(data,bspace,indices=indices)
densevol <- BrainVolume(data,bspace,indices=indices)
sum(sparsevol) == sum(densevol)
```

---

# splitFill *Generic function to fill disjoint sets of values with the output of a function*

---

## Description

Generic function to fill disjoint sets of values with the output of a function

## Usage

```
splitFill(x, fac, FUN)

## S4 method for signature 'BrainVolume,factor,`function`'
splitFill(x, fac, FUN)
```

## Arguments

| | |
|---|---|
| `x` | the object to split |
| `fac` | the factor to split by |
| `FUN` | the function to summarize the the sets |

## Details

`FUN` can either return a scalar for each input vector or a vector equal to the length of the input vector. If it returns a scalar then every voxel in the set will be filled with that value in the output vector.

## Value

a new object where the original values have been replaced by the function output

## Examples

```
## summarize with mean -- FUN returns a scalar
x = BrainSpace(c(10,10,10), c(1,1,1))
vol <- BrainVolume(rnorm(10*10*10), x)
fac <- factor(rep(1:10, length.out=1000))
ovol.mean <- splitFill(vol, fac, mean)
identical(dim(ovol.mean), dim(vol))
length(unique(as.vector(ovol.mean))) == 10
## transform by reversing vector -- FUN returns a vector.
ovol2 <- splitFill(vol, fac, rev)
```

---

# splitReduce *Generic function to summarize subsets of an object by first splitting by row and then "reducing" by a summary* function

---

## Description

Generic function to summarize subsets of an object by first splitting by row and then "reducing" by a summary function

## Usage

```
splitReduce(x, fac, FUN)

## S4 method for signature 'matrix,integer,`function`'
splitReduce(x, fac, FUN)

## S4 method for signature 'matrix,integer,missing'
splitReduce(x, fac)

## S4 method for signature 'matrix,factor,missing'
splitReduce(x, fac)

## S4 method for signature 'matrix,factor,`function`'
splitReduce(x, fac, FUN)

## S4 method for signature 'BrainVector,factor,`function`'
splitReduce(x, fac, FUN)

## S4 method for signature 'BrainVector,factor,missing'
splitReduce(x, fac, FUN)
```

### Arguments

| | |
|---|---|
| x | a numeric matrix(like) object |
| fac | the factor to define subsets of the object |
| FUN | the function to apply to each subset. if FUN is missing, than the mean of each sub-matrix column is computed. |

### Details

if FUN is supplied it must take a vector and return a single scalar value. If it returns more than one value, an error will occur.

if x is a BrainVector instance then voxels (dims 1:3) are treated as columns and time-series (dim 4) as rows. The summary function then is applied to groups of voxels. However, if the goal is to apply a function to groups of time-points, then this can be achieved as follows:

```
splitReduce(t(as.matrix(bvec)), fac)
```

### Value

a new matrix where the original values have been reduced

### Examples

```
mat = matrix(rnorm(100*100), 100, 100)
fac = sample(1:3, nrow(mat), replace=TRUE)
## compute column means of each sub-matrix
ms <- splitReduce(mat, fac)
all.equal(row.names(ms), levels(fac))

## compute column medians of each sub-matrix
ms <- splitReduce(mat, fac, median)

## compute time-series means grouped over voxels.
## Here, \code{length(fac)} must equal the number of voxels: \code{prod(dim(bvec)[1:3]}
bvec <- BrainVector(array(rnorm(24*24*24*24), c(24,24,24,24)), BrainSpace(c(24,24,24,24), c(1,1,1)))
fac <- factor(sample(1:3, prod(dim(bvec)[1:3]), replace=TRUE))
ms <- splitReduce(bvec, fac)
ms2 <- splitReduce(bvec, fac, mean)
all.equal(row.names(ms), levels(fac))
all.equal(ms,ms2)
```

---

## splitScale — *Generic function to center/scale row-subsets of a matrix or matrix-like object*

---

### Description

Generic function to center/scale row-subsets of a matrix or matrix-like object

## Usage

```
splitScale(x, f, center, scale)

