Package 'ieegio'

November 1, 2024

Title File IO for Intracranial Electroencephalography

Version 0.0.2

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, jsonlite, oro.nifti, R.matlab (>= 3.7.0), R6, readNSx (>= 0.0.5), rpyANTs (>= 0.0.3), stringr, utils, hdf5r, yaml

Suggests reticulate, rgl, rpymat (>= 0.1.7), RNifti (>= 1.7.0), xml2, knitr, 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 2024-10-31 23:20:02 UTC

Contents

ieegio_sample_data 2
imaging-surface 3

imaging-volume 5
io_h5_valid 8
io_read_h5 9
io_write_h5 10
LazyH5 12
low-level-read-write 14
NWBHDF5IO 18
plot.ieegio_surface 21
plot.ieegio_volume 24
pynwb_module 26
read_bci2000 26
read_brainvis 28
read_edf 29
read_nsx 30
read_nwb 31
SignalDataCache 34

Index **36**

`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"),
  ...
)

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, 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; default is the file name

Value

A surface object container

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("oro", "rnifti", "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, ...)
```

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.

Format

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

Value

imaging readers return `ieegio_volume` objects.

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

image(vol$data[,,128], asp = 1, axes = FALSE)

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

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

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

# ---- 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

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)
```

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("basic", "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, "basic")
  plot(merged, "full")

  # ---- 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))

  plot(
    merged,
    name = "time_series",
    vlim = c(-25, 25),
    slice_index = c(1, 17, 33, 49, 64, 80, 96, 112, 128),
    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),
  crosshair_gap = 4,
  crosshair_lty = 2,
  col = c("black", "white"),
  crosshair_col = "#00FF00A0",
  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`	cross-hair focused position
`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$transforms; this needs to be the transform matrix from voxel index to 'RAS' (right-anterior-superior coordinate system), often called 'xform', 'sform', 'qform' in 'NIfTI' terms, or 'Norig' in 'FreeSurfer'
zoom	zoom-in level
pixel_width	pixel size, ranging from 0.05 to 50; default is the half of zoom or 1, whichever is greater; the unit of pixel_width divided by zoom is milliliter
crosshair_gap	the cross-hair gap in milliliter
crosshair_lty	the cross-hair line type
col	color palette for continuous x values
crosshair_col	the cross-hair color; set to NA to hide
continuous	reserved
vlim	the range limit of the data; default is computed from range of x$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 <- "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)

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 = vol, position = ras_position, crosshair_gap = 10,
       crosshair_lty = 2, zoom = 3, which = which,
       main = sprintf("%s T1RAS=[%s]", which, ras_str))
}


}
```

`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)
```

`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

as_ANTsImage, *6*

configure_conda, *26*

data.frame, *15*
data.table, *15*
drop, *14*

fst, *15*

hdf5r-package, *9*

ieegio_sample_data, 2
image, *25*
imaging-surface, 3
imaging-volume, 5
install_pynwb *(*pynwb_module*)*, 26
io_h5_names *(*io_h5_valid*)*, 8
io_h5_valid, 8
io_read_fs *(*imaging-surface*)*, 3
io_read_fst *(*low-level-read-write*)*, 14
io_read_gii *(*imaging-surface*)*, 3
io_read_h5, 9, *11*
io_read_ini *(*low-level-read-write*)*, 14
io_read_json *(*low-level-read-write*)*, 14
io_read_mat *(*low-level-read-write*)*, 14
io_read_mgz *(*imaging-volume*)*, 5
io_read_nii *(*imaging-volume*)*, 5
io_read_yaml *(*low-level-read-write*)*, 14
io_write_fst *(*low-level-read-write*)*, 14
io_write_gii *(*imaging-surface*)*, 3
io_write_h5, *10*, 10
io_write_json *(*low-level-read-write*)*, 14
io_write_mat *(*low-level-read-write*)*, 14
io_write_mgz *(*imaging-volume*)*, 5
io_write_nii *(*imaging-volume*)*, 5
io_write_yaml *(*low-level-read-write*)*, 14

LazyH5, *9, 10*, 12
low-level-read-write, 14

mode, *13*

NWBHDF5IO, 18, *31*

plot.ieegio_surface, 21
plot.ieegio_volume, 24
pynwb_module, 26

read.fs.curv, *4*
read.fs.surface, *4*
read_bci2000, 26
read_brainvis, 28
read_edf, 29
read_nsx, 30
read_nwb, *18*, 31
read_surface, *22*
read_surface *(*imaging-surface*)*, 3
read_volume, *24*
read_volume *(*imaging-volume*)*, 5
readMat, *15*
readNIfTI, *6*
readNifti, *6*

SignalDataCache, *27–29*, *31*, 34

write_surface *(*imaging-surface*)*, 3
write_volume *(*imaging-volume*)*, 5