Package ‘ieegio’

March 7, 2025

Title File IO for Intracranial Electroencephalography

Version 0.0.4

Language en-US

Encoding UTF-8

Description Integrated toolbox supporting common file formats used for intracranial Electroencephalography (iEEG) and deep-brain stimulation (DBS) study.

URL `http://dipterix.org/ieegio/`

BugReports `https://github.com/dipterix/ieegio/issues`

License MIT + file LICENSE

RoxygenNote 7.3.2

Imports data.table (>= 1.16.0), digest, fastmap, filearray (>= 0.1.8), freesurferformats, fs, fst (>= 0.9.0), gifti (>= 0.8.0), grDevices, hdf5r, jsonlite, oro.nifti, R.matlab (>= 3.7.0), R6, readNSx (>= 0.0.5), rpyANTs (>= 0.0.3), stringr, utils, yaml

Suggests reticulate, ravetools, rgl, RNifti (>= 1.7.0), rpymat (>= 0.1.7), xml2, knitr, r3js, rmarkdown, tools, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Zhengjia Wang [aut, cre] (<`https://orcid.org/0000-0001-5629-1116`>)

Maintainer Zhengjia Wang `<dipterix.wang@gmail.com>`

Repository CRAN

Date/Publication 2025-03-07 20:30:05 UTC

Contents

`as_ieegio_surface` *Convert other surface formats to* `ieegio` *surface*

Description

Convert other surface formats to `ieegio` surface

Usage

```
as_ieegio_surface(x, ...)

## Default S3 method:
as_ieegio_surface(
  x,
  vertices = x,
  faces = NULL,
  face_start = NA,
  transform = NULL,
  vertex_colors = NULL,
  annotation_labels = NULL,
  annotation_values = NULL,
  measurements = NULL,
  time_series_slice_duration = NULL,
```

```
  time_series_value = NULL,
  name = NULL,
  ...
)

## S3 method for class 'character'
as_ieegio_surface(x, ...)

## S3 method for class 'ieegio_surface'
as_ieegio_surface(x, ...)

## S3 method for class 'mesh3d'
as_ieegio_surface(x, ...)

## S3 method for class 'fs.surface'
as_ieegio_surface(x, ...)
```

Arguments

`x`	R object or file path
`...`	passed to default method
`vertices`	`n` by 3 matrix, each row is a vertex node position
`faces`	(optional) face index, either zero or one-indexed (`Matlab` and `R` start counting from 1 while `C` and `Python` start indices from 0); one-index face order is recommended
`face_start`	(optional) either 0 or 1, indicating whether `faces` is zero or one-indexed; default is `NA`, which will check whether the minimum value of `faces` is 0. If so, then `faces` will be bumped by 1 internally
`transform`	(optional) a 4 by 4 matrix indicating the vertex position to scanner `RAS` transform. Default is missing (identity matrix), i.e. the vertex positions are already in the scanner `RAS` coordinate system.
`vertex_colors`	(optional) integer or color (hex) vector indicating the vertex colors
`annotation_labels`	(optional) a data frame containing at the following columns. Though optional, `annotation_labels` must be provided when `annotation_values` is provided `"Key"` unique integers to appear in `annotation_values`, indicating the key of the annotation label `"Label"` a character vector (strings) of human-readable labels of the corresponding key `"Color"` hex string indicating the color of the key/label
`annotation_values`	(optional) an integer table where each column is a vector of annotation key (for example, 'FreeSurfer' segmentation key) and each row corresponds to a vertex node
`measurements`	(optional) a numeric table where each column represents a variable (for example, curvature) and each row corresponds to a vertex node. Unlike annotations, which is for discrete node values, `measurements` is for continuous values

`time_series_slice_duration`	(optional) a numeric vector indicating the duration of each slice; default is `NA`
`time_series_value`	(optional) a numeric matrix (`n` by `m`) where `n` is the number of vertices and `m` is the number of time points, hence each column is a time slice and each row is a vertex node.
`name`	(optional) name of the geometry

Value

An `ieeg_surface` object; see `read_surface` or 'Examples'.

Examples

```
# ---- Simple usage
# vertices only
dodecahedron_vert <- matrix(
  ncol = 3, byrow = TRUE,
  c(-0.62, -0.62, -0.62, 0.62, -0.62, -0.62, -0.62, 0.62, -0.62,
    0.62, 0.62, -0.62, -0.62, -0.62, 0.62, 0.62, -0.62, 0.62,
    -0.62, 0.62, 0.62, 0.62, 0.62, 0.62, 0.00, -0.38, 1.00,
    0.00, 0.38, 1.00, 0.00, -0.38, -1.00, 0.00, 0.38, -1.00,
    -0.38, 1.00, 0.00, 0.38, 1.00, 0.00, -0.38, -1.00, 0.00,
    0.38, -1.00, 0.00, 1.00, 0.00, -0.38, 1.00, 0.00, 0.38,
    -1.00, 0.00, -0.38, -1.00, 0.00, 0.38)
)

point_cloud <- as_ieegio_surface(dodecahedron_vert)
plot(point_cloud, col = "red")

# with face index
dodecahedron_face <- matrix(
  ncol = 3L, byrow = TRUE,
  c(1, 11, 2, 1, 2, 16, 1, 16, 15, 1, 15, 5, 1, 5, 20, 1, 20, 19,
    1, 19, 3, 1, 3, 12, 1, 12, 11, 2, 11, 12, 2, 12, 4, 2, 4, 17,
    2, 17, 18, 2, 18, 6, 2, 6, 16, 3, 13, 14, 3, 14, 4, 3, 4, 12,
    3, 19, 20, 3, 20, 7, 3, 7, 13, 4, 14, 8, 4, 8, 18, 4, 18, 17,
    5, 9, 10, 5, 10, 7, 5, 7, 20, 5, 15, 16, 5, 16, 6, 5, 6, 9,
    6, 18, 8, 6, 8, 10, 6, 10, 9, 7, 10, 8, 7, 8, 14, 7, 14, 13)
)
mesh <- as_ieegio_surface(dodecahedron_vert,
                          faces = dodecahedron_face)
plot(mesh)

# with vertex colors
mesh <- as_ieegio_surface(dodecahedron_vert,
                          faces = dodecahedron_face,
                          vertex_colors = sample(20))
plot(mesh, name = "color")
```

```
# with annotations
mesh <- as_ieegio_surface(
  dodecahedron_vert,
  faces = dodecahedron_face,
  annotation_labels = data.frame(
    Key = 1:3,
    Label = c("A", "B", "C"),
    Color = c("red", "green", "blue")
  ),
  annotation_values = data.frame(
    MyVariable = c(rep(1, 7), rep(2, 7), rep(3, 6))
  )
)
plot(mesh, name = "annotations")

# with measurements
mesh <- as_ieegio_surface(
  dodecahedron_vert,
  faces = dodecahedron_face,
  measurements = data.frame(
    MyVariable = dodecahedron_vert[, 1]
  )
)
plot(mesh, name = "measurements",
     col = c("blue", "gray", "red"))
```

`as_ieegio_volume` *Convert objects to* 'ieegio' *image volumes*

Description

Convert array, path, or 'NIfTI' images in other formats to `'ieegio'` image volume instance

Usage

```
as_ieegio_volume(x, ...)

## S3 method for class 'character'
as_ieegio_volume(x, ...)

## S3 method for class 'ieegio_volume'
as_ieegio_volume(x, ...)

## S3 method for class 'array'
as_ieegio_volume(x, vox2ras = NULL, as_color = is.character(x), ...)
```

```
## S3 method for class 'niftiImage'
as_ieegio_volume(x, ...)

## S3 method for class 'nifti'
as_ieegio_volume(x, ...)

## S3 method for class 'ants.core.ants_image.ANTsImage'
as_ieegio_volume(x, ...)
```

Arguments

`x`	R object such as array, image path, or objects such as `'RNifti'` or `'oro.nifti'` image instances
`...`	passed to other methods
`vox2ras`	a 4x4 'affine' matrix representing the transform from 'voxel' index (column-row-slice) to 'RAS' (right-anterior-superior) coordinate. This transform is often called `'xform'`, `'sform'`, `'qform'` in 'NIfTI' terms, or `'Norig'` in 'FreeSurfer'
`as_color`	for converting arrays to volume, whether to treat `x` as array of colors; default is true when `x` is a raster matrix (matrix of color strings) and false when `x` is not a character array.