## S4 method for signature 'matrix,factor,logical,logical'
splitScale(x, f, center = TRUE,
  scale = TRUE)

## S4 method for signature 'matrix,factor,missing,missing'
splitScale(x, f)
```

## Arguments

| | |
|---|---|
| x | a numeric matrix or matrix-like object |
| f | the splitting object, typically a factor or set of integer indices. must be equal to number of rows of matrix. |
| center | should values within each submatrix be centered? (mean removed from each column of submatrix) |
| scale | should values be scaled? (divide vector by standard deviation from each column of submatrix) |

## Value

a new matrix or matrix-like object where the original rows have been grouped by f and then centered and/or scaled for each grouping

## Examples

```
M <- matrix(rnorm(1000), 10, 100)
fac <- factor(rep(1:2, each=5))
Ms <- splitScale(M, fac)

## correctly centered
all(abs(apply(Ms[fac == 1,], 2, mean)) < .000001)
all(abs(apply(Ms[fac == 2,], 2, mean)) < .000001)

# correctly scaled
all.equal(apply(Ms[fac == 1,], 2, sd), rep(1, ncol(Ms)))
all.equal(apply(Ms[fac == 2,], 2, sd), rep(1, ncol(Ms)))
```

---

# stripExtension *Generic function to strip extension from file name, given a* BrainFileDescriptor *instance.*

---

## Description

Generic function to strip extension from file name, given a BrainFileDescriptor instance.

### Usage

```
stripExtension(x, fileName)

## S4 method for signature 'BrainFileDescriptor,character'
stripExtension(x, fileName)
```

### Arguments

| | |
|---|---|
| x | descriptor instance |
| fileName | file name to be stripped of its extension |

### Value

fileName without extension

---

## subVector *Generic function to extract a sub-vector from a* BrainVector *object.*

---

### Description

Generic function to extract a sub-vector from a BrainVector object.

### Usage

```
subVector(x, i, ...)

## S4 method for signature 'SparseBrainVector,numeric'
subVector(x, i)

## S4 method for signature 'DenseBrainVector,numeric'
subVector(x, i)
```

### Arguments

| | |
|---|---|
| x | four-dimensional image |
| i | the indices of the volume(s) to extract |
| ... | additional arguments |

### Value

a BrainVector object that is a sub-vector of the supplied object.

## Examples

```
bvec <- BrainVector(array(rnorm(24*24*24*24), c(24,24,24,24)), BrainSpace(c(24,24,24,24), c(1,1,1)))
vec <- subVector(bvec,1:2)
all.equal(2, dim(vec)[4])

vec <- subVector(bvec, c(1,3,5,7))
all.equal(4, dim(vec)[4])

mask <- LogicalBrainVolume(rep(TRUE, 24*24*24), BrainSpace(c(24,24,24), c(1,1,1)))
svec <- SparseBrainVector(array(rnorm(24*24*24*24), c(24,24,24,24)),
BrainSpace(c(24,24,24,24), c(1,1,1)), mask)
vec <- subVector(svec, c(1,3,5))
all.equal(3, dim(vec)[4])
```

---

# SurfaceDataMetaInfo *Constructor for* SurfaceDataMetaInfo *class*

---

## Description

Constructor for SurfaceDataMetaInfo class

## Usage

```
SurfaceDataMetaInfo(descriptor, header)
```

## Arguments

| | |
|---|---|
| descriptor | the file descriptor |
| header | a list containing header information |

---

# SurfaceDataMetaInfo-class *SurfaceDataMetaInfo This class contains meta information for surface-based data (the values that map to a surface geometry)*

---

## Description

SurfaceDataMetaInfo

This class contains meta information for surface-based data (the values that map to a surface geometry)

### Slots

`headerFile` name of the file containing meta information

`dataFile` name of the file containing data

`fileDescriptor` descriptor of image file format

`nodeCount` the number of nodes for which surface data exists

`nels` the number of data vectors (typically the number of columns in the surface data matrix; nels = 1 for a single surface data set)

`label` a label indicating the type of surface (e.g. white, pial, inflated, flat, spherical)

---

`SurfaceGeometryMetaInfo` *Constructor for* `SurfaceGeometryMetaInfo` *class*

---

### Description

Constructor for `SurfaceGeometryMetaInfo` class

### Usage

