Package ‘raveio’

October 13, 2021

Type Package

Title File-System Toolbox for RAVE Project

Version 0.0.5

Language en-US

Description Includes multiple cross-platform read/write interfaces for 'RAVE' project. 'RAVE' stands for ‘R analysis and visualization of human intracranial electroencephalography data’. The whole project aims at providing powerful free-source package that analyze brain recordings from patients with electrodes placed on the cortical surface or inserted into the brain. 'raveio' as part of this project provides tools to read/write neurophysiology data from/to 'RAVE' file structure, as well as several popular formats including 'EDF(+)', 'Matlab', 'BIDS-iEEG', and 'HDF5', etc. Documentation and examples about 'RAVE' project are provided at <https://openwetware.org/wiki/RAVE>, and the paper by John F. Magnotti, Zhengjia Wang, Michael S. Beauchamp (2020) <doi:10.1016/j.neuroimage.2020.117341>; see 'citation('raveio')' for details.

BugReports https://github.com/beauchamplab/raveio/issues

URL https://beauchamplab.github.io/raveio/

License GPL-3

Encoding UTF-8

RoxygenNote 7.1.2

SystemRequirements little-endian platform

biocViews Infrastructure, DataImport

Depends R (>= 4.0.0)

Imports utils, data.table, parallel, edfReader (>= 1.2.1), dipsaus (>= 0.1.8), filearray (>= 0.1.1), fst (>= 0.9.2), glue, hdf5r (>= 1.3.4), ini (>= 0.3.1), jsonlite (>= 1.7.0), R.matlab (>= 3.6.2), R6, stringr (>= 1.4.0), yaml (>= 2.2.1), targets (>= 0.8.0), callr (>= 3.7.0), remotes, threeBrain (>= 0.2.3)

Suggests testthat, reticulate, knitr, rmarkdown, bs4Dash, clustermq, shiny, shinybusy, visNetwork, shinyWidgets, arrow
R topics documented:

as_rave_project ............................................. 3
as_rave_subject ............................................. 4
as_rave_unit .................................................. 4
cfgl ........................................................... 5
configure_knitr ............................................... 6
dir_create2 .................................................... 7
ECoGTensor ..................................................... 8
find_path ....................................................... 9
get_projects ................................................... 10
get_val2 ....................................................... 11
h5_names ...................................................... 12
h5_valid ....................................................... 12
is.blank ....................................................... 13
is.zerolenth .................................................. 14
is_valid_ish ................................................... 14
join_tensors ................................................... 15
LazyFST ......................................................... 16
LazyH5 ........................................................ 18
LFP_electrode ................................................... 21
load_bids_ieeg_header ....................................... 26
load_fst_or_h5 ............................................... 27
load_h5 ........................................................ 28
load_meta2 ..................................................... 29
load_yaml ...................................................... 30
rave-pipeline ................................................. 30
rave-raw-validation ......................................... 33
RAVEAbstractElectrode ....................................... 35
RAVEEpoch ..................................................... 38
raveio-option ................................................ 40
RAVEPreprocessSettings .................................... 41
RAVEProject .................................................... 44
RAVESubject ................................................... 46
rave_brain ..................................................... 48
rave_import .................................................... 49
read_brainvision-eeg ........................................ 51
read-write-fst ............................................... 53
read_csv_ieeg .................................................. 53
as_rave_project

Convert character to RAVEProject instance

Usage

as_rave_project(project, ...)

Arguments

  project character project name
  ... passed to other methods

Value

A RAVEProject instance

See Also

RAVEProject
as_rave_subject  Get RAVESubject instance from character

Description
Get RAVESubject instance from character

Usage
as_rave_subject(subject_id, strict = TRUE)

Arguments
subject_id  character in format "project/subject"
strict  whether to check if subject directories exist or not

Value
RAVESubject instance

See Also
RAVESubject

as_rave_unit  Convert numeric number into print-friendly format

Description
Convert numeric number into print-friendly format

Usage
as_rave_unit(x, unit, label = "")

Arguments
x  numeric or numeric vector
unit  the unit of x
label  prefix when printing x

Value
Still numeric, but print-friendly class
Examples

```r
sp <- as_rave_unit(1024, 'GB', 'Hard-disk space is ')
print(sp, digits = 0)

sp - 12

as.character(sp)

as.numeric(sp)

# Vectorize
sp <- as_rave_unit(c(500, 200), 'MB/s', c('Writing: ', 'Reading: '))
print(sp, digits = 0, collapse = '\n')
```

---

**catgl**  
*Print colored messages*

**Description**

Print colored messages

**Usage**

```r
catgl(..., .envir = parent.frame(), level = "DEBUG", .pal, .capture = FALSE)
```

**Arguments**

- `...`, `.envir` passed to `glue`
- `level` passed to `cat2`
- `.pal` see `pal` in `cat2`
- `.capture` logical, whether to capture message and return it without printing

**Details**

The level has order that sorted from low to high: "DEBUG", "DEFAULT", "INFO", "WARNING", "ERROR", "FATAL". Each different level will display different colors and icons before the message. You can suppress messages with certain levels by setting `raveio` options via `raveio_setopt('verbose_level', <level>)`. Messages with levels lower than the threshold will be muffled. See examples.

**Value**

The message as characters
configure_knitr

Configure `rmarkdown` files to build `RAVE` pipelines

Description

Allows building `RAVE` pipelines from `rmarkdown` files. Please use it in `rmarkdown` scripts only. Use `pipeline_create_template` to create an example.
Usage

configure_knitr(languages = c("R", "python"))

Arguments

languages one or more programming languages to support; options are 'R' and 'python'

Value

A function that is supposed to be called later that builds the pipeline scripts

---

Description

Force creating directory with checks

Usage

```r
dir_create2(x, showWarnings = FALSE, recursive = TRUE, check = TRUE, ...)
```

Arguments

- `x` path to create
- `showWarnings`, `recursive`, ...
  passed to `dir.create`
- `check` whether to check the directory after creation

Value

Normalized path

Examples

```r
path <- file.path(tempfile(), 'a', 'b', 'c')

# The following are equivalent
dir.create(path, showWarnings = FALSE, recursive = TRUE)
dir_create2(path)
```
ECoGTensor

'iEEG/ECoG' Tensor class inherit from Tensor

Description

Four-mode tensor (array) especially designed for 'iEEG/ECoG' data. The Dimension names are: Trial, Frequency, Time, and Electrode.

Super class

raveio::Tensor -> ECoGTensor

Methods

Public methods:

- ECoGTensor$flatten()
- ECoGTensor$new()

Method flatten(): converts tensor (array) to a table (data frame)

Usage:
ECoGTensor$flatten(include_index = TRUE, value_name = "value")

Arguments:
include_index logical, whether to include dimension names
value_name character, column name of the value

Returns: a data frame with the dimension names as index columns and value_name as value column

Method new(): constructor

Usage:
ECoGTensor$new(data, dim, dimnames, varnames, hybrid = FALSE, swap_file = temp_tensor_file(), temporary = TRUE, multi_files = FALSE, use_index = TRUE, ...

Arguments:
data array or vector
dim dimension of data, mush match with data
find_path

dimnames   list of dimension names, equal length as dim
varnames  names of dimnames, recommended names are: Trial, Frequency, Time, and Electrode
hybrid    whether to enable hybrid mode to reduce RAM usage
swap_file if hybrid mode, where to store the data; default stores in raveio_getopt('tensor_temp_path')
temporary whether to clean up the space when exiting R session
multi_files logical, whether to use multiple files instead of one giant file to store data
use_index logical, when multi_files is true, whether use index dimension as partition number

... further passed to Tensor constructor

Returns: an ECoGTensor instance

Author(s)
Zhengjia Wang

find_path  Try to find path along the root directory

Description
Try to find path under root directory even if the original path is missing; see examples.

Usage
find_path(path, root_dir, all = FALSE)

Arguments
path   file path
root_dir top directory of the search path
all    return all possible paths, default is false

Details
When file is missing, find_path concatenates the root directory and path combined to find the file. For example, if path is "a/b/c/d", the function first seek for existence of "a/b/c/d". If failed, then "b/c/d", and then "~/c/d" until reaching root directory. If all=TRUE, then all files/directories found along the search path will be returned

Value
The absolute path of file if exists, or NULL if missing/failed.
Examples

```r
root <- tempdir()

# ------ Case 1: basic use case -------
# Create a path in root
dir_create2(file.path(root, 'a'))

# find path even it's missing. The search path will be
# root/ins/cd/a - missing
# root/cd/a - missing
# root/a - exists!
find_path('ins/cd/a', root)

# ------ Case 2: priority -------
# Create two paths in root
dir_create2(file.path(root, 'cc/a'))
dir_create2(file.path(root, 'a'))

# If two paths exist, return the first path found
# root/ins/cc/a - missing
# root/cd/a - exists - returned
# root/a - exists, but ignored
find_path('ins/cc/a', root)

# ------ Case 3: find all -------
# Create two paths in root
dir_create2(file.path(root, 'cc/a'))
dir_create2(file.path(root, 'a'))

# If two paths exist, return the first path found
# root/ins/cc/a - missing
# root/cd/a - exists - returned
# root/a - exists - returned
find_path('ins/cc/a', root, all = TRUE)
```