Value

An `ieegio` volume object; see `imaging-volume`

Examples

```
shape <- c(50, 50, 50)
vox2ras <- matrix(
  c(-1, 0, 0, 25,
    0, 0, 1, -25,
    0, -1, 0, 25,
    0, 0, 0, 1),
  nrow = 4, byrow = TRUE
)

# continuous
x <- array(rnorm(125000), shape)

volume <- as_ieegio_volume(x, vox2ras = vox2ras)
plot(volume, zoom = 3, pixel_width = 0.5)

# color rgb(a)
x <- array(
  sample(c("red","blue", "green", "cyan", "yellow"),
         12500, replace = TRUE),
  shape
)
rgb <- as_ieegio_volume(x, vox2ras = vox2ras)
```

```
plot(rgb, zoom = 3, pixel_width = 0.5)
```

`burn_volume` *Burn image at given positions*

Description

Burn image at given positions with given color and radius.

Usage

```
burn_volume(
  image,
  ras_position,
  col = "red",
  radius = 1,
  reshape = FALSE,
  alpha = FALSE,
  blank_underlay = FALSE,
  ...,
  preview = NULL
)
```

Arguments

`image`	volume
`ras_position`	image-defined right-anterior-posterior positions, an `nx3` matrix, each row is an 'RAS' coordinate
`col`	vector of integer or characters, color of each contact
`radius`	vector of positive number indicating the burning radius
`reshape`	whether to reshape the image at a different resolution; default is false; can be `TRUE` (image resolution will be doubled), a single number (size of isotropic volume along one side), or a length of three defining the new shape.
`alpha`	whether to include alpha (transparent) channel. Default is false for compatibility concerns (legacy software might not support reading alpha channel). In this case, the background will be black. If `alpha=TRUE` is set, then the background will be fully transparent.
`blank_underlay`	whether to use blank image or the input `image` as underlay; default is `FALSE` (using `image` as underlay); alternative is `TRUE`, and use black or transparent background
`...`	passed to `as_ieegio_volume`, useful if `image` is an array
`preview`	indices (integer) of the position to visualize; default is `NULL` (no preview)

Value

Color image that is burnt; see `imaging-volume`.

Examples

```
if(interactive()) {

dim <- c(6, 6, 6)
image <- as_ieegio_volume(
  array(rnorm(prod(dim)), dim),
  vox2ras = rbind(cbind(diag(1, 3), -dim / 2),
                  c(0, 0, 0, 1))
)

ras_positions <- rbind(c(1, -1, 1.5), c(-2.25, -1, -0.75))


burned <- burn_volume(
  image,
  ras_positions,
  col = c("red", "green"),
  radius = 0.5,
  reshape = c(24, 24, 24)
)

plot(
  burned,
  position = ras_positions[1, ],
  zoom = 15,
  pixel_width = 0.25
)

}
```

`ieegio_sample_data` *Download sample files*

Description

Download sample files

Usage

```
ieegio_sample_data(file, test = FALSE, cache_ok = TRUE)
```

Arguments

`file`	file to download; set to `NULL` to view all possible files
`test`	test whether the sample file exists instead of downloading them; default is `FALSE`
`cache_ok`	whether to use cache

Value

When `test` is false, returns downloaded file path (character); when `test` is true, returns whether the expected sample exists (logical).

Examples

```
# list available files
ieegio_sample_data()

# check if file edfPlusD.edf exists
ieegio_sample_data("edfPlusD.edf", test = TRUE)

## Not run:

ieegio_sample_data("edfPlusD.edf")


## End(Not run)
```

`imaging-surface` *Read and write surface files*

Description

Supports surface geometry, annotation, measurement, and time-series data. Please use the high-level function `read_surface`, which calls other low-level functions internally.

Usage

```
read_surface(file, format = "auto", type = NULL, ...)

write_surface(
  x,
  con,
  format = c("gifti", "freesurfer"),
  type = c("geometry", "annotations", "measurements", "color", "time_series"),
  ...,
  name = 1
)

io_read_fs(
  file,
  type = c("geometry", "annotations", "measurements"),
  format = "auto",
  name = basename(file),
  ...
)
```

```
io_read_gii(file)

io_write_gii(x, con, ...)
```

Arguments

`file, con`	path the file
`format`	format of the file, for `write_surface`, this is either `'gifti'` or `'freesurfer'`; for `read_surface`, see 'Arguments' section in `read.fs.surface` (when file type is `'geometry'`) and `read.fs.curv` (when file type is `'measurements'`)
`type`	type of the data; ignored if the file format is 'GIfTI'. For 'FreeSurfer' files, supported types are `'geometry'` contains positions of mesh vertex nodes and face indices; `'annotations'` annotation file (usually with file extension `'annot'`) containing a color look-up table and an array of color keys. These files are used to display discrete values on the surface such as brain atlas; `'measurements'` measurement file such as `'sulc'` and `'curv'` files, containing numerical values (often with continuous domain) for each vertex node
`...`	for `read_surface`, the arguments will be passed to `io_read_fs` if the file is a 'FreeSurfer' file.
`x`	surface (geometry, annotation, measurement) data
`name`	name of the data; for `io_read_fs`, this argument must be a character, and default is the file name; for `write_surface`, this argument can be an integer or a character, representing the index or name of the corresponding measurement or annotation column.

Value

A surface object container for `read_surface`, and the file path for `write_surface`

Examples

```
library(ieegio)

# geometry
geom_file <- "gifti/GzipBase64/sujet01_Lwhite.surf.gii"

# measurements
shape_file <- "gifti/GzipBase64/sujet01_Lwhite.shape.gii"

# time series
ts_file <- "gifti/GzipBase64/fmri_sujet01_Lwhite_projection.time.gii"

if(ieegio_sample_data(geom_file, test = TRUE)) {

  geometry <- read_surface(ieegio_sample_data(geom_file))
```

```
  print(geometry)

  measurement <- read_surface(ieegio_sample_data(shape_file))
  print(measurement)

  time_series <- read_surface(ieegio_sample_data(ts_file))
  print(time_series)

  # merge measurement & time_series into geometry
  merged <- merge(geometry, measurement, time_series)
  print(merged)

  # make sure you install `rgl` package
  plot(merged, name = c("measurements", "Shape001"))

  plot(merged, name = "time_series",
       slice_index = c(1, 11, 21, 31))

}
```

`imaging-volume` *Read and write volume data*

Description

Read and write volume data ('MRI', 'CT', etc.) in 'NIfTI' or 'MGH' formats. Please use `read_volume` and `write_volume` for high-level function. These functions will call other low-level functions internally.