```
SurfaceGeometryMetaInfo(descriptor, header)
```

### Arguments

| | |
|---|---|
| `descriptor` | the file descriptor |
| `header` | a `list` containing header information |

---

`SurfaceGeometryMetaInfo-class` *SurfaceGeometryMetaInfo This class contains meta information for brain surface geometry*

---

### Description

SurfaceGeometryMetaInfo

This class contains meta information for brain surface geometry

### Slots

`headerFile` name of the file containing meta information

`dataFile` name of the file containing data

`fileDescriptor` descriptor of image file format

`vertices` the number of surface vertices

`faces` the number of faces

`embedDimension` the dimensionality of the embedding

`label` a label indicating the type of surface (e.g. white, pial, inflated, flat, spherical)

## takeSeries *Generic function to extract a set of series from a 4D image*

### Description

Generic function to extract a set of series from a 4D image

### Usage

```
takeSeries(x, indices, ...)
```

### Arguments

| | |
|---|---|
| x | a four dimensional image |
| indices | the indices of the series' to extract |
| ... | additional arguments |

## takeVolume *Generic function to extract a one or more individual volumes from a four-dimensional image*

### Description

Generic function to extract a one or more individual volumes from a four-dimensional image

### Usage

```
takeVolume(x, i, ...)

## S4 method for signature 'SparseBrainVector,numeric'
takeVolume(x, i, merge = FALSE)

## S4 method for signature 'BrainVector,numeric'
takeVolume(x, i, merge = FALSE)
```

### Arguments

| | |
|---|---|
| x | four-dimensional image |
| i | the indices of the volume(s) to extract |
| ... | additional arguments |
| merge | concatenate extracted volumes |

### Value

a list of BrainVolume elements

## Examples

```
bvec <- BrainVector(array(rnorm(24*24*24*24), c(24,24,24,24)), BrainSpace(c(24,24,24,24), c(1,1,1)))
vol <- takeVolume(bvec,1)
all.equal(dim(vol), c(24,24,24))

vol <- takeVolume(bvec,1:3)
length(vol) == 3
class(vol) == "list"
```

---

`tesselate` *tesselate*

---

## Description

tesselate

tesselate a LogicalBrainVolume into K spatial disjoint components

## Usage

```
tesselate(x, K, ...)

## S4 method for signature 'LogicalBrainVolume,numeric'
tesselate(x, K, features = NULL,
  spatialWeight = 4)
```

## Arguments

| | |
|---|---|
| `x` | the object to tesselate |
| `K` | the number of partitions |
| `...` | extra arguments |
| `features` | use addiitonal feature set to tesselate volume |
| `spatialWeight` | weight voxels according to distance |

---

`trans` *Generic getter to extract image coordinate transformation*

---

## Description

Generic getter to extract image coordinate transformation

## Usage

```
trans(x)

## S4 method for signature 'BrainMetaInfo'
trans(x)

## S4 method for signature 'NIfTIMetaInfo'
trans(x)

## S4 method for signature 'BrainSpace'
trans(x)

## S4 method for signature 'BrainData'
trans(x)
```

## Arguments

| | |
|---|---|
| x | an object with a transformation |

## Details

This function returns a transformation that can be used to go from "grid coordinates" to "real world coordinates" in millimeters.

This function returns a transformation that can be used to go from "grid coordinates" to "real world coordinates" in millimeters. see BrainSpace

## Examples

```
bspace <- BrainSpace(c(10,10,10), c(2,2,2))
trans(bspace)
all.equal(dim(trans(bspace)), c(4,4))
```

---

# values *Generic function to extract data values of object*

---

## Description

Generic function to extract data values of object

## Usage

```
values(x, ...)

## S4 method for signature 'ROIVolume'
values(x, ...)
```

### Arguments

| | |
|---|---|
| x | the object to get values from |
| ... | additional arguments |

---

## voxels *extract voxel coordinates*

---

### Description

extract voxel coordinates

### Usage

```
voxels(x, ...)