---

**get_projects**

Get all possible projects in 'RAVE' directory

**Description**

Get all possible projects in 'RAVE' directory

**Usage**

`get_projects()`
**get_val2**

Get value or return default if invalid

---

### Description

Get value or return default if invalid

### Usage

```r
get_val2(x, key = NA, default = NULL, na = FALSE, min_len = 1L, ...)
```

### Arguments

- `x`: a list, or environment, or just any R object
- `key`: the name to obtain from `x`. If `NA`, then return `x`. Default is `NA`
- `default`: default value if `na, min_len, ...` passed to `is_valid_ish`

### Value

values of the keys or default is invalid

### Examples

```r
x <- list(a=1, b = NA, c = character(0))

# ------------------------ Basic usage ------------------------
get_val2(x)
get_val2(x, 'a', default = 'invalid')

# no key, returns x if x is valid
get_val2(x)
get_val2(x, 'b', default = 'invalid')

# get 'b', NA is not filtered out
get_val2(x, 'b', default = 'invalid')

# get 'b', NA is considered invalid
get_val2(x, 'b', default = 'invalid', na = TRUE)
```
# get 'c', length 0 is allowed
get_val2(x, 'c', default = 'invalid', min_len = 0)

# length 0 is forbidden
get_val2(x, 'c', default = 'invalid', min_len = 1)

h5_names

\textit{Returns all names contained in 'HDF5' file}

\textbf{Description}

Returns all names contained in 'HDF5' file

\textbf{Usage}

h5_names(file)

\textbf{Arguments}

file, \hspace{1em} 'HDF5' file path

\textbf{Value}

characters, data set names

h5_valid

\textit{Check whether a 'HDF5' file can be opened for read/write}

\textbf{Description}

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

\textbf{Usage}

h5_valid(file, mode = c("r", "w"), close_all = FALSE)

\textbf{Arguments}

file \hspace{1em} path to file

mode \hspace{1em} 'r' for read access and 'w' for write access

close_all \hspace{1em} whether to close all connections or just close current connection; default is false.
\hspace{1em} Set this to TRUE if you want to close all other connections to the file
is.blank

Check If Input Has Blank String

Description
Check If Input Has Blank String

Usage
is.blank(x)

Arguments
x input data: a vector or an array

Value
x == ""

Examples

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

# No data written to the file, hence invalid
h5_valid(f, 'r')

save_h5(x, f, 'dset')
h5_valid(f, 'w')

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

# Can read, but cannot write
h5_valid(f, 'r') # TRUE
h5_valid(f, 'w') # FALSE

# However, this can be reset via `close_all=TRUE`
h5_valid(f, 'r', close_all = TRUE)
h5_valid(f, 'w') # TRUE

# Now the connection is no longer valid
ptr
is.zerolenth  

Check If Input Has Zero Length

Usage

```r
is.zerolenth(x)
```

Arguments

- `x`: input data: a vector, list, or array

Value

whether `x` has zero length

is_valid_ish  

Check if data is close to “valid”

Usage

```r
is_valid_ish(
  x,
  min_len = 1,
  max_len = Inf,
  mode = NA,
  na = TRUE,
  blank = FALSE,
  all = FALSE
)
```

Arguments

- `x`: data to check
- `min_len`, `max_len`: minimal and maximum length
- `mode`: which storage mode (see `mode`) should `x` be considered valid. Default is NA: disabled.
na whether NA values considered invalid?
blank whether blank string considered invalid?
all if na or blank is true, whether all element of x being invalid will result in failure?

Value

logicals whether input x is valid

Examples

# length checks
is_valid_ish(NULL) # FALSE
is_valid_ish(integer(0)) # FALSE
is_valid_ish(integer(0), min_len = 0) # TRUE
is_valid_ish(1:10, max_len = 9) # FALSE

# mode check
is_valid_ish(1:10) # TRUE
is_valid_ish(1:10, mode = 'numeric') # TRUE
is_valid_ish(1:10, mode = 'character') # FALSE

# NA or blank checks
is_valid_ish(NA) # FALSE
is_valid_ish(c(1,2,NA), all = FALSE) # FALSE
is_valid_ish(c(1,2,NA), all = TRUE) # TRUE as not all elements are NA
is_valid_ish(c('1',''), all = FALSE) # TRUE
is_valid_ish(1:3, all = FALSE) # TRUE as 1:3 are not characters

join_tensors

Join Multiple Tensors into One Tensor

Description

Join Multiple Tensors into One Tensor

Usage

join_tensors(tensors, temporary = TRUE)

Arguments

tensors list of Tensor instances
temporary whether to garbage collect space when exiting R session
Details

Merges multiple tensors. Each tensor must share the same dimension with the last one dimension as 1, for example, 100x100x1. Join 3 tensors like this will result in a 100x100x3 tensor. This function is handy when each sub-tensors are generated separately. However, it does no validation test. Use with cautions.

Value

A new Tensor instance with the last dimension

Author(s)

Zhengjia Wang

Examples

tensor1 <- Tensor$new(data = 1:9, c(3,3,1), dimnames = list(A = 1:3, B = 1:3, C = 1), varnames = c('A', 'B', 'C'))
tensor2 <- Tensor$new(data = 10:18, c(3,3,1), dimnames = list(A = 1:3, B = 1:3, C = 2), varnames = c('A', 'B', 'C'))
merged <- join_tensors(list(tensor1, tensor2))
merged$get_data()

LazyFST

R6 Class to Load 'fst' Files

Description

provides hybrid data structure for 'fst' file

Methods

Public methods:
- LazyFST$open()
- LazyFST$close()
- LazyFST$save()
- LazyFST$new()
- LazyFST$get_dims()
- LazyFST$subset()

Method open(): to be compatible with LazyH5

Usage:
LazyFST$open(...)
LazyFST

Arguments:
... ignored

Returns: none

Method close(): close the connection

Usage:
LazyFST$close(..., .remove_file = FALSE)

Arguments:
... ignored
.remove_file whether to remove the file when garbage collected

Returns: none

Method save(): to be compatible with LazyH5

Usage:
LazyFST$save(...)

Arguments:
... ignored

Returns: none

Method new(): constructor

Usage:
LazyFST$new(file_path, transpose = FALSE, dims = NULL, ...)

Arguments:
file_path where the data is stored
transpose whether to load data transposed
dims data dimension, only support 1 or 2 dimensions
... ignored

Method get_dims(): get data dimension

Usage:
LazyFST$get_dims(...)

Arguments:
... ignored

Returns: vector, dimensions

Method subset(): subset data

Usage:
LazyFST$subset(i = NULL, j = NULL, ..., drop = TRUE)

Arguments:
i, j, ... index along each dimension
drop whether to apply drop the subset

Returns: subset of data
Author(s)

Zhengjia Wang

Examples

```r
if(interactive()){
    # Data to save, total 8 MB
    x <- matrix(rnorm(1000000), ncol = 100)

    # Save to local disk
    f <- tempfile()
    fst::write_fst(as.data.frame(x), path = f)

    # Load via LazyFST
    dat <- LazyFST$new(file_path = f, dims = c(10000, 100))

    # dat < 1 MB
    range(dat[] - x)

    # The reading of column is very fast
    system.time(dat[,100])

    # Reading rows might be slow
    system.time(dat[1,])
}
```

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

- LazyH5$new()
- LazyH5$save()
- LazyH5$open()
- LazyH5$close()
- LazyH5$subset()
- LazyH5$get_dims()

**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
c_type 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 _open()_ method

Method _open():_ open connection

_Useage:_
LazyH5$open(new_dataset = FALSE, obj, ...)

_Arguments:_
new_dataset only used when the internal pointer is closed, or to write the data
obj data array to save
... passed to _createDataSet_ in _hdf5r_ package

Method _close():_ close connection

_Useage:_
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

_Useage:_
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

_Retums:_ subset of data

Method _get_dims():_ get data dimension

_Useage:_
LazyH5$get_dims(stay_open = TRUE)

_Arguments:_
stay_open whether to leave the connection opened

_Returns:_ dimension of the array
Author(s)

Zhengjia Wang

Examples

```r
# Data to save
x <- array(rnorm(1000), c(10,10,10))

# Save to local disk
f <- tempfile()
save_h5(x, file = f, name = 'x', chunk = c(10,10,10), level = 0)

# Load via LazyFST
dat <- LazyH5$new(file_path = f, data_name = 'x', read_only = TRUE)

dat

# Check whether the data is identical
range(dat - x)

# Read a slice of the data
system.time(dat[,10,])
```

---

**LFP_electrode**

**Definition for 'LFP' electrodes**

**Description**

Definition for 'LFP' electrodes

**Super class**

`raveio::RAVEAbstractElectrode` -> LFP_electrode

**Public fields**

- `type` type of electrode

**Active bindings**

- `exists` whether electrode exists in subject
- `h5_fname` 'HDF5' file name
- `valid` whether current electrode is valid: subject exists and contains current electrode or reference; subject electrode type matches with current electrode type
- `raw_sample_rate` voltage sample rate
LFP_electrode

Methods

Public methods:

• LFP_electrode$set_reference()
• LFP_electrode$new()
• LFP_electrode$.load_noref_wavelet()
• LFP_electrode$.load_noref_voltage()
• LFP_electrode$.load_wavelet()
• LFP_electrode$.load_voltage()
• LFP_electrode$load_data()
• LFP_electrode$load_unreferenced_voltage()
• LFP_electrode$load_unreferenced_power()
• LFP_electrode$load_unreferenced_phase()
• LFP_electrode$reference_power()
• LFP_electrode$reference_phase()
• LFP_electrode$reference_voltage()
• LFP_electrode$clear_cache()
• LFP_electrode$clear_memory()
• LFP_electrode$clone()

Method set_reference(): set reference for current electrode

Usage:
LFP_electrode$set_reference(reference)

Arguments:
reference either NULL or LFP_electrode instance

Method new(): constructor

Usage:
LFP_electrode$new(subject, number, is_reference = FALSE)

Arguments:
subject, number, is_reference see constructor in RAVEAbstractElectrode

Method .load_noref_wavelet(): load non-referenced wavelet coefficients (internally used)

Usage:
LFP_electrode$.load_noref_wavelet(reload = FALSE)

Arguments:
Method .load_noref_voltage(): load non-referenced voltage (internally used)

Usage:
LFP_electrode$.load_noref_voltage(srate, reload = FALSE)

Arguments:
srate  voltage signal sample rate  
reload  whether to reload cache

Method .load_wavelet(): load referenced wavelet coefficients (internally used)

Usage:
LFP_electrode$.load_wavelet(type = c("power", "phase", "coef"), reload = FALSE)

Arguments:
type  type of data to load  
reload  whether to reload cache

Method .load_voltage(): load referenced voltage (internally used)

Usage:
LFP_electrode$.load_voltage(reload = FALSE)

Arguments:
reload  whether to reload cache

Method load_data(): method to load electrode data

Usage:
LFP_electrode$load_data(
  type = c("power", "phase", "voltage", "wavelet-coefficient")
)

Arguments:
type  data type such as "power", "phase", "voltage", "wavelet-coefficient".

Method load_unreferenced_voltage(): load voltage data, non-referenced

Usage:
LFP_electrode$load_unreferenced_voltage(block, persist = FALSE)

Arguments:
block  experiment block  
persist  whether to persist in the instance, default is false, however, if this function will be called multiple times, set it to true.

Returns:  voltage data before reference

Method load_unreferenced_power(): load power data, non-referenced

Usage:
LFP_electrode$load_unreferenced_power(block, persist = FALSE)

Arguments:
Method `load_unreferenced_phase()`: load phase data, non-referenced

Usage:
LFP_electrode$load_unreferenced_phase(block, persist = FALSE)

Arguments:
- `block`: experiment block
- `persist`: whether to persist in the instance, default is false, however, if this function will be called multiple times, set it to true.

Returns: phase data before reference

Method `reference_power()`: reference power for given block

Usage:
LFP_electrode$reference_power(block)

Arguments:
- `block`: character, experiment block

Returns: referenced power

Method `reference_phase()`: reference phase for given block

Usage:
LFP_electrode$reference_phase(block)

Arguments:
- `block`: character, experiment block

Returns: referenced phase

Method `reference_voltage()`: reference voltage for given block

Usage:
LFP_electrode$reference_voltage(block)

Arguments:
- `block`: character, experiment block

Returns: referenced voltage

Method `clear_cache()`: method to clear cache on hard drive

Usage:
LFP_electrode$clear_cache(...)

Arguments:
- `...`: ignored

Method `clear_memory()`: method to clear memory
Usage:
LFP_electrode$clear_memory(...)

Arguments:
... ignored

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

Usage:
LFP_electrode$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

Examples

## Not run:

# Download demo subject KC

# Electrode 14
e <- LFP_electrode$new(subject = 'demo/DemoSubject',
number = 14, is_reference = FALSE)

# Reference "ref_13-16,24"
ref <- LFP_electrode$new(subject = 'demo/DemoSubject',
number = "ref_13-16,24", is_reference = TRUE)

# ------ Reference ------
# By default there is no reference
e$reference_name # "noref"

# set reference
e$set_reference(reference = ref)
e$reference_name # "ref_13-16,24"

# Set epoch
e$set_epoch(epoch = 'auditory_onset')

# Now epoch power
power <- e$epoch_power(before_onset = 1, after_onset = 2)

# Trial x Frequency x Time x Electrodes
power

# Subset power
subset(power, Time ~ Time < 0, Electrode ~ Electrode == 14)

# clear memory in RAM and cache on hard disk
e$clear_cache()
e$clear_memory()
load_bids_ieeg_header

Read in description files from 'BIDS-iEEG' format

Description

Analyze file structures and import all json and tsv files. File specification can be found at [https://bids-specification.readthedocs.io/en/stable/](https://bids-specification.readthedocs.io/en/stable/), chapter "Modality specific files", section "Intracranial Electroencephalography" (doi: 10.1038/s4159701901057). Please note that this function has very limited support on BIDS format.

Usage

```r
load_bids_ieeg_header(bids_root, project_name, subject_code, folder = "ieeg")
```

Arguments

- `bids_root` : 'BIDS' root directory
- `project_name` : project folder name
- `subject_code` : subject code, do not include "sub-" prefix
- `folder` : folder name corresponding to 'iEEG' data. It's possible to analyze other folders. However, by default, the function is designed for 'ieeg' folder.

Value

A list containing the information below:

- `subject_code`: character, removed leading "sub-"
- `project_name`: character, project name
- `has_session`: whether session/block names are indicated by the file structure
- `session_names`: session/block names indicated by file structure. If missing, then session name will be "default"
- `paths`: a list containing path information
- `stimuli_path`: stimuli path, not used for now
- `sessions`: A named list containing meta information for each session/block. The names of the list is task name, and the items corresponding to the task contains events and channel information. Miscellaneous files are stored in "others" variable.
Examples

```r
# Download https://github.com/bids-standard/bids-examples/
# extract to directory ~/rave_data/bids_dir/

bids_root <- '~/rave_data/bids_dir/
project_name <- 'ieeg_visual'

if(dir.exists(bids_root) &
  dir.exists(file.path(bids_root, project_name, 'sub-01'))){
  header <- load_bids_ieeg_header(bids_root, project_name, '01')
  print(header)

  # sessions
  names(header$sessions)

  # electrodes
  head(header$sessions$'01'$spaces$unknown_space$table)

  # visual task channel settings
  head(header$sessions$'01'$tasks$'01-visual-01'$channels)

  # event table
  head(header$sessions$'01'$tasks$'01-visual-01'$channels)
}
```

### load_fst_or_h5

Function try to load 'fst' arrays, if not found, read 'HDF5' arrays

#### Description

Function try to load 'fst' arrays, if not found, read 'HDF5' arrays

#### Usage

```r
load_fst_or_h5(
  fst_path,
  h5_path,
  h5_name,
  fst_need_transpose = FALSE,
  fst_need_drop = FALSE,
  ram = FALSE
)
```
Arguments

- `fst_path` : 'fst' file cache path
- `h5_path` : alternative 'HDF5' file path
- `h5_name` : 'HDF5' data name
- `fst_needTranspose` : does 'fst' data need transpose?
- `fst_need_drop` : drop dimensions
- `ram` : whether to load to memory directly or perform lazy loading

Details

RAVE stores data with redundancy. One electrode data is usually saved with two copies in different formats: 'HDF5' and 'fst', where 'HDF5' is cross-platform and supported by multiple languages such as Matlab, Python, etc, while 'fst' format is supported by R only, with super high read/write speed. `load_fst_or_h5` checks whether the presence of 'fst' file, if failed, then it reads data from persistent 'HDF5' file.

Value

If 'fst' cache file exists, returns `LazyFST` object, otherwise returns `LazyH5` instance

---

**load_h5**

_Easy 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

```r
load_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

save_h5

Examples

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

# save x to file with name /group/dataset/1
save_h5(x, file, '/group/dataset/1', quiet = TRUE)

# read data
y <- load_h5(file, '/group/dataset/1', ram = TRUE)
class(y)  # array

z <- load_h5(file, '/group/dataset/1', ram = FALSE)
class(z)  # LazyH5

dim(z)
```

---

**load_meta2**  
*Load 'RAVE' subject meta data*

### Description

Load 'RAVE' subject meta data

### Usage

```r
load_meta2(meta_type, project_name, subject_code, subject_id, meta_name)
```

### Arguments

- **meta_type**
  - electrodes, epochs, time_points, frequencies, references ...
- **project_name**
  - project name
- **subject_code**
  - subject code
- **subject_id**
  - "project_name/subject_code"
- **meta_name**
  - only used if meta_type is epochs or references

### Value

A data frame of the specified meta type or NULL is no meta data is found.
load_yaml

A port to read_yaml

Description

For more examples, see save_yaml.