Usage

```
read_volume(file, header_only = FALSE, format = c("auto", "nifti", "mgh"), ...)

write_volume(x, con, format = c("auto", "nifti", "mgh"), ...)

io_read_mgz(file, header_only = FALSE)

io_write_mgz(x, con, ...)

## S3 method for class 'ieegio_volume'
io_write_mgz(x, con, ...)

## S3 method for class 'ieegio_mgh'
io_write_mgz(x, con, ...)

## S3 method for class 'nifti'
io_write_mgz(x, con, ...)
```

```
## S3 method for class 'niftiImage'
io_write_mgz(x, con, ...)

## S3 method for class 'ants.core.ants_image.ANTsImage'
io_write_mgz(x, con, ...)

## S3 method for class 'array'
io_write_mgz(x, con, vox2ras = NULL, ...)

io_read_nii(
  file,
  method = c("rnifti", "oro", "ants"),
  header_only = FALSE,
  ...
)

io_write_nii(x, con, ...)

## S3 method for class 'ieegio_nifti'
io_write_nii(x, con, ...)

## S3 method for class 'ants.core.ants_image.ANTsImage'
io_write_nii(x, con, ...)

## S3 method for class 'niftiImage'
io_write_nii(x, con, ...)

## S3 method for class 'nifti'
io_write_nii(x, con, gzipped = NA, ...)

## S3 method for class 'ieegio_mgh'
io_write_nii(x, con, ...)

## S3 method for class 'array'
io_write_nii(
  x,
  con,
  vox2ras = NULL,
  datatype_code = NULL,
  xyzt_units = c("NIFTI_UNITS_MM", "NIFTI_UNITS_SEC"),
  intent_code = "NIFTI_INTENT_NONE",
  ...,
  gzipped = NA
)
```

Arguments

`file`	file path to read volume data

`header_only`	whether to read header data only; default is `FALSE`
`format`	format of the file to be written; choices are `'auto'`, `'nifti'` or `'mgh'`; default is to `'auto'` detect the format based on file names, which will save as a 'MGH' file when file extension is `'mgz'` or `'mgh'`, otherwise 'NIfTI' format. We recommend explicitly setting this argument
`...`	passed to other methods
`x`	volume data (such as 'NIfTI' image, array, or 'MGH') to be saved
`con`	file path to store image
`vox2ras`	a `4x4` transform matrix from voxel indexing (column, row, slice) to scanner (often 'T1-weighted' image) 'RAS' (right-anterior-superior) coordinate
`method`	method to read the file; choices are `'oro'` (using `readNIfTI`), `'rnifti'` (using `readNifti`), and `'ants'` (using `as_ANTsImage`).
`gzipped`	for writing `'nii'` data: whether the file needs to be compressed; default is inferred from the file name. When the file ends with `'nii'`, then no compression is used; otherwise the file will be compressed. If the file name does not end with `'nii'` nor `'nii.gz'`, then the file extension will be added automatically.
`datatype_code`, `xyzt_units`, `intent_code`	additional flags for 'NIfTI' headers, for advanced users

Format

format of the file; default is auto-detection, other choices are `'nifti'` and `'mgh'`;

Value

Imaging readers return `ieegio_volume` objects. The writers return the file path to where the file is saved to.

Examples

```
library(ieegio)

nifti_file <- "brain.demosubject.nii.gz"

# Use `ieegio_sample_data(nifti_file)`
#   to download sample data


if( ieegio_sample_data(nifti_file, test = TRUE) ) {

# ---- NIfTI examples ---------------------------------------------

file <- ieegio_sample_data(nifti_file)

# basic read
vol <- read_volume(file)
```

```
# voxel to scanner RAS
vol$transforms$vox2ras

# to freesurfer surface
vol$transforms$vox2ras_tkr

# to FSL
vol$transforms$vox2fsl

plot(vol, position = c(10, 0, 30))

# ---- using other methods --------------------------------------
# default
vol <- read_volume(file, method = "rnifti", format = "nifti")
vol$header

# lazy-load nifti
vol2 <- read_volume(file, method = "oro", format = "nifti")
vol2$header

## Not run:
# requires additional python environment

# Using ANTsPyx
vol3 <- read_volume(file, method = "ants", format = "nifti")
vol3$header


## End(Not run)

# ---- write ---------------------------------------------------------

# write as NIfTI
f <- tempfile(fileext = ".nii.gz")

write_volume(vol, f, format = "nifti")

# alternative method
write_volume(vol$header, f, format = "nifti")

# write to mgz/mgh
f2 <- tempfile(fileext = ".mgz")

write_volume(vol, f, format = "mgh")

# clean up
unlink(f)
unlink(f2)

}
```

---

io_h5_valid *Check whether a 'HDF5' file can be opened for read/write*

Description

Check whether a 'HDF5' file can be opened for read/write

Usage

```
io_h5_valid(file, mode = c("r", "w"), close_all = FALSE)

io_h5_names(file)
```

Arguments

`file`	path to file
`mode`	`'r'` for read access and `'w'` for write access
`close_all`	whether to close all connections or just close current connection; default is false. Set this to `TRUE` if you want to close all other connections to the file

Value

`io_h5_valid` returns a logical value indicating whether the file can be opened. `io_h5_names` returns a character vector of dataset names.

Examples

```
x <- array(1:27, c(3,3,3))
f <- tempfile()

# No data written to the file, hence invalid
io_h5_valid(f, 'r')

io_write_h5(x, f, 'dset')
io_h5_valid(f, 'w')

# Open the file and hold a connection
ptr <- hdf5r::H5File$new(filename = f, mode = 'w')

# Can read, but cannot write
io_h5_valid(f, 'r')  # TRUE
io_h5_valid(f, 'w')  # FALSE

# However, this can be reset via `close_all=TRUE`
io_h5_valid(f, 'r', close_all = TRUE)
io_h5_valid(f, 'w')  # TRUE

# Now the connection is no longer valid
```

```
ptr

# clean up
unlink(f)
```

---

io_read_h5 *Lazy Load 'HDF5' File via* `hdf5r-package`

Description

Wrapper for class `LazyH5`, which load data with "lazy" mode - only read part of dataset when needed.

Usage

```
io_read_h5(file, name, read_only = TRUE, ram = FALSE, quiet = FALSE)
```

Arguments

`file`	'HDF5' file
`name`	`group/data_name` path to dataset (`H5D` data)
`read_only`	only used if `ram=FALSE`, whether the returned `LazyH5` instance should be read only
`ram`	load data to memory immediately, default is false
`quiet`	whether to suppress messages

Value

If `ram` is true, then return data as arrays, otherwise return a `LazyH5` instance.

See Also

`io_write_h5`

Examples

```
file <- tempfile()
x <- array(1:120, dim = c(4,5,6))

# save x to file with name /group/dataset/1
io_write_h5(x, file, '/group/dataset/1', quiet = TRUE)

# read data
y <- io_read_h5(file, '/group/dataset/1', ram = TRUE)
class(y)   # array

z <- io_read_h5(file, '/group/dataset/1', ram = FALSE)
```

```
class(z)   # LazyH5

dim(z)

# clean up
unlink(file)
```

---

io_write_h5 *Save objects to 'HDF5' file without trivial checks*

Description

Save objects to 'HDF5' file without trivial checks

Usage

```
io_write_h5(
  x,
  file,
  name,
  chunk = "auto",
  level = 4,
  replace = TRUE,
  new_file = FALSE,
  ctype = NULL,
  quiet = FALSE,
  ...
)
```

Arguments

`x`	an array, a matrix, or a vector
`file`	path to 'HDF5' file
`name`	path/name of the data; for example, `"group/data_name"`
`chunk`	chunk size
`level`	compress level from 0 - no compression to 10 - max compression
`replace`	should data be replaced if exists
`new_file`	should removing the file if old one exists
`ctype`	data type such as "character", "integer", or "numeric". If set to `NULL` then automatically detect types. Note for complex data please store separately the real and imaginary parts.
`quiet`	whether to suppress messages, default is false
`...`	passed to other `LazyH5$save`

Value

Absolute path of the file saved

See Also

`io_read_h5`

Examples

```
file <- tempfile()
x <- array(1:120, dim = 2:5)

# save x to file with name /group/dataset/1
io_write_h5(x, file, '/group/dataset/1', chunk = dim(x))

# load data
y <- io_read_h5(file, '/group/dataset/1')

# read data to memory
y[]

# clean up
unlink(file)
```

`LazyH5` *Lazy 'HDF5' file loader*

Description

Provides hybrid data structure for 'HDF5' file. The class is not intended for direct-use. Please see `io_read_h5` and `io_write_h5`.