## S4 method for signature 'Kernel'
voxels(x, centerVoxel = NULL)
```

### Arguments

| | |
|---|---|
| x | the object to extract voxels from |
| ... | additional arguments to function |
| centerVoxel | the absolute location of the center of the voxel, default is (0,0,0) |

---

## writeElements *Generic function to write a sequence of elements from an input source*

---

### Description

Generic function to write a sequence of elements from an input source

writeElements

### Usage

```
writeElements(x, els)

## S4 method for signature 'BinaryWriter,numeric'
writeElements(x, els)
```

### Arguments

| | |
|---|---|
| x | the output channel |
| els | the elements to write |

writeVector *Generic function to write a 4D image vector to disk*

## Description

Generic function to write a 4D image vector to disk

## Usage

```
writeVector(x, fileName, format, dataType)

## S4 method for signature 'BrainVector,character,missing,missing'
writeVector(x, fileName)

## S4 method for signature 'BrainVector,character,character,missing'
writeVector(x, fileName,
  format)

## S4 method for signature 'BrainVector,character,missing,character'
writeVector(x, fileName,
  dataType)
```

## Arguments

| | |
|---|---|
| x | an image object, typically a BrainVector instance. |
| fileName | output file name. |
| format | file format string. Since "NIFTI" is the only currently supported format, this parameter can be safely ignored and omitted. |
| dataType | the numeric data type. If specified should be a character vector of: "BINARY", "UBYTE", "SHORT", "INT", "FLOAT", "DOUBLE". Otherwise output format will be inferred from R the datatype of the image. |

## Examples

```
bvec <- BrainVector(array(0, c(10,10,10,10)), BrainSpace(c(10,10,10,10), c(1,1,1)))
## Not run:
writeVector(bvol, "out.nii")
writeVector(bvol, "out.nii.gz")
writeVector(bvec, "out.nii")
writeVector(bvec, "out.nii.gz")

## End(Not run)
```

writeVolume *Generic function to write a 3D image volume to disk*

## Description

Generic function to write a 3D image volume to disk

## Usage

```
writeVolume(x, fileName, format, dataType)

## S4 method for signature 'BrainVolume,character,missing,missing'
writeVolume(x, fileName)

## S4 method for signature 'ClusteredBrainVolume,character,missing,missing'
writeVolume(x,
  fileName)

## S4 method for signature 'BrainVolume,character,character,missing'
writeVolume(x, fileName,
  format)

## S4 method for signature 'BrainVolume,character,missing,character'
writeVolume(x, fileName,
  dataType)
```

## Arguments

| | |
|---|---|
| x | an image object, typically a BrainVolume instance. |
| fileName | output file name |
| format | file format string. Since "NIFTI" is the only currently supported format, this parameter can be safely ignored and omitted. |
| dataType | output data type, If specified should be a character vector of: "BINARY", "UBYTE", "SHORT", "INT", "FLOAT", "DOUBLE". Otherwise output format will be inferred from R the datatype of the image. |

## Details

The output format will be inferred from file extension.

The output format will be inferred from file extension. writeVolume(x, "out.nii") outputs a NIFTI file. writeVolume(x, "out.nii.gz") outputs a gzipped NIFTI file.

No other file output formats are currently supported.

## Examples

```
bvol <- BrainVolume(array(0, c(10,10,10)), BrainSpace(c(10,10,10), c(1,1,1)))
## Not run:
writeVolume(bvol, "out.nii")
writeVolume(bvol, "out.nii.gz")

## End(Not run)
```

---

`[,BrainBucket,index,missing,ANY-method`
*extract labeled volume from* `BrainBucket`

---

## Description

extract labeled volume from `BrainBucket`

## Usage

```
## S4 method for signature 'BrainBucket,index,missing,ANY'
x[i]
```

## Arguments

| | |
|---|---|
| `x` | the object |
| `i` | first index |

---

`[,ROIVolume,numeric,missing,ANY-method`
*extract data from* `ROIVolume`

---

## Description

extract data from `ROIVolume`

## Usage

```
## S4 method for signature 'ROIVolume,numeric,missing,ANY'
x[i, j, drop]
```

## Arguments

| | |
|---|---|
| `x` | the object |
| `i` | first index |
| `j` | second index |
| `drop` | drop dimension |

## [,SparseBrainVector,missing,missing,ANY-method *extractor*

### Description

extractor

### Usage