Usage

load_yaml(file, ..., map = NULL)

Arguments

file, ... passed to read_yaml
map fastmap2 instance or NULL

Value

A fastmap2. If map is provided then return map, otherwise return newly created one

See Also

cfascmap2, save_yaml, read_yaml, write_yaml,
```r
quick = TRUE,
env = parent.frame(),
pipe_dir = Sys.getenv("RAVE_PIPELINE", "."),
skip_names
)

pipeline_visualize(pipe_dir = Sys.getenv("RAVE_PIPELINE", "."))

pipeline_run(
  pipe_dir = Sys.getenv("RAVE_PIPELINE", "."),
  type = c("basic", "async", "vanilla", "custom"),
  envir = parent.frame(),
callr_function = NULL,
  ...
)

pipeline_progress(
  pipe_dir = Sys.getenv("RAVE_PIPELINE", "."),
  method = c("summary", "details", "custom"),
  func = targets::tar_progress_summary
)

pipeline_fork(
  src = Sys.getenv("RAVE_PIPELINE", "."),
  dest = tempfile(pattern = "rave_pipeline_."),
  filter_pattern = "\.\.(R|yaml|txt|csv|fst|conf)$",
  activate = FALSE
)

pipeline_build(pipe_dir = Sys.getenv("RAVE_PIPELINE", "."))

pipeline_read(
  var_names,
  pipe_dir = Sys.getenv("RAVE_PIPELINE", "."),
  branches = NULL,
  ifnotfound = NULL
)

pipeline_vartable(
  pipe_dir = Sys.getenv("RAVE_PIPELINE", "."),
  targets_only = TRUE,
  complete_only = FALSE,
  ...
)

pipeline_hasname(var_names, pipe_dir = Sys.getenv("RAVE_PIPELINE", "."))

pipeline_watch(
```
pipe_dir = Sys.getenv("RAVE_PIPELINE", "."),
targets_only = TRUE,
...)

pipeline_create_template(
    root_path,
    pipeline_name,
    overwrite = FALSE,
    activate = TRUE,
    template_type = c("rmd", "r")
)

pipeline_create_subject_pipeline(
    subject,
    pipeline_name,
    overwrite = FALSE,
    activate = TRUE,
    template_type = c("rmd", "r")
)

pipeline_description(file)

Arguments

root_path the root directory for pipeline templates
name, pipeline_name the pipeline name to create; usually also the folder name within subject's pipeline path
... other parameters, targets, etc.
quick whether to skip finished targets to save time
env, envir environment to execute the pipeline
pipe_dir where the pipeline directory is; can be set via system environment `Sys.setenv("RAVE_PIPELINE"=...)`
skip_names hint of target names to fast skip provided they are up-to-date; only used when quick=TRUE. If missing, then skip_names will be automatically determined
type how the pipeline should be executed; current choices are "basic" to run in the main session; "async" to run in a separate session without blocking the main session; "vanilla" to run in a separate session and wait for the results; or "custom" to run customized scheduler callr_function
callr_function function to customized when type="custom"
method how the progress should be presented; choices are "summary", "details", "custom". If custom method is chosen, then func will be called
func function to call when reading customized pipeline progress; default is `tar_progress_summary`
src, dest pipeline folder to copy the pipeline script from and to
filter_pattern file name patterns used to filter the scripts to avoid copying data files; default is ".(R|yaml|txt|csv|fst|conf)$"
activate  
whether to activate the new pipeline folder from dest; default is false

var_names  
variable name to fetch or to check

branches  
branch to read from; see \texttt{tar\_read}

ifnotfound  
default values to return if variable is not found

targets\_only  
whether to return the variable table for targets only; default is true

complete\_only  
whether only to show completed and up-to-date target variables; default is false

overwrite  
whether to overwrite existing pipeline; default is false so users can double-check; if true, then existing pipeline, including the data will be erased

template\_type  
which template type to create; choices are ‘r’ or ‘rmd’

subject  
character indicating valid ’RAVE’ subject ID, or \texttt{RAVESubject} instance

file  
path to the ’DESCRIPTION’ file under the pipeline folder, or pipeline collection folder that contains the pipeline information, structures, dependencies, etc.

\begin{verbatim}

tiff  
interpretation of image file

tiffadjust  
control adjustment to tiff file

tiffadjuster  
method of interpreting tiff file

tiffnote  
comment about tiff file

\end{verbatim}

---

\textbf{Description}

Validate subjects and returns whether the subject can be imported into ’rave’

\textbf{Usage}

\begin{verbatim}
validate_raw_file(
  subject_code,
  blocks,
  electrodes,
  format,
  data_type = c("continuous"),
  ...
)
\end{verbatim}

\textbf{Arguments}

| subject_code | subject code, direct folder under ’rave’ raw data path |
| blocks       | block character, direct folder under subject folder. For raw files following ’BIDS’ convention, see details |
| electrodes   | electrodes to verify |
| format       | integer or character. For characters, run names(IMPORT\_FORMATS) |
| data_type    | currently only support continuous type of signals |
| ...          | other parameters used if validating ’BIDS’ format; see details |
Format

An object of class list of length 6.

Details

Six types of raw file structures are supported. They can be basically classified into two categories: 'rave' native raw structure and 'BIDS-iEEG' structure.

In 'rave' native structure, subject folders are stored within the root directory, which can be obtained via raveio_getopt('raw_data_dir'). Subject directory is the subject code. Inside of subject folder are block files. In 'rave', term 'block' is the combination of session, task, and run. Within each block, there should be 'iEEG' data files.

In 'BIDS-iEEG' format, the root directory can be obtained via raveio_getopt('bids_data_dir'). 'BIDS' root folder contains project folders. This is unlike 'rave' native raw data format. Subject folders are stored within the project directories. The subject folders start with 'sub-'. Within subject folder, there are session folders with prefix 'ses-'. Session folders are optional. 'iEEG' data is stored in 'ieeg' folder under the session/subject folder. 'ieeg' folder should contain at least

```
electrodes.tsv sub-<label>*.electrodes.tsv
```

'iEEG' description sub-<label>*_task-<label>_run-<index>_ieeg.json

'iEEG' data file sub-<label>*_task-<label>_run-<index>_ieeg.<ext>, in current 'rave', only extensions '.vhdr+.eeg/.dat' ('BrainVision') or 'EDF' (or plus) are supported.

When format is 'BIDS', project_name must be specified.

The following formats are supported:

'.mat/.h5 file per electrode per block' 'rave' native raw format, each block folder contains multiple 'Matlab' or 'HDF5' files. Each file corresponds to a channel/electrode. File names should follow 'xxx001.mat' or 'xxx001.h5'. The numbers before the extension are channel numbers.

'Single .mat/.h5 file per block' 'rave' native raw format, each block folder contains only one 'Matlab' or 'HDF5' file. The file name can be arbitrary, but extension must be either '.mat' or '.h5'. Within the file there should be a matrix containing all the data. The short dimension of the matrix will be channels, and larger side of the dimension corresponds to the time points.

'Single EDF(+) file per block' 'rave' native raw format, each block folder contains only one '.edf' file.

'Single BrainVision file (.vhdr+.eeg, .vhdr+.dat) per block' 'rave' native raw format, each block folder contains only two files. The first file is header '.vhdr' file. It contains all meta information. The second is either '.eeg' or '.dat' file containing the body, i.e. signal entries.

'BIDS & EDF(+)' 'BIDS' format. The data file should have '.edf' extension

'BIDS & BrainVision (.vhdr+.eeg, .vhdr+.dat)' 'BIDS' format. The data file should have '.vhdr+' or '.eeg/.dat' extensions
RAVEAbstarctElectrode

**Value**

logical true or false whether the directory is valid. Attributes containing error reasons or snapshot of the data. The attributes might be:

- **snapshot**: description of data found if passing the validation
- **valid_run_names**: For ’BIDS’ format, valid session+task+run name if passing the validation
- **reason**: named list where the names are the reason why validation fails and values are corresponding sessions or electrodes or both.

---

**RAVEAbstarctElectrode**  
*Abstract definition of electrode class in RAVE*

---

**Description**

This class is not intended for direct use. Please create new child classes and implement some key methods.

**Public fields**

- **type**: type of electrode
- **subject**: subject instance (*RAVESubject*)
- **number**: integer stands for electrode number or reference ID
- **reference**: reference electrode, either NULL for no reference or an electrode instance inherits RAVEAbstarctElectrode
- **epoch**: a *RAVEEpoch* instance
- **is_reference**: whether this instance is a reference electrode

**Active bindings**

- **exists**: whether electrode exists in subject
- **preprocess_file**: path to preprocess ’HDF5’ file
- **power_file**: path to power ’HDF5’ file
- **phase_file**: path to phase ’HDF5’ file
- **voltage_file**: path to voltage ’HDF5’ file
- **reference_name**: reference electrode name
- **epoch_name**: current epoch name
- **cache_root**: run-time cache path; NA if epoch or trial intervals are missing
- **trial_intervals**: trial intervals relative to epoch onset
Methods

Public methods:

- `RAVEAbstractElectrode$new()`
- `RAVEAbstractElectrode$.set_reference()`
- `RAVEAbstractElectrode$.set_epoch()`
- `RAVEAbstractElectrode$clear_cache()`
- `RAVEAbstractElectrode$clear_memory()`
- `RAVEAbstractElectrode$load_data()`
- `RAVEAbstractElectrode$clone()`

**Method `new()`**: constructor

*Usage:*

`RAVEAbstractElectrode$new(subject, number, is_reference = FALSE)`

*Arguments:*

- `subject` character or `RAVESubject` instance
- `number` current electrode number or reference ID
- `is_reference` whether instance is a reference

**Method `.set_reference()`**: set reference for instance

*Usage:*

`RAVEAbstractElectrode$.set_reference(reference, type)`

*Arguments:*

- `reference` NULL or `RAVEAbstractElectrode` instance
- `type` reference electrode type, default is the same as current instance

**Method `set_epoch()`**: set epoch instance for the electrode

*Usage:*

`RAVEAbstractElectrode$set_epoch(epoch)`

*Arguments:*

- `epoch` characters or `RAVEEpoch` instance. For characters, make sure "epoch_<name>.csv" is in meta folder.