Public fields

`quiet` whether to suppress messages

Methods

Public methods:

- `LazyH5$finalize()`
- `LazyH5$print()`
- `LazyH5$new()`
- `LazyH5$save()`
- `LazyH5$open()`
- `LazyH5$close()`
- `LazyH5$subset()`

- `LazyH5$get_dims()`
- `LazyH5$get_type()`

Method `finalize()`: garbage collection method

Usage:

```
LazyH5$finalize()
```

Returns: none

Method `print()`: overrides print method

Usage:

```
LazyH5$print()
```

Returns: self instance

Method `new()`: constructor

Usage:

```
LazyH5$new(file_path, data_name, read_only = FALSE, quiet = FALSE)
```

Arguments:

`file_path` where data is stored in 'HDF5' format

`data_name` the data stored in the file

`read_only` whether to open the file in read-only mode. It's highly recommended to set this to be true, otherwise the file connection is exclusive.

`quiet` whether to suppress messages, default is false

Returns: self instance

Method `save()`: save data to a 'HDF5' file

Usage:

```
LazyH5$save(
  x,
  chunk = "auto",
  level = 7,
  replace = TRUE,
  new_file = FALSE,
  force = TRUE,
  ctype = NULL,
  size = NULL,
  ...
)
```

Arguments:

`x` vector, matrix, or array

`chunk` chunk size, length should matches with data dimension

`level` compress level, from 1 to 9

`replace` if the data exists in the file, replace the file or not

`new_file` remove the whole file if exists before writing?

`force` if you open the file in read-only mode, then saving objects to the file will raise error. Use `force=TRUE` to force write data

`ctype` data type, see `mode`, usually the data type of `x`. Try `mode(x)` or `storage.mode(x)` as hints.

`size` deprecated, for compatibility issues

`...` passed to self `open()` method

Method `open()`: open connection

Usage:

```
LazyH5$open(new_dataset = FALSE, robj, ...)
```

Arguments:

`new_dataset` only used when the internal pointer is closed, or to write the data

`robj` data array to save

`...` passed to `createDataSet` in `hdf5r` package

Method `close()`: close connection

Usage:

```
LazyH5$close(all = TRUE)
```

Arguments:

`all` whether to close all connections associated to the data file. If true, then all connections, including access from other programs, will be closed

Method `subset()`: subset data

Usage:

```
LazyH5$subset(..., drop = FALSE, stream = FALSE, envir = parent.frame())
```

Arguments:

`drop` whether to apply `drop` the subset

`stream` whether to read partial data at a time

`envir` if `i,j,...` are expressions, where should the expression be evaluated

`i, j, ...` index along each dimension

Returns: subset of data

Method `get_dims()`: get data dimension

Usage:

```
LazyH5$get_dims(stay_open = TRUE)
```

Arguments:

`stay_open` whether to leave the connection opened

Returns: dimension of the array

Method `get_type()`: get data type

Usage:

```
LazyH5$get_type(stay_open = TRUE)
```

Arguments:

`stay_open` whether to leave the connection opened

Returns: data type, currently only character, integer, raw, double, and complex are available, all other types will yield "unknown"

`low-level-read-write` *Low-level file read and write*

Description

Interfaces to read from or write to files with common formats.

Usage

```
io_read_fst(
  con,
  method = c("proxy", "data_table", "data_frame", "header_only"),
  ...,
  old_format = FALSE
)

io_write_fst(x, con, compress = 50, ...)

io_read_ini(con, ...)

io_read_json(con, ...)

io_write_json(
  x,
  con = stdout(),
  ...,
  digits = ceiling(-log10(.Machine$double.eps)),
  pretty = TRUE,
  serialize = TRUE
)

io_read_mat(
  con,
  method = c("auto", "R.matlab", "pymatreader", "mat73"),
  verbose = TRUE,
  on_convert_error = c("warning", "error", "ignore"),
  ...
)

io_write_mat(x, con, method = c("R.matlab", "scipy"), ...)

io_read_yaml(con, ...)

io_write_yaml(x, con, ..., sorted = FALSE)
```

Arguments

`con`	connection or file

`method`	method to read table. For `'fst'`, the choices are `'proxy'` do not read data to memory, query the table when needed; `'data_table'` read as `data.table`; `'data_frame'` read as `data.frame`; `'header_only'` read `'fst'` table header. For `'mat'`, the choices are `'auto'` automatically try the native option, and then `'pymatreader'` if fails; `'R.matlab'` use the native method (provided by `readMat`); only support 'MAT 5.0' format; `'pymatreader'` use 'Python' library `'pymatreader'`; `'mat73'` use 'Python' library `'mat73'`.
`...`	passed to internal function calls
`old_format`	see `fst`
`x`	data to write to disk
`compress`	compress level from 0 to 100; default is 50
`digits, pretty`	for writing numeric values to 'json' format
`serialize`	set to `TRUE` to serialize the data to 'json' format (with the data types, default); or `FALSE` to save the values without types
`verbose`	whether to print out the process
`on_convert_error`	for reading `'mat'` files with 'Python' modules, the results will be converted to R objects in the end. Not all objects can be converted. This input defines the behavior when the conversion fails; choices are `"error"`, `"warning"`, or `"ignore"`
`sorted`	whether to sort the list; default is `FALSE`

Value

The reader functions returns the data extracted from files, mostly as R objects, with few exceptions on some 'Matlab' files. When reading a 'Matlab' file requires using 'Python' modules, `io_read_mat` will try its best effort to convert 'Python' objects to R. However, such conversion might fail. In this case, the result might partially contain 'Python' objects with warnings.

Examples

```
# ---- fst ----------------------------------------------------------------


f <- tempfile(fileext = ".fst")
x <- data.frame(
  a = 1:10,
  b = rnorm(10),
  c = letters[1:10]
)
```

```
io_write_fst(x, con = f)

# default reads in proxy
io_read_fst(f)

# load as data.table
io_read_fst(f, "data_table")

# load as data.frame
io_read_fst(f, "data_frame")

# get header
io_read_fst(f, "header_only")

# clean up
unlink(f)



# ---- json ---------------------------------------------------------------
f <- tempfile(fileext = ".json")

x <- list(a = 1L, b = 2.3, c = "a", d = 1+1i)

# default is serialize
io_write_json(x, f)

io_read_json(f)

cat(readLines(f), sep = "\n")

# just values
io_write_json(x, f, serialize = FALSE, pretty = FALSE)

io_read_json(f)

cat(readLines(f), sep = "\n")

# clean up
unlink(f)



# ---- Matlab .mat --------------------------------------------------------

## Not run:

f <- tempfile(fileext = ".mat")

x <- list(a = 1L, b = 2.3, c = "a", d = 1+1i)

# save as MAT 5.0
io_write_mat(x, f)
```

```
io_read_mat(f)

# require setting up Python environment

io_read_mat(f, method = "pymatreader")

# MAT 7.3 example
sample_data <- ieegio_sample_data("mat_v73.mat")
io_read_mat(sample_data)

# clean up
unlink(f)


## End(Not run)



# ---- yaml ---------------------------------------------------------------

f <- tempfile(fileext = ".yaml")

x <- list(a = 1L, b = 2.3, c = "a")
io_write_yaml(x, f)

io_read_yaml(f)

# clean up
unlink(f)
```

`merge.ieegio_surface` *Merge two* 'ieegio' *surfaces*

Description

Either merge surface objects by attributes or merge geometries

Usage

```
## S3 method for class 'ieegio_surface'
merge(
  x,
  y,
  ...,
  merge_type = c("attribute", "geometry"),
  merge_space = c("model", "world"),
  transform_index = 1,
  verbose = TRUE
)
```