```
## S4 method for signature 'SparseBrainVector,missing,missing,ANY'
x[i, j, k, m, ...,
  drop = TRUE]
```

### Arguments

| | |
|---|---|
| x | the object |
| i | first index |
| j | second index |
| k | third index |
| m | the fourth index |
| ... | additional args |
| drop | dimension |

## [,SparseBrainVector,missing,numeric,ANY-method *extractor*

### Description

extractor

### Usage

```
## S4 method for signature 'SparseBrainVector,missing,numeric,ANY'
x[i, j, k, m, ...,
  drop = TRUE]
```

### Arguments

| | |
|---|---|
| x | the object |
| i | first index |
| j | second index |
| k | third index |
| m | the fourth index |
| ... | additional args |
| drop | dimension |

---

## [,SparseBrainVector,numeric,missing,ANY-method
*extractor*

---

### Description

extractor

### Usage

```
## S4 method for signature 'SparseBrainVector,numeric,missing,ANY'
x[i, j, k, m, ...,
  drop = TRUE]
```

### Arguments

| | |
|---|---|
| x | the object |
| i | first index |
| j | second index |
| k | third index |
| m | the fourth index |
| ... | additional args |
| drop | dimension |

## [,SparseBrainVector,numeric,numeric,ANY-method *extractor*

### Description

extractor

### Usage

```
## S4 method for signature 'SparseBrainVector,numeric,numeric,ANY'
x[i, j, k, m, ...,
  drop = TRUE]
```

### Arguments

| | |
|---|---|
| x | the object |
| i | first index |
| j | second index |
| k | third index |
| m | the fourth index |
| ... | additional args |
| drop | dimension |

## [,SparseBrainVolume,matrix,missing,ANY-method *extractor*

### Description

extractor

### Usage

```
## S4 method for signature 'SparseBrainVolume,matrix,missing,ANY'
x[i, j, k, ...,
  drop = TRUE]
```

### Arguments

| | |
|---|---|
| x | the object |
| i | first index |
| j | second index |
| k | third index |
| ... | additional args |
| drop | dimension |

---

## [,SparseBrainVolume,missing,missing,ANY-method
*extractor*

---

### Description

extractor

### Usage

```
## S4 method for signature 'SparseBrainVolume,missing,missing,ANY'
x[i, j, k, ...,
  drop = TRUE]
```

### Arguments

| | |
|---|---|
| x | the object |
| i | first index |
| j | second index |
| k | third index |
| ... | additional args |
| drop | dimension |

---

`[,SparseBrainVolume,missing,numeric,ANY-method`
*extractor*

---

## Description

extractor

## Usage

```
## S4 method for signature 'SparseBrainVolume,missing,numeric,ANY'
x[i, j, k, ...,
  drop = TRUE]
```

## Arguments

| | |
|---|---|
| x | the object |
| i | first index |
| j | second index |
| k | third index |
| ... | additional args |
| drop | dimension |

---

`[,SparseBrainVolume,numeric,missing,missing-method`
*extractor*

---

## Description

extractor

## Usage

```
## S4 method for signature 'SparseBrainVolume,numeric,missing,missing'
x[i, j, k, ..., drop]
```

## Arguments

| | |
|---|---|
| x | the object |
| i | first index |
| j | second index |
| k | third index |
| ... | additional args |
| drop | drop dimension |

---

`[,SparseBrainVolume,numeric,numeric,ANY-method`
*extractor*

---

## Description

extractor

## Usage

```
## S4 method for signature 'SparseBrainVolume,numeric,numeric,ANY'
x[i, j, k, ...,
  drop = TRUE]
```

## Arguments

| | |
|---|---|
| `x` | the object |
| `i` | first index |
| `j` | second index |
| `k` | third index |
| `...` | additional args |
| `drop` | dimension |

---

`[[,BrainBucket,index,missing-method`
*extract labeled volume from* `BrainBucket`

---

## Description

extract labeled volume from `BrainBucket`

## Usage

```
## S4 method for signature 'BrainBucket,index,missing'
x[[i]]
```

## Arguments

| | |
|---|---|
| `x` | the object |
| `i` | the first index |

# Index