**Method `clear_cache()`**: method to clear cache on hard drive

*Usage:*

`RAVEAbstractElectrode$clear_cache(...)`

*Arguments:*

... implemented by child instances

**Method `clear_memory()`**: method to clear memory

*Usage:*

`RAVEAbstractElectrode$clear_memory(...)`

*Arguments:*
... implemented by child instances

**Method** `load_data()`: method to load electrode data

*Usage:*

`RAVEAbstractElectrode$load_data(type)`

*Arguments:*

type data type such as "power", "phase", "voltage", "wavelet-coefficient", or others depending on child class implementations

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

*Usage:*

`RAVEAbstractElectrode$clone(deep = FALSE)`

*Arguments:*

depth Whether to make a deep clone.

**Examples**

```r
## Not run:

# To run this example, please download demo subject (~700 MB) from
# https://github.com/beauchamplab/rave/releases/tag/v0.1.9-beta

generator <- RAVEAbstractElectrode

e <- generator$new("demo/DemoSubject", number = 14)

# set epoch
e$subject$epoch_names
e$set_epoch("auditory_onset")
head(e$epoch$table)

# set epoch range (-1 to 2 seconds relative to onset)
e$trial_intervals <- c(-1,2)
# or to set multiple ranges
e$trial_intervals <- list(c(-2,-1), c(0, 2))

# set reference
e$subject$reference_names
reference_table <- e$subject$meta_data(
    meta_type = "reference",
    meta_name = "default")
ref_name <- subset(reference_table, Electrode == 14)["Reference"]

# the reference is CAR type, mean of electrode 13-16,24
ref_name

# load & set reference
ref <- generator$new(e$subject, ref_name, is_reference = TRUE)
e$.set_reference(ref, e$type)
```
RAVEEpoch

**Definition for epoch class**

---

**Description**

Trial epoch, contains the following information: Block experiment block/session string; Time trial onset within that block; Trial trial number; Condition trial condition. Other optional columns are Event_xxx (starts with "Event"). See [https://openwetware.org/wiki/RAVE:Epoching](https://openwetware.org/wiki/RAVE:Epoching) or more details.

**Public fields**

- name  epoch name, character
- subject  RAVESubject instance
- data  a list of trial information, internally used
- table  trial epoch table
- .columns  epoch column names, internally used

**Active bindings**

- columns  columns of trial table
- n_trials  total number of trials
- trials  trial numbers

**Methods**

- **Public methods:**
  - RAVEEpoch$new()  
  - RAVEEpoch$trial_at()  
  - RAVEEpoch$update_table()  
  - RAVEEpoch$set_trial()  
  - RAVEEpoch$clone()  

  **Method new():** constructor  
  
  **Usage:**  
  RAVEEpoch$new(subject, name)  

  **Arguments:**  
  subject  RAVESubject instance or character  
  name  character, make sure "epoch_<name>.csv" is in meta folder
Method trial_at(): get ith trial

Usage:
RAVEEpoch$trial_at(i, df = TRUE)

Arguments:
i  trial number
df  whether to return as data frame or a list

Method update_table(): manually update table field

Usage:
RAVEEpoch$update_table()

Returns: self$table

Method set_trial(): set one trial

Usage:
RAVEEpoch$set_trial(Block, Time, Trial, Condition, ...)

Arguments:
Block  block string
Time  time in second
Trial  positive integer, trial number
Condition  character, trial condition
... other key-value pairs corresponding to other optional columns

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

Usage:
RAVEEpoch$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.

Examples

# Please download DemoSubject ~700MB from
# https://github.com/beauchamplab/rave/releases/tag/v0.1.9-beta

## Not run:
# Load meta/epoch_auditory_onset.csv from subject demo/DemoSubject
ePOCH <- RAVEEpoch$new(subject = 'demo/DemoSubject',
                      name = 'auditory_onset')

# first several trials
head(P$ tabela)

# query specific trial
old_trial1 <- P$trial_at(1)
# Create new trial or change existing trial
epoch$set_trial(Block = '008', Time = 10, 
    Trial = 1, Condition = 'AknownVmeant')
new_trial1 <- epoch$trial_at(1)

# Compare new and old trial 1
rbind(old_trial1, new_trial1)

# To get updated trial table, must update first
epoch$update_table()
head(epoch$table)

## End(Not run)

---

### raveio-option

#### Set/Get 'raveio' option

**Description**

Persist settings on local configuration file

**Usage**

- `raveio_setopt(key, value, .save = TRUE)`
- `raveio_resetopt(all = FALSE)`
- `raveio_getopt(key, default = NA, temp = TRUE)`
- `raveio_confpath(cfile = "settings.yaml")`

**Arguments**

- `key`: character, option name
- `value`: character or logical of length 1, option value
- `.save`: whether to save to local drive, internally used to temporary change option. Not recommended to use it directly.
- `all`: whether to reset all non-default keys
- `default`: is key not found, return default value
- `temp`: when saving, whether the key-value pair should be considered temporary, a temporary settings will be ignored when saving; when getting options, setting `temp` to false will reveal the actual settings.
- `cfile`: file name in configuration path
Details

`raveio_setopt` stores key-value pair in local path. The values are persistent and shared across multiple sessions. There are some read-only keys such as "session_string". Trying to set those keys will result in error.

`raveio_getopt` returns value corresponding to the keys. If key is missing, the whole option will be returned.

If set all=TRUE, `raveio_resetopt` resets all keys including non-standard ones. However "session_string" will never reset.

Value

`raveio_setopt` returns modified value; `raveio_resetopt` returns current settings as a list; `raveio_confpath` returns absolute path for the settings file; `raveio_getopt` returns the settings value to the given key, or default if not found.

See Also

R_user_dir

---

RAVEPreprocessSettings

Defines preprocess configurations

Description

R6 class definition

Public fields

current_version  current configuration setting version
path  settings file path
backup_path  alternative back up path for redundancy checks
data  list of raw configurations, internally used only
subject  RAVESubject instance
read_only  whether the configuration should be read-only, not yet implemented

Active bindings

version  configure version of currently stored files
old_version  whether settings file is old format
blocks  experiment blocks
electrodes  electrode numbers
sample_rates  voltage data sample rate
RAVEPreprocessSettings

notch_filtered whether electrodes are notch filtered
has_wavelet whether each electrode has wavelet transforms
data_imported whether electrodes are imported
data_locked whether electrode, blocks and sample rate are locked? usually when an electrode is
imported into 'rave', that electrode is locked
electrode_locked whether electrode is imported and locked
wavelet_params wavelet parameters
notch_params Notch filter parameters
electrode_types electrode signal types
@freeze_blocks whether to free block, internally used
@freeze_lfp_ecog whether to freeze electrodes that record 'LFP' and 'ECoG' signals, internally used
@lfp_ecog_sample_rate 'LFP' and 'ECoG' sample rates, internally used
all_blocks characters, all possible blocks even not included in some projects
raw_path raw data path

Methods

Public methods:

- RAVEPreprocessSettings$new()
- RAVEPreprocessSettings$valid()
- RAVEPreprocessSettings$has_raw()
- RAVEPreprocessSettings$set_blocks()
- RAVEPreprocessSettings$set_electrodes()
- RAVEPreprocessSettings$set_sample_rates()
- RAVEPreprocessSettings$migrate()
- RAVEPreprocessSettings$electrode_info()
- RAVEPreprocessSettings$save()

Method new(): constructor

Usage:
RAVEPreprocessSettings$new(subject, read_only = TRUE)

Arguments:
subject character or RAVESubject instance
read_only whether subject should be read-only (not yet implemented)

Method valid(): whether configuration is valid or not

Usage:
RAVEPreprocessSettings$valid()

Method has_raw(): whether raw data folder exists

Usage:
RAVEPreprocessSettings$has_raw()

**Method** set_blocks(): set blocks

*Usage:*
RAVEPreprocessSettings$set_blocks(blocks, force = FALSE)

*Arguments:*
blocks character, combination of session task and run
force whether to ignore checking. Only used when data structure is not native, for example, 'BIDS' format

**Method** set_electrodes(): set electrodes

*Usage:*
RAVEPreprocessSettings$set_electrodes(
  electrodes,
  type = c("LFP", "ECoG", "Spike", "EEG"),
  add = FALSE
)

*Arguments:*
electrodes integer vectors
type type of electrodes, options are LFP, ECoG, and Spike.
add whether to add to current settings

**Method** set_sample_rates(): set sample frequency

*Usage:*
RAVEPreprocessSettings$set_sample_rates(
  srate,
  type = c("LFP", "ECoG", "Spike", "EEG")
)

*Arguments:*
srate sample rate, must be positive number
type electrode type to set sample rate. In 'rave', all electrodes with the same type must have the same sample rate.