Arguments

`x, y, ...`	`'ieegio'` surface objects, see `as_ieegio_surface` or `read_surface`. Object `x` must contain geometry information.
`merge_type`	type of merge: `"attribute"` merge `y,...` into `x` by attributes such as color, measurements, annotations, or time-series data, assuming `x,y,...` all refer to the same geometry, hence the underlying number of vertices should be the same. `"geometry"` merge `y,...` into `x` by geometry; this requires the surfaces to merge have geometries and cannot be only surface attributes. Two mesh objects will be merged into one, and face index will be re-calculated. The merge happens in transformed space, Notice the attributes will be ignored and eventually discarded during merge.
`merge_space`	space to merge the geometries; only used when `merge_type` is `"geometry"`. Default is to directly merge the surfaces in `"model"` space, i.e. assuming the surfaces share the same transform; alternatively, if the model to world transforms are different, users can choose to merge in `"world"` space, then all the surfaces will be transformed into world space and mapped back to the model space in `x`
`transform_index`	which local-to-world transform to use when merging geometries in the world space; default is the first transform for each surface object. The transform list can be obtained from `surface$geometry$transforms` and `transform_index` indicates the index of the transform matrices. The length of `transform_index` can be either 1 (same for all surfaces) or the length of all the surfaces, (i.e. length of `list(x,y,...)`), when the index needs to be set for each surface respectively. If any index is set to `NA`, then it means no transform is to be applied and that surface will be merged assuming its model space is the world space.
`verbose`	whether to verbose the messages

Value

A merged surface object

Examples

```
# Construct example geometry
dodecahedron_vert <- matrix(
  ncol = 3, byrow = TRUE,
  c(-0.62, -0.62, -0.62, 0.62, -0.62, -0.62, -0.62, 0.62, -0.62,
    0.62, 0.62, -0.62, -0.62, -0.62, 0.62, 0.62, -0.62, 0.62,
    -0.62, 0.62, 0.62, 0.62, 0.62, 0.62, 0.00, -0.38, 1.00,
    0.00, 0.38, 1.00, 0.00, -0.38, -1.00, 0.00, 0.38, -1.00,
    -0.38, 1.00, 0.00, 0.38, 1.00, 0.00, -0.38, -1.00, 0.00,
    0.38, -1.00, 0.00, 1.00, 0.00, -0.38, 1.00, 0.00, 0.38,
    -1.00, 0.00, -0.38, -1.00, 0.00, 0.38)
)
```

```
dodecahedron_face <- matrix(
  ncol = 3L, byrow = TRUE,
  c(1, 11, 2, 1, 2, 16, 1, 16, 15, 1, 15, 5, 1, 5, 20, 1, 20, 19,
    1, 19, 3, 1, 3, 12, 1, 12, 11, 2, 11, 12, 2, 12, 4, 2, 4, 17,
    2, 17, 18, 2, 18, 6, 2, 6, 16, 3, 13, 14, 3, 14, 4, 3, 4, 12,
    3, 19, 20, 3, 20, 7, 3, 7, 13, 4, 14, 8, 4, 8, 18, 4, 18, 17,
    5, 9, 10, 5, 10, 7, 5, 7, 20, 5, 15, 16, 5, 16, 6, 5, 6, 9,
    6, 18, 8, 6, 8, 10, 6, 10, 9, 7, 10, 8, 7, 8, 14, 7, 14, 13)
)

x0 <- as_ieegio_surface(dodecahedron_vert, faces = dodecahedron_face)

plot(x0)


# ---- merge by attributes -----------------------------------

# point-cloud but with vertex measurements
y1 <- as_ieegio_surface(
  dodecahedron_vert,
  measurements = data.frame(MyVariable = dodecahedron_vert[, 1]),
  transform = diag(c(2,1,0.5,1))
)

plot(y1)

# the geometry of `y1` will be discarded and only attributes
# (in this case, measurements:MyVariable) will be merged to `x`

z1 <- merge(x0, y1, merge_type = "attribute")

plot(z1)

# ---- merge by geometry ----------------------------------------

y2 <- as_ieegio_surface(
  dodecahedron_vert + 4, faces = dodecahedron_face,
  transform = diag(c(2, 1, 0.5, 1))
)

plot(y2)

# merge directly in model space: transform matrix of `y2` will be ignored
z2 <- merge(x0, y2, merge_type = "geometry", merge_space = "model")

plot(z2)

# merge x, y2 in the world space where transforms will be respected
z3 <- merge(x0, y2, merge_type = "geometry", merge_space = "world")

plot(z3)
```

`merge.ieegio_volume` *Merge* '`ieegio`' *volumes*

Description

Merge volume data into base image. The images must be static 3-dimensional volume data. Currently time-series or 4-dimensional data is not supported.

Usage

```
## S3 method for class 'ieegio_volume'
merge(x, y, ..., thresholds = 0, reshape = dim(x), na_fill = NA)
```

Arguments

`x`	base image to be merged
`y, ...`	images to be merged into `x`
`thresholds`	numerical threshold for `y, ...`, can be length of one or more, if images to overlay is more than one. The image values lower than the threshold will be trimmed out
`reshape`	output shape, default is the dimension of `x`; if changed, then the underlay will be sampled into the new shape
`na_fill`	how to handle missing values; default is `NA`; for compatibility, you might want to set to 0

Value

Merged volume with dimension `reshape`.

Examples

```
base_array <- array(0, c(15, 15, 15))
base_array[4:6, 4:6, 4:6] <- runif(27) * 255

# generate a 15x15x15 mask with 1mm spacing
vox2ras1 <- diag(1, 4)
vox2ras1[1:3, 4] <- -5
x <- as_ieegio_volume(base_array, vox2ras = vox2ras1)


# 15x15x15 mask with 0.5mmx1mmx1mm spacing but oblique to `x`
vox2ras2 <- matrix(
  nrow = 4, byrow = TRUE,
  c(
    2, 0.2, -0.1, -3,
    -0.2, 1, 0.4, -4,
```

```
    0.3, -0.1, 1, -1,
    0, 0, 0, 1
  )
)
# vox2ras2[1:3, 4] <- c(-3,-4, -1)
base_array[4:6, 4:6, 4:6] <- runif(27) * 255
y <- as_ieegio_volume(base_array, vox2ras = vox2ras2)


# merge y into x and up-sample mask to 64^3 volume
# set to higher number to get better interpolation quality
# Only voxels of y>0 will be merged to x
z <- merge(x, y, reshape = c(64, 64, 64), thresholds = 0)

# Visualize

oldpar <- par(mfrow = c(1, 3), mar = c(0, 0, 2, 0))

zoom <- 10
crosshair_ras <- c(0, 0, 0)
pixel_width <- 2

plot(x,
     zoom = zoom,
     position = crosshair_ras,
     pixel_width = pixel_width,
     main = "Original - underlay")
plot(y,
     zoom = zoom,
     position = crosshair_ras,
     pixel_width = pixel_width,
     main = "Original - overlay")
plot(
  z,
  zoom = zoom,
  position = crosshair_ras,
  pixel_width = pixel_width,
  main = "Merged & up-sampled")

# reset graphical state
par(oldpar)
```

`NWBHDF5IO` *Creates a* `NWBHDF5IO` *file container*

Description

Class definition for 'PyNWB' container; use `read_nwb` for construction function.

Active bindings

`opened` Whether the container is opened.

Methods

Public methods:

- `NWBHDF5IO$new()`
- `NWBHDF5IO$get_handler()`
- `NWBHDF5IO$open()`
- `NWBHDF5IO$close()`
- `NWBHDF5IO$close_linked_files()`
- `NWBHDF5IO$read()`
- `NWBHDF5IO$with()`
- `NWBHDF5IO$clone()`

Method `new()`**:** Initialize the class

Usage:

```
NWBHDF5IO$new(path = NULL, mode = c("r", "w", "r+", "a", "w-", "x"), ...)
```

Arguments:

`path` Path to a `'.nwb'` file

`mode` Mode for opening the file

`...` Other parameters passed to `nwb$NWBHDF5IO`

Method `get_handler()`**:** Get internal file handler. Please make sure you close the handler correctly.