**Method** migrate(): convert old format to new formats

*Usage:*
RAVEPreprocessSettings$migrate(force = FALSE)

*Arguments:*
force whether to force migrate and save settings

**Method** electrode_info(): get electrode information

*Usage:*
RAVEPreprocessSettings$electrode_info(electrode)

*Arguments:*
electrode integer
Returns: list of electrode type, number, etc.

**Method** `save()`: save settings to hard disk

**Usage:**
RAVEPreprocessSettings$save()

**Examples**

```r
# The following example require downloading demo subject (~700 MB) from
# https://github.com/beauchamplab/rave/releases/tag/v0.1.9-beta

## Not run:
conf <- RAVEPreprocessSettings$new(subject = 'demo/DemoSubject')
conf$blocks # "008" "010" "011" "012"

conf$electrodes # 5 electrodes

# Electrode 14 information
conf$electrode_info(electrode = 14)

conf$data_imported # All 5 electrodes are imported

conf$data_locked # Whether block, sample rates should be locked

## End(Not run)
```

---

**RAVEProject**

Definition for 'RAVE' project class

**Description**

Definition for 'RAVE' project class

**Active bindings**

- path project folder, absolute path
- name project name, character
- pipeline_path path to pipeline scripts under project’s folder
Methods

Public methods:
- `RAVEProject$print()`
- `RAVEProject$new()`
- `RAVEProject$subjects()`
- `RAVEProject$has_subject()`
- `RAVEProject$group_path()`
- `RAVEProject$clone()`

Method `print()`: override print method

Usage:
`RAVEProject$print()`

Arguments:
... ignored

Method `new()`: constructor

Usage:
`RAVEProject$new(project_name, strict = TRUE)`

Arguments:
- `project_name` character
- `strict` whether to check project path

Method `subjects()`: get all imported subjects within project

Usage:
`RAVEProject$subjects()`

Returns: character vector

Method `has_subject()`: whether a specific subject exists in this project

Usage:
`RAVEProject$has_subject(subject_code)`

Arguments:
- `subject_code` character, subject name

Returns: true or false whether subject is in the project

Method `group_path()`: get group data path for 'rave' module

Usage:
`RAVEProject$group_path(module_id, must_work = FALSE)`

Arguments:
- `module_id` character, 'rave' module ID
- `must_work` whether the directory must exist; if not exists, should a new one be created?

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

Usage:
`RAVEProject$clone(deep = FALSE)`

Arguments:
- `deep` Whether to make a deep clone.
RAVESubject  

**Defines 'RAVE' subject class**

---

**Description**

R6 class definition

**Active bindings**

- `project` project instance of current subject; see RAVEProject
- `project_name` character string of project name
- `subject_code` character string of subject code
- `subject_id` subject ID: "project/subject"
- `path` subject root path
- `rave_path` 'rave' directory under subject root path
- `meta_path` meta data directory for current subject
- `freesurfer_path` 'FreeSurfer' directory for current subject. If no path exists, values will be NA
- `preprocess_path` preprocess directory under subject 'rave' path
- `data_path` data directory under subject 'rave' path
- `cache_path` path to 'FST' copies under subject 'data' path
- `pipeline_path` path to pipeline scripts under subject's folder
- `epoch_names` possible epoch names
- `reference_names` possible reference names
- `reference_path` reference path under 'rave' folder
- `preprocess_settings` preprocess instance; see RAVEPreprocessSettings
- `blocks` subject experiment blocks in current project
- `electrodes` all electrodes, no matter excluded or not
- `raw_sample_rates` voltage sample rate
- `power_sample_rate` power spectrum sample rate
- `has_wavelet` whether electrodes have wavelet transforms
- `notch_filtered` whether electrodes are Notch-filtered
- `electrode_types` electrode signal types

**Methods**

**Public methods:**
- `RAVESubject$print()`
- `RAVESubject$new()`
- `RAVESubject$meta_data()`
RAVESubject

- RAVESubject$valid_electrodes()
- RAVESubject$initialize_paths()
- RAVESubject$clone()

Method `print()`: override print method

Usage:
RAVESubject$print(...)

Arguments:
... ignored

Method `new()`: constructor

Usage:
RAVESubject$new(project_name, subject_code = NULL, strict = TRUE)

Arguments:
project_name character project name
subject_code character subject code
strict whether to check if subject folders exist

Method `meta_data()`: get subject meta data located in "meta/" folder

Usage:
RAVESubject$meta_data(
  meta_type = c("electrodes", "frequencies", "time_points", "epoch", "references"),
  meta_name = "default"
)

Arguments:
meta_type choices are 'electrodes', 'frequencies', 'time_points', 'epoch', 'references'
meta_name if meta_type='epoch', read in 'epoch_<meta_name>.csv';
            if meta_type='references', read in 'reference_<meta_name>.csv'.

Returns: data frame

Method `valid_electrodes()`: get valid electrode numbers

Usage:
RAVESubject$valid_electrodes(reference_name, refresh = FALSE)

Arguments:
reference_name character, reference name, see meta_name in self$meta_data or load_meta2
              when meta_type is 'reference'
refresh whether to reload reference table before obtaining data, default is false

Returns: integer vector of valid electrodes

Method `initialize_paths()`: create subject's directories on hard disk

Usage:
RAVESubject$initialize_paths(include_freesurfer = TRUE)

Arguments:
include_freesurfer  whether to create 'FreeSurfer' path

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

Usage:
RAVESubject$clone(deep = FALSE)

Arguments:
depth Whether to make a deep clone.

See Also
load_meta2

rave_brain Load 'FreeSurfer' or 'AFNI/SUMA' brain from 'RAVE'

Description
Create 3D visualization of the brain and visualize with modern web browsers

Usage
rave_brain(
  subject,
  surfaces = "pial",
  use_141 = TRUE,
  recache = FALSE,
  clean_before_cache = FALSE,
  compute_template = FALSE,
  usetemplateifmissing = FALSE
)

Arguments
subject character, list, or RAVESubject instance; for list or other objects, make sure subject$subject_id is a valid 'RAVE' subject 'ID'
surfaces one or more brain surface types from "pial", "white", "smoothwm", "pial-outer-smoothed", etc.; check freesurfer_brain2
use_141 whether to use 'AFNI/SUMA' standard 141 brain
recache whether to re-calculate cache; only should be used when the original 'FreeSurfer' or 'AFNI/SUMA' files are changed; such as new files are added
clean_before_cache whether to clean the original cache before recache; only set it to be true if original cached files are corrupted
compute_template whether to compute template mappings; useful when template mapping with multiple subjects are needed
usetemplateifmissing

whether to use template brain when the subject brain files are missing. If set to true, then a template (usually 'N27') brain will be displayed as an alternative solution, and electrodes will be rendered according to their 'MNI305' coordinates, or 'VertexNumber' if given.

Value

A 'threeBrain' instance if brain is found or usetemplateifmissing is set to true; otherwise returns NULL.

Examples

```r
# Please make sure DemoSubject is correctly installed
# The subject is ~1GB from Github
brain <- rave_brain("demo/DemoSubject")

brain

if(interactive() && !is.null(brain)){
  brain$plot()
}
```

Description

Import data into 'rave' projects. Supported file formats include 'Matlab', 'HDF5', 'EDF+', 'BrainVision' ('.eeg/.dat/.vhdr'). Supported file structures include 'rave' native structure and 'BIDS' (very limited) format. Please see [https://openwetware.org/wiki/RAVE:ravepreprocess](https://openwetware.org/wiki/RAVE:ravepreprocess) for tutorials.

Usage

```r
rave_import(
  project_name,
  subject_code,
  blocks,
  electrodes,
  format,
  sample_rate,
  conversion = NA,
  data_type = c("LFP", "ECoG", "EEG"),
  task_runs = NULL,
)```
add = FALSE,
...
)

Arguments

project_name  project name, for 'rave' native structure, this can be any character; for 'BIDS' format, this must be consistent with 'BIDS' project name. For subjects with multiple tasks, see Section "RAVE' Project"

subject_code  subject code in character. For 'rave' native structure, this is a folder name under raw directory. For 'BIDS', this is subject label without "sub-" prefix

blocks  characters, for 'rave' native format, this is the folder names subject directory; for 'BIDS', this is session name with "ses-", Section "Block vs. Session" for different meaning of "blocks" in 'rave' and 'BIDS'

electrodes  integers electrode numbers

format  integer from 1 to 6, or character. For characters, you can get options by running names(IMPORT_FORMATS)

sample_rate  sample frequency, must be positive

corversion  physical unit conversion, choices are NA, V, mV, uV

data_type  electrode type; only 'LFP', 'ECoG', and 'EEG' are supported

task_runs  for 'BIDS' formats only, see Section "Block vs. Session"

add  whether to add electrodes. If set to true, then only new electrodes are allowed to be imported, blocks will be ignored and trying to import electrodes that have been imported will still result in error.

...  other parameters

Value

None

'RAVE' Project

A 'rave' project can be very flexible. A project can refer to a task, a research objective, or "arbitrarily" as long as you find common research interests among subjects. One subject can appear in multiple projects with different blocks, hence project_name should be objective-based. There is no concept of "project" in 'rave' raw directory. When importing data, you choose subset of blocks from subjects forming a project.

When importing 'BIDS' data into 'rave', project_name must be consistent with 'BIDS' project name as a compromise. Once imported, you may change the project folder name in imported rave data directory to other names. Because once raw traces are imported, 'rave' data will become self-contained and 'BIDS' data are no longer required for analysis. This naming inconsistency will also be ignored.
Block vs. Session

' rave' and ' BIDS' have different definitions for a "chunk" of signals. In ' rave', we use "block". it means combination of session (days), task, and run, i.e. a block of continuous signals captured. Raw data files are supposed to be stored in file hierarchy of <raw-root>/<subject_code>/<block>/<datafiles>. In ' BIDS', sessions, tasks, and runs are separated, and only session names are indicated under subject folder. Because some previous compatibility issues, argument ' block' refers to direct folder names under subject directories. This means when importing data from ' BIDS' format, block argument needs to be session names to comply with ' subject/block' structure, and there is an additional mandatory argument task_runs especially designed for ' BIDS' format.

For ' rave' native raw data format, block will be as-is once imported.
For ' BIDS' format, task_runs will be treated as blocks once imported.

File Formats

Following file structure. Here use project " demo" and subject " YAB" and block "008"), electrode 14 as an example.

format=1, or " .mat/.h5 file per electrode per block" folder <raw>/YAB/008 contains 'Matlab' or ' HDF5' files per electrode. Data file name should look like "xxx_14.mat"

format=2, or "Single .mat/.h5 file per block" <raw>/YAB/008 contains only one ' Matlab' or ' HDF5' file. Data within the file should be a 2-dimensional matrix, where the column 14 is signal recorded from electrode 14

format=3, or "Single EDF(+) file per block" <raw>/YAB/008 contains only one 'edf' file

format=4, or "Single BrainVision file (.vhdr+.eeg, .vhdr+.dat) per block" <raw>/YAB/008 contains only one 'vhdr' file, and the data file must be inferred from the header file

format=5, or "BIDS & EDF(+)" <bids>/demo/sub-YAB/ses-008/ must contains *.electrodes.tsv, each run must have channel file. The channel files and electrode file must be consistent in names.

Argument task_runs is mandatory, characters, combination of session, task name, and run number. For example, a task header file in BIDS with name ' sub-YAB_ses-008_task-visual_run-01_iieeg.edf' has task_runs name as ' 008-visual-01', where the first ' 008' refers to session, 'visual' is task name, and the second ' 01' is run number.

format=6, or "BIDS & BrainVision (.vhdr+.eeg, .vhdr+.dat)" Same as previous format "BIDS & EDF(+)", but data files have 'BrainVision' formats.

read-brainvision-eeg  Load from 'BrainVision' file

Description

Read in 'eeg' or 'iieeg' data from 'BrainVision' files with .eeg or .dat extensions.

Usage

read_eeg_header(file)

read_eeg_data(header, path = NULL)
read-brainvision-eeg

Arguments

file: path to 'vhdr' header file
header: header object returned by read_eeg_header
path: optional, path to data file if original data file is missing or renamed; must be absolute path.

Details

A 'BrainVision' dataset is usually stored separately in header file (.vhdr), marker file (.vmrk, optional) and data file (.eeg or .dat). These files must store under a same folder to be read into R. Header data contains channel information. Data "channel" contains channel name, reference, resolution and physical unit. "resolution" times digital data values is the physical value of the recorded data. read_eeg_data makes this conversion internally. "unit" is the physical unit of recordings. By default 'uV' means micro-volts.

Marker file that ends with .vmrk is optional. If the file is indicated by header file and exists, then a marker table will be included when reading headers. A marker table contains six columns: marker number, type, description, start position (in data point), size (duration in data points), and target channel (0 means applied for all channels).

Signal file name is usually contained within header file. Therefore it is desired that the signal file name never changed once created. However, in some cases when the signal files are renamed and cannot be indexed by header files, please specify path to force load signals from a different file.

Value

read_eeg_header returns a list containing information below:

- raw: raw header contents
- common: a list of descriptors of header
- channels: table of channels, including number, reference, resolution and unit
- sample_rate: sampling frequency
- root_path: directory to where the data is stored
- channel_counts: total channel counts
- markers: NULL if marker file is missing, or list of marker description and table containing 6 columns.

read_eeg_data returns header, signal data and data description:

- data: a matrix of signal values. Each row is a channel and each column is a time point.

Examples

```r
header_file <- 'sub-01_ses-01_task-visual_run-01_ieeg.vhdr'

if( file.exists(header_file) ){  # load a subject header
  header <- read_eeg_header(header_file)
```
# load entire signal
data <- read_eeg_data(header)

data$description

---

**read-write-fst**

*Read a 'fst' file*

**Description**

Read a 'fst' file

**Usage**

```r
save_fst(x, path, ...)
load_fst(path, ..., as.data.table = TRUE)
```

**Arguments**

- `x`: data frame to write to path
- `path`: path to 'fst' file: must not be connection.
- `...`: passed to `read_fst` or `write_fst`
- `as.data.table`: passed to `read_fst` in `fst` package

---

**read_csv_ieeg**

*Read comma separated value file and ignore headers*

**Description**

Resolved some irregular 'iEEG' format where the header could be missing.

**Usage**

```r
read_csv_ieeg(file, nrows = Inf, drop = NULL)
```

**Arguments**

- `file`: comma separated value file to read from. The file must contains all numerical values
- `nrows`: number of rows to read
- `drop`: passed to `fread`
Details

The function checks the first two rows of comma separated value file. If the first row has different storage.mode than the second row, then the first row is considered header, otherwise header is treated missing. Note file must have at least two rows.

read_edf_header Read 'EDF(+)' or 'BDF(+)' file headers

Description

Wrapper of readEdfHeader, but added some information

Usage

read_edf_header(path)

Arguments

path file path, passed to readEdfHeader

Details

The added names are: isAnnot2, sampleRate2, and unit2. To avoid conflict with other names, there is a "2" appended to each names. isAnnot2 indicates whether each channel is annotation channel or recorded signals. sampleRate2 is a vector of sample rates for each channels. unit2 is physical unit of recorded signals. For 'iEEG' data, this is electric potential unit, and choices are 'V' for volt, 'mV' for millivolt, and 'uV' for micro-volt. For more details, see https://www.edfplus.info/specs/edftexts.html

Value

A list is header information of an 'EDF/BDF' file.

See Also

readEdfHeader
read_edf_signal

Read 'EDF(+)' or 'BDF(+)' file signals

Description
Read 'EDF(+)' or 'BDF(+)' file signals

Usage
read_edf_signal(
  path,
  signal_numbers = NULL,
  convert_volt = c("NA", "V", "mV", "uV")
)

Arguments
path     file path, passed to readEdfHeader
signal_numbers channel/electrode numbers
convert_volt convert voltage (electric potential) to a new unit, NA means no conversion, other
choices are 'V', 'mV', and 'uV'.

Value
A list containing header information, signal lists, and channel/electrode names. If signal_numbers is specified, the corresponding names should appear as selected_signal_names. get_signal() can get physical signals after unit conversion.

read_mat

Read 'Matlab' files

Description
A compatible reader that can read both 'Matlab' files prior and after version 6.0

Usage
read_mat(file, ram = TRUE)

Arguments
file     path to a 'Matlab' file
ram      whether to load data into memory. Only available when the file is in 'HDF5' format. Default is false and will load arrays, if set to true, then lazy-load data. This is useful when array is very large.
**Details**

`readMat` can only read 'Matlab' files prior to version 6. After version 6, 'Matlab' uses 'HDF5' format to store its data, and `read_mat` can handle both cases.

The performance of `read_mat` can be limited when the file is too big or has many datasets as it reads all the data contained in 'Matlab' file into memory.

**Value**

A list of All the data stored in the file

**See Also**

`readMat, load_h5`

**Examples**

```r
# Matlab .mat <= v7.3
x <- matrix(1:16, 4)
f <- tempfile()
R.matlab::writeMat(con = f, x = x)

read_mat(f)

# Matlab .mat >= v7.3, using hdf5
# Make sure you have installed hdf5r
if( dipsaus::package_installed('hdf5r') ){
  f <- tempfile()
  save_h5(x, file = f, name = 'x')

  read_mat(f)

  # For v7.3, you don't have to load all data into RAM
  dat <- read_mat(f, ram = FALSE)
  dat

  dat$x[]
}
```

---

**safe_read_csv**

*Read comma separated value files with given column classes*
safe_read_csv

Description

Read comma separated value files with given column classes

Usage

safe_read_csv(
  file,
  header = TRUE,
  sep = ",",
  colClasses = NA,
  skip = 0,
  quote = "\\",
  ...
  stringsAsFactors = FALSE
)

Arguments

file, header, sep, colClasses, skip, quote, stringsAsFactors, ...

passed to read.csv

Details

Reading a comma separated value file using built-in function read.csv might result in some unexpected behavior. safe_read_csv does some preprocessing on the format so that it takes care of the following cases.

1. If skip exceeds the maximum rows of the data, return a blank data frame instead of raising error.
2. If row names are included in the file, colClasses automatically skip that column and starts from the second column.
3. If length of colClasses does not equal to the number of columns, instead of cycling the class types, we set those columns to be NA type and let read.csv decide the default types.
4. stringsAsFactors is by default FALSE to be consistent with R 4.0, if the function is called in R 3.x.

Value

A data frame

Examples

f <- tempfile()
x <- data.frame(a = letters[1:10], b = 1:10, c = 2:11)

# ------------------ Auto-detect row names ------------------
# Write with rownames
utils::write.csv(x, f, row.names = LETTERS[2:11])

# read csv with base library utils
table1 <- utils::read.csv(f, colClasses = c('character', 'character'))

# 4 columns including row names
str(table1)

# read csv via safe_read_csv
table2 <- safe_read_csv(f, colClasses = c('character', 'character'))

# row names are automatically detected, hence 3 columns
# Only first columns are characters, the third column is auto
# detected as numeric
str(table2)

# read table without row names
utils::write.csv(x, f, row.names = FALSE)
table2 <- safe_read_csv(f, colClasses = c('character', 'character'))

# still 3 columns, and row names are 1:nrow
str(table2)

# --------------- Blank data frame when nrow too large ---------------
# instead of raising errors, return blank data frame
safe_read_csv(f, skip = 1000)

---

safe_write_csv | Save data to comma separated value files with backups

**Description**

Save comma separated value files, if file exists, backup original file.

**Usage**