Usage:

```
NWBHDF5IO$get_handler()
```

Returns: File handler, i.e. 'PyNWB' `NWBHDF5IO` instance.

Method `open()`**:** Open the connections, must be used together with `$close` method. For high-level method, see `$with`

Usage:

```
NWBHDF5IO$open()
```

Returns: container itself

Examples:

```
\dontrun{

# low-level method to open NWB file, for safer methods, see
# `container$with()` below

container$open()

data <- container$read()
```

```
# process data...

# Make sure the container is closed!
container$close()

}
```

Method `close()`**:** Close the connections (low-level method, see 'with' method below)

Usage:

`NWBHDF5IO$close(close_links = TRUE)`

Arguments:

`close_links` Whether to close all files linked to from this file; default is true

Returns: Nothing

Method `close_linked_files()`**:** Close all opened, linked-to files. `'MacOS'` and `'Linux'` automatically release the linked-to file after the linking file is closed, but `'Windows'` does not, which prevents the linked-to file from being deleted or truncated. Use this method to close all opened, linked-to files.

Usage:

`NWBHDF5IO$close_linked_files()`

Returns: Nothing

Method `read()`**:** Read the `'NWB'` file from the 'IO' source. Please use along with `'$with'` method

Usage:

`NWBHDF5IO$read()`

Returns: `'NWBFile'` container

Method `with()`**:** Safe wrapper for reading and handling `'NWB'` file. See class examples.

Usage:

`NWBHDF5IO$with(expr, quoted = FALSE, envir = parent.frame())`

Arguments:

`expr` R expression to evaluate

`quoted` Whether `expr` is quoted; default is false

`envir` environment for `expr` to evaluate; default is the parent frame (see `parent.frame`)

Returns: Whatever results generated by `expr`

Examples:

```
\dontrun{

container$with({
  data <- container$read()
  # process data
})

}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`NWBHDF5IO$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:

# Running this example requires a .nwb file

library(rnwb)
container <- NWBHDF5IO$new(path = file)
container$with({

  data <- container$read()
  electrode_table <- data$electrodes[convert = TRUE]

})

print(electrode_table)


## End(Not run)

## ------------------------------------------------
## Method `NWBHDF5IO$open`
## ------------------------------------------------

## Not run:

# low-level method to open NWB file, for safer methods, see
# `container$with()` below

container$open()

data <- container$read()

# process data...

# Make sure the container is closed!
container$close()


## End(Not run)

## ------------------------------------------------
## Method `NWBHDF5IO$with`
## ------------------------------------------------
```

```
## Not run:

container$with({
  data <- container$read()
  # process data
})


## End(Not run)
```

`plot.ieegio_surface` *Plot '3D' surface objects*

Description

Plot '3D' surface objects

Usage

```
## S3 method for class 'ieegio_surface'
plot(
  x,
  method = c("auto", "r3js", "rgl_basic", "rgl_full"),
  transform = 1L,
  name = "auto",
  vlim = NULL,
  col = c("black", "white"),
  slice_index = NULL,
  ...
)
```

Arguments

`x`	`'ieegio_surface'` object, see `read_surface`
`method`	plot method; `'basic'` for just rendering the surfaces; `'full'` for rendering with axes and title
`transform`	which transform to use, can be a 4-by-4 matrix; if the surface contains transform matrix, then this argument can be an integer index of the transform embedded, or the target (transformed) space name; print `names(x$transforms)` for choices
`name`	attribute and name used for colors, options can be `'color'` if the surface has color matrix; `c('annotations', varname)` for rendering colors from annotations with variable `varname`; `c('measurements', varname)` for rendering colors from measurements with variable `varname`; `'time_series'` for plotting time series slices; or `"flat"` for flat color; default is `'auto'`, which will plot the first available data. More details see 'Examples'.

`vlim`	when plotting with continuous data (`name` is measurements or time-series), the value limit used to generate color palette; default is `NULL`: the range of the values. This argument can be length of 1 (creating symmetric value range) or 2. If set, then values exceeding the range will be trimmed to the limit
`col`	color or colors to form the color palette when value data is continuous; when `name="flat"`, the last color will be used
`slice_index`	when plotting the `name="time_series"` data, the slice indices to plot; default is to select a maximum of 4 slices
`...`	ignored

Examples

```
library(ieegio)

# geometry
geom_file <- "gifti/GzipBase64/sujet01_Lwhite.surf.gii"

# measurements
shape_file <- "gifti/GzipBase64/sujet01_Lwhite.shape.gii"

# time series
ts_file <- "gifti/GzipBase64/fmri_sujet01_Lwhite_projection.time.gii"

if(ieegio_sample_data(geom_file, test = TRUE)) {

  geometry <- read_surface(ieegio_sample_data(geom_file))
  measurement <- read_surface(ieegio_sample_data(shape_file))
  time_series <- read_surface(ieegio_sample_data(ts_file))
  ts_demean <- apply(
    time_series$time_series$value,
    MARGIN = 1L,
    FUN = function(x) {
      x - mean(x)
    }
  )
  time_series$time_series$value <- t(ts_demean)

  # merge measurement & time_series into geometry (optional)
  merged <- merge(geometry, measurement, time_series)
  print(merged)

  # ---- plot method/style ------------------------------------
  plot(merged)

  # ---- plot data --------------------------------------------

  ## Measurements or annotations
```

```
  # the first column of `measurements`
  plot(merged, name = "measurements")

  # equivalent to
  plot(merged, name = list("measurements", 1L))

  # equivalent to
  measurement_names <- names(merged$measurements$data_table)
  plot(merged, name = list("measurements", measurement_names[[1]]))

  ## Time-series

  # automatically select 4 slices, trim the color palette
  # from -25 to 25
  plot(merged, name = "time_series", vlim = c(-25, 25),
       slice_index = 1L)

  plot(
    merged,
    name = "time_series",
    vlim = c(-25, 25),
    slice_index = 64,
    col = c("#053061", "#2166ac", "#4393c3",
            "#92c5de", "#d1e5f0", "#ffffff",
            "#fddbc7", "#f4a582", "#d6604d",
            "#b2182b", "#67001f")
  )


}
```

`plot.ieegio_volume` *Plot '3D' volume in anatomical slices*

Description

Plot '3D' volume in anatomical slices

Usage

```
## S3 method for class 'ieegio_volume'
plot(
  x,
  position = c(0, 0, 0),
  center_position = FALSE,
  which = c("coronal", "axial", "sagittal"),
  slice_index = 1L,
```

```
  transform = "vox2ras",
  zoom = 1,
  pixel_width = max(zoom/2, 1),
  col = c("black", "white"),
  alpha = NA,
  crosshair_gap = 4,
  crosshair_lty = 2,
  crosshair_col = "#00FF00A0",
  label_col = crosshair_col,
  continuous = TRUE,
  vlim = NULL,
  add = FALSE,
  main = "",
  axes = FALSE,
  background = col[[1]],
  foreground = col[[length(col)]],
  ...,
  .xdata = x$data
)
```

Arguments

`x`	`'ieegio_volume'` object; see `read_volume`
`position`	position in `'RAS'` (right-anterior-superior) coordinate system on which cross-hair should focus
`center_position`	whether to center canvas at `position`, default is `FALSE`
`which`	which slice to plot; choices are `"coronal"`, `"axial"`, and `"sagittal"`
`slice_index`	length of 1: if `x` has fourth dimension (e.g. 'fMRI'), then which slice index to draw
`transform`	which transform to apply, can be a 4-by-4 matrix, an integer or name indicating the matrix in `x$transforms`; this needs to be the transform matrix from voxel index to 'RAS' (right-anterior-superior coordinate system), often called `'xform'`, `'sform'`, `'qform'` in 'NIfTI' terms, or `'Norig'` in 'FreeSurfer'
`zoom`	zoom-in level
`pixel_width`	pixel size, ranging from `0.05` to `50`; default is the half of `zoom` or `1`, whichever is greater; the unit of `pixel_width` divided by `zoom` is milliliter
`col`	color palette for continuous `x` values
`alpha`	opacity value if the image is to be displayed with transparency
`crosshair_gap`	the cross-hair gap in milliliter
`crosshair_lty`	the cross-hair line type
`crosshair_col`	the cross-hair color; set to `NA` to hide
`label_col`	the color of anatomical axis labels (i.e. `"R"` for right, `"A"` for anterior, and `"S"` for superior); default is the same as `crosshair_col`
`continuous`	reserved

`vlim`	the range limit of the data; default is computed from range of `x$data`; data values exceeding the range will be trimmed
`add`	whether to add the plot to existing underlay; default is `FALSE`
`main, ...`	passed to `image`
`axes`	whether to draw axes; default is `FALSE`
`background, foreground`	background and foreground colors; default is the first and last elements of `col`
`.xdata`	default is `x$data`, used to speed up the calculation when multiple different angles are to be plotted

Examples

```
library(ieegio)

nifti_file <- "nifti/rnifti_example.nii.gz"
nifti_rgbfile <- "nifti/rnifti_example_rgb.nii.gz"

# Use
#   `ieegio_sample_data(nifti_file)`
# and
#   `ieegio_sample_data(nifti_rgbfile)`
# to download sample data


if(
  ieegio_sample_data(nifti_file, test = TRUE) &&
  ieegio_sample_data(nifti_rgbfile, test = TRUE)
) {

# ---- NIfTI examples ---------------------------------------------

underlay_path <- ieegio_sample_data(nifti_file)
overlay_path <- ieegio_sample_data(nifti_rgbfile)

# basic read
underlay <- read_volume(underlay_path)
overlay <- read_volume(overlay_path)

par(mfrow = c(1, 3), mar = c(0, 0, 3.1, 0))

ras_position <- c(50, -10, 15)

ras_str <- paste(sprintf("%.0f", ras_position), collapse = ",")

for(which in c("coronal", "axial", "sagittal")) {
  plot(x = underlay, position = ras_position, crosshair_gap = 10,
       crosshair_lty = 2, zoom = 3, which = which,
       main = sprintf("%s T1RAS=[%s]", which, ras_str))
  plot(x = overlay, position = ras_position,
       crosshair_gap = 10, label_col = NA,
       add = TRUE, alpha = 0.9, zoom = 5, which = which)
```

```
}


}
```

`pynwb_module` *Install* 'NWB' *via* 'pynwb'

Description

Install `'NWB'` via `'pynwb'`

Usage

```
install_pynwb(python_ver = "auto", verbose = TRUE)

pynwb_module(force = FALSE, error_if_missing = TRUE)
```

Arguments

`python_ver`	'Python' version, see `configure_conda`; default is `"auto"`, which is suggested
`verbose`	whether to print the installation messages
`force`	whether to force-reload the module
`error_if_missing`	whether to raise errors when the module fails to load; default is true

Value

A 'Python' module pynwb.

`read_bci2000` *Read 'BCI2000' data file*

Description

Read 'BCI2000' data file

Usage

```
read_bci2000(
  file,
  extract_path = getOption("ieegio.extract_path", NULL),
  header_only = FALSE,
  cache_ok = TRUE,
  verbose = TRUE
)
```

Arguments

`file`	file path to the data file
`extract_path`	location to where the extracted information is to be stored
`header_only`	whether to only load header data
`cache_ok`	whether existing cache should be reused; default is `TRUE`. This input can speed up reading large data files; set to `FALSE` to delete cache before importing.
`verbose`	whether to print processing messages; default is `TRUE`

Value

A cached object that is readily to be loaded to memory; see `SignalDataCache` for class definition.

Examples

```
if( ieegio_sample_data("bci2k.dat", test = TRUE) ) {
  file <- ieegio_sample_data("bci2k.dat")

  x <- read_bci2000(file)
  print(x)

  channel <- x$get_channel(1)

  plot(
    channel$time,
    channel$value,
    type = "l",
    main = channel$info$Label,
    xlab = "Time",
    ylab = channel$info$Unit
  )
}
```

`read_brainvis` *Read 'BrainVision' data*

Description

Read 'BrainVision' data

Usage

```
read_brainvis(
  file,
  extract_path = getOption("ieegio.extract_path", NULL),
  header_only = FALSE,
  cache_ok = TRUE,
  verbose = TRUE
)
```

Arguments

`file`	file path to the data file
`extract_path`	location to where the extracted information is to be stored
`header_only`	whether to only load header data
`cache_ok`	whether existing cache should be reused; default is `TRUE`. This input can speed up reading large data files; set to `FALSE` to delete cache before importing.
`verbose`	whether to print processing messages; default is `TRUE`

Value

A cached object that is readily to be loaded to memory; see `SignalDataCache` for class definition.

Examples

```
if( ieegio_sample_data("brainvis.dat", test = TRUE) ) {
  # ensure the header and marker files are downloaded as well
  ieegio_sample_data("brainvis.vhdr")
  ieegio_sample_data("brainvis.dat")
  file <- ieegio_sample_data("brainvis.vmrk")

  x <- read_brainvis(file)
  print(x)

  x$get_header()

  x$get_channel_table()

  x$get_annotations()

  channel <- x$get_channel(10)

  plot(
    channel$time,
    channel$value,
    type = "l",
    main = channel$info$Label,
    xlab = "Time",
    ylab = channel$info$Unit
  )
}
```

`read_edf` *Read 'EDF' or 'BDF' data file*

Description

Read 'EDF' or 'BDF' data file

Usage

```
read_edf(
  con,
  extract_path = getOption("ieegio.extract_path", NULL),
  header_only = FALSE,
  cache_ok = TRUE,
  begin = 0,
  end = Inf,
  convert = TRUE,
  verbose = TRUE
)
```

Arguments

`con`	file or connection to the data file
`extract_path`	location to where the extracted information is to be stored
`header_only`	whether to only load header data
`cache_ok`	whether existing cache should be reused; default is `TRUE`. This input can speed up reading large data files; set to `FALSE` to delete cache before importing.
`begin, end`	begin and end of the data to read
`convert`	whether to convert digital numbers to analog signals; default is `TRUE`
`verbose`	whether to print processing messages; default is `TRUE`

Value

A cached object that is readily to be loaded to memory; see `SignalDataCache` for class definition.

Examples

```
# ---- EDF/BDF(+) ---------------------------------------------------------

# Run `ieegio_sample_data("edfPlusD.edf")` to download sample data

# Tun example if the sample data exists
if(ieegio_sample_data("edfPlusD.edf", test = TRUE)) {
```

```
  edf_path <- ieegio_sample_data("edfPlusD.edf")

  data <- read_edf(edf_path)

  data$get_header()

  data$get_annotations()

  data$get_channel_table()

  channel <- data$get_channel(1)

  plot(
    channel$time,
    channel$value,
    type = "l",
    main = channel$info$Label,
    xlab = "Time",
    ylab = channel$info$Unit
  )

}
```

`read_nsx` *Read ('BlackRock') 'NEV' 'NSx' data*

Description

Read ('BlackRock') 'NEV' 'NSx' data

Usage

```
read_nsx(
  file,
  extract_path = getOption("ieegio.extract_path", NULL),
  header_only = FALSE,
  cache_ok = TRUE,
  include_waveform = FALSE,
  verbose = TRUE
)
```

Arguments

`file`	file path to the data file
`extract_path`	location to where the extracted information is to be stored
`header_only`	whether to only load header data

`cache_ok`	whether existing cache should be reused; default is `TRUE`. This input can speed up reading large data files; set to `FALSE` to delete cache before importing.
`include_waveform`	whether to include 'waveform' data (usually for online spike sorting); default is `FALSE`
`verbose`	whether to print processing messages; default is `TRUE`

Value

A cached object that is readily to be loaded to memory; see `SignalDataCache` for class definition.

`read_nwb` *Read 'NWB' format*

Description

Life cycle: experimental. Read "Neurodata Without Borders" ('NWB' format) file. Unlike other readers `read_nwb` returns low-level 'Python' class handler via `pynwb` module.

Usage

```
read_nwb(file, mode = c("r", "w", "r+", "a", "w-", "x"), ...)
```

Arguments

`file`	path to 'NWB' file
`mode`	file open mode; default is `'r'` (read-only)
`...`	passed to `NWBHDF5IO` initialize function

Value

A `NWBHDF5IO` instance

Examples

```
if(ieegio_sample_data("nwb_sample.nwb", test = TRUE)) {
  file <- ieegio_sample_data("nwb_sample.nwb")

  # Create NWBIO container
  container <- read_nwb(file)

  # Open connection
  container$open()

  # read meta data
  data <- container$read()
  data
```

```
  # get `test_timeseries` data
  ts_data <- data$get_acquisition("test_timeseries")
  ts_data

  # read timeseries data into memory
  ts_arr <- ts_data$data[]
  ts_arr

  # Convert Python array to R
  # using `rpymat::py_to_r(ts_arr)` or
  as.numeric(ts_arr)

  # Make sure you close the connection
  container$close()

}


# Requires setting up Python environment
# run `ieegio::install_pynwb()` to set up environment first

## Not run:


# Replicating tutorial
# https://pynwb.readthedocs.io/en/stable/tutorials/general/plot_file.html

library(rpymat)

# Load Python module
pynwb <- import("pynwb")
uuid <- import("uuid")
datetime <- import("datetime")
np <- import("numpy")
tz <- import("dateutil.tz")



# 2018L is 2018 as integer
session_start_time <- datetime$datetime(
  2018L, 4L, 25L, 2L, 30L, 3L,
  tzinfo=tz$gettz("US/Pacific"))

#  ---- Create NWB file object ------------------------
nwbfile <- pynwb$NWBFile(
  session_description="Mouse exploring a closed field",
  identifier=py_str(uuid$uuid4()),
  session_start_time=session_start_time,
  session_id="session_4321",
  experimenter=py_list(c("Baggins, Frodo")),
  lab="Bag End Laboratory",
  institution="University of Middle Earth at the Shire",
```

```
  experiment_description="Thank you Bilbo Baggins.",
  keywords=py_list(c("behavior", "exploration"))
)

# ---- Add subject -----------------------------------
subject <- pynwb$file$Subject(
  subject_id="001",
  age="P90D",
  description="mouse 5",
  species="Mus musculus",
  sex="M"
)

nwbfile$subject <- subject

nwbfile

# ---- Add TimeSeries -----------------------------------
data <- seq(100, 190, by = 10)
time_series_with_rate <- pynwb$TimeSeries(
  name="test_timeseries",
  description="an example time series",
  data=data,
  unit="m",
  starting_time=0.0,
  rate=1.0
)
time_series_with_rate

nwbfile$add_acquisition(time_series_with_rate)

# ---- New Spatial positions -----------------------------------
position_data <- cbind(
  seq(0, 10, length.out = 50),
  seq(0, 9, length.out = 50)
)
position_timestamps = seq(0, 49) / 200

spatial_series_obj = pynwb$behavior$SpatialSeries(
  name="SpatialSeries",
  description="(x,y) position in open field",
  data=position_data,
  timestamps=position_timestamps,
  reference_frame="(0,0) is bottom left corner",
)
spatial_series_obj

position_obj = pynwb$behavior$Position(
  spatial_series=spatial_series_obj)
position_obj

# ---- Behavior Processing Module ------------------------------
behavior_module <- nwbfile$create_processing_module(
```

```
  name="behavior", description="processed behavioral data"
)
behavior_module$add(position_obj)

nwbfile$processing$behavior

# omit some process

# ---- Write ---------------------------------------------------
f <- normalizePath(tempfile(fileext = ".nwb"),
                   winslash = "/",
                   mustWork = FALSE)
io <- pynwb$NWBHDF5IO(f, mode = "w")
io$write(nwbfile)
io$close()




## End(Not run)
```

`resample_volume` *Down-sample or super-sample volume*

Description

Using nearest-neighbor.

Usage

```
resample_volume(x, new_dim, na_fill = NA)
```

Arguments

`x`	image volume
`new_dim`	new dimension
`na_fill`	value to fill if missing

Value

A new volume with desired shape

Examples

```
# ---- Toy example ----------------------------

dm <- c(6, 6, 6)
arr <- array(seq_len(prod(dm)) + 0.5, dm)
orig <- as_ieegio_volume(
  arr, vox2ras = cbind(diag(1, nrow = 4, ncol = 3), c(-dm / 2, 1)))

# resample
downsampled <- resample_volume(orig, new_dim = c(3, 3, 3))
dim(downsampled)

# up-sample on coronal
upsampled <- resample_volume(orig, new_dim = c(20, 20, 24))
dim(upsampled)

par(mfrow = c(2, 2), mar = c(0, 0, 2.1, 0.1))
plot(orig, pixel_width = 0.5, zoom = 20, main = "Original")
plot(downsampled, pixel_width = 0.5, zoom = 20, main = "Down-sampled")
plot(upsampled, pixel_width = 0.5, zoom = 20, main = "Super-sampled")
plot(
  orig,
  main = "Overlay super-sample (diff)",
  col = c("black", "white"),
  pixel_width = 0.5, zoom = 20
)
plot(
  upsampled,
  add = TRUE,
  col = c("white", "black"),
  pixel_width = 0.5, zoom = 20,
  alpha = 0.5
)

# ---- Real example ---------------------------
nifti_file <- "brain.demosubject.nii.gz"

if( ieegio_sample_data(nifti_file, test = TRUE) ) {

  orig <- read_volume(ieegio_sample_data(nifti_file))
  dim(orig)

  # resample
  downsampled <- resample_volume(orig, new_dim = c(30, 30, 30))
  dim(downsampled)

  # up-sample on coronal
  upsampled <- resample_volume(orig, new_dim = c(300, 300, 64))
  dim(upsampled)

  par(mfrow = c(2, 2), mar = c(0, 0, 2.1, 0.1))
  plot(orig, main = "Original")
```

```
  plot(downsampled, main = "Down-sampled")
  plot(upsampled, main = "Super-sampled")
  plot(
    orig,
    main = "Overlay super-sample",
    col = c("black", "white"),
    zoom = 2,
    vlim = c(0, 255)
  )
  plot(
    upsampled,
    add = TRUE,
    col = c("white", "black"),
    zoom = 2,
    alpha = 0.5,
    vlim = c(0, 255)
  )


}
```

`SignalDataCache` *Class definition for signal cache*

Description

This class is an internal abstract class

Methods

Public methods:

- `SignalDataCache$get_header()`
- `SignalDataCache$get_annotations()`
- `SignalDataCache$get_channel_table()`
- `SignalDataCache$get_channel()`
- `SignalDataCache$delete()`

Method `get_header()`**:** Get header information, often small list object

Usage:

`SignalDataCache$get_header(...)`

Arguments:

`...` passed to child methods

Method `get_annotations()`**:** Get annotation information, often a large table

Usage:

`SignalDataCache$get_annotations(...)`

Arguments:

`...` passed to child methods

Method `get_channel_table():` Get channel table

Usage:

```
SignalDataCache$get_channel_table(...)
```

Arguments:

`...` passed to child methods

Method `get_channel():` Get channel data

Usage:

```
SignalDataCache$get_channel(x, ...)
```

Arguments:

`x` channel order or label

`...` passed to child methods

Returns: Channel signal with time-stamps inheriting class `'ieegio_get_channel'`

Method `delete():` Delete file cache

Usage:

```
SignalDataCache$delete(...)
```

Arguments:

`...` passed to child methods

Index