```r
safe_write_csv(x, file, ..., quiet = FALSE)
```

**Arguments**

- `x, file, ...` pass to `write.csv`
- `quiet` whether to suppress overwrite message

**Value**

Normalized path of file
save_h5

Examples

```r
f <- tempfile()
x <- data.frame(a = 1:10)

# File not exists, same as write file, returns normalized 'f'
safe_write_csv(x, f)

# Check whether file exists
file.exists(f)

# write again, and the old file will be copied
safe_write_csv(x, f)
```

save_h5  

Save objects to 'HDF5' file without trivial checks

Description

Save objects to 'HDF5' file without trivial checks

Usage

```r
save_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

Value

Absolute path of the file saved

See Also

load_h5

Examples

```r
file <- tempfile()
x <- array(1:120, dim = 2:5)
# save x to file with name /group/dataset/1
save_h5(x, file, '/group/dataset/1', chunk = dim(x))
# read data
y <- load_h5(file, '/group/dataset/1')
y[]
```

save_meta2 Function to save meta data to 'RAVE' subject

Description

Function to save meta data to 'RAVE' subject

Usage

```r
save_meta2(data, meta_type, project_name, subject_code)
```

Arguments

- **data**: data table
- **meta_type**: see load meta
- **project_name**: project name
- **subject_code**: subject code

Value

Either none if no meta matched or the absolute path of file saved.
save_yaml

Description

Write named list to file

Usage

save_yaml(x, file, ...)

Arguments

x  a named list, fastmap2, or anything that can be transformed into named list via as.list
file, ... passed to write_yaml

Value

Normalized file path

See Also

fastmap2, load_yaml, read_yaml, write_yaml,

Examples

```r
x <- list(a = 1, b = 2)
f <- tempfile()
save_yaml(x, f)
load_yaml(f)

map <- dipsaus::fastmap2(missing_default = NA)
map$c <- 'lol'
load_yaml(f, map = map)

map$a
map$d
```
**Tensor**  

*R6 Class for large Tensor (Array) in Hybrid Mode*

---

**Description**

can store on hard drive, and read slices of GB-level data in seconds

**Public fields**

- `dim` dimension of the array
- `dimnames` dimension names of the array
- `use_index` whether to use one dimension as index when storing data as multiple files
- `hybrid` whether to allow data to be written to disk
- `last_used` timestamp of the object was read
- `temporary` whether to remove the files once garbage collected

**Active bindings**

- `varnames` dimension names (read-only)
- `read_only` whether to protect the swap files from being changed
- `swap_file` file or files to save data to

**Methods**

**Public methods:**

- `Tensor$finalize()`
- `Tensor$print()`
- `Tensor$.use_multi_files()`
- `Tensor$new()`
- `Tensor$flatten()`
- `Tensor$to_swap()`
- `Tensor$to_swap_now()`
- `Tensor$get_data()`
- `Tensor$set_data()`
- `Tensor$collapse()`
- `Tensor$operate()`

**Method** `finalize()`:

release resource and remove files for temporary instances

*Usage:*

```
Tensor$finalize()
```

**Method** `print()`:

print out the data dimensions and snapshot
Usage:
Tensor$print(...)

Arguments:
... ignored

Returns: self

Method .use_multi_files(): Internally used, whether to use multiple files to cache data instead of one

Usage:
Tensor$.use_multi_files(mult)

Arguments:
mult logical

Method new(): constructor

Usage:
Tensor$new(
data, dim, dimnames, varnames, hybrid = FALSE, use_index = FALSE, swap_file = temp_tensor_file(), temporary = TRUE, multi_files = FALSE)

Arguments:
data numeric array
dim dimension of the array
dimnames dimension names of the array
varnames characters, names of dimnames
hybrid whether to enable hybrid mode
use_index whether to use the last dimension for indexing
swap_file where to store the data in hybrid mode files to save data by index; default stores in
raveio getopt('tensor_temp_path')
temporary whether to remove temporary files when existing
multi_files if use_index is true, whether to use multiple

Method subset(): subset tensor

Usage:
Tensor$subset(..., drop = FALSE, data_only = FALSE, .env = parent.frame())

Arguments:
... dimension slices
drop whether to apply drop on subset data
data_only whether just return the data value, or wrap them as a Tensor instance
.env environment where ... is evaluated

*Returns:* the sliced data

**Method `flatten()`:** converts tensor (array) to a table (data frame)

*Usage:*

```
Tensor$flatten(include_index = FALSE, value_name = "value")
```

*Arguments:*

- `include_index` logical, whether to include dimension names
- `value_name` character, column name of the value

*Returns:* a data frame with the dimension names as index columns and `value_name` as value column

**Method `to_swap()`:** Serialize tensor to a file and store it via *write_fst*

*Usage:*

```
Tensor$to_swap(use_index = FALSE, delay = 0)
```

*Arguments:*

- `use_index` whether to use one of the dimension as index for faster loading
- `delay` if greater than 0, then check when last used, if not long ago, then do not swap to hard drive. If the difference of time is greater than delay in seconds, then swap immediately.

**Method `to_swap_now()`:** Serialize tensor to a file and store it via *write_fst* immediately

*Usage:*

```
Tensor$to_swap_now(use_index = FALSE)
```

*Arguments:*

- `use_index` whether to use one of the dimension as index for faster loading

**Method `get_data()`:** restore data from hard drive to memory

*Usage:*

```
Tensor$get_data(drop = FALSE, gc_delay = 3)
```

*Arguments:*

- `drop` whether to apply drop to the data
- `gc_delay` seconds to delay the garbage collection

*Returns:* original array

**Method `set_data()`:** set/replace data with given array

*Usage:*

```
Tensor$set_data(v)
```

*Arguments:*

- `v` the value to replace the old one, must have the same dimension
notice the a tensor is an environment. If you change at one place, the data from all other places will change. So use it carefully.

**Method collapse():** apply mean, sum, or median to collapse data

*Usage:*

```
Tensor$collapse(keep, method = "mean")
```

*Arguments:*

- `keep` which dimensions to keep
- `method` "mean", "sum", or "median"

*Returns:* the collapsed data

**Method operate():** apply the tensor by anything along given dimension

*Usage:*

```
Tensor$operate(
  by,
  fun = .Primitive("/"),
  match_dim,
  mem_optimize = FALSE,
  same_dimension = FALSE
)
```

*Arguments:*

- `by` R object
- `fun` function to apply
- `match_dim` which dimensions to match with the data
- `mem_optimize` optimize memory
- `same_dimension` whether the return value has the same dimension as the original instance

### Examples

```r
if(interactive()) # Avoid checkings from CRAN

# Create a tensor
ts <- Tensor$new(
  data = 1:18000000, c(3000,300,20),
  dimnames = list(A = 1:3000, B = 1:300, C = 1:20),
  varnames = c('A', 'B', 'C'))

# Size of tensor when in memory is usually large
# `lobstr::obj_size(ts)` -> 8.02 MB

# Enable hybrid mode
ts$to_swap_now()

# Hybrid mode, usually less than 1 MB
# `lobstr::obj_size(ts)` -> 814 kB
```
# Subset data
start1 <- Sys.time()
subset(ts, C ~ C < 10 & C > 5, A ~ A < 10)
#> Dimension: 9 x 300 x 4
#> - A: 1, 2, 3, 4, 5, 6,...
#> - B: 1, 2, 3, 4, 5, 6,...
#> - C: 6, 7, 8, 9
end1 <- Sys.time(); end1 - start1
#> Time difference of 0.188035 secs

# Join tensors
start2 <- Sys.time()
ts <- lapply(1:20, function(ii){
    Tensor$new(
        data = 1:9000, c(30,300,1),
        dimnames = list(A = 1:30, B = 1:300, C = ii),
        varnames = c('A', 'B', 'C'), use_index = 2)
})
ts <- join_tensors(ts, temporary = TRUE)
end2 <- Sys.time(); end2 - start2
#> Time difference of 0.485908 secs

# test_hdspeed

## Description

Simple hard disk speed test

## Usage

test_hdspeed(path = tempdir(),
file_size = 1e+06,
quiet = FALSE,
abort_if_slow = TRUE,
use_cache = FALSE)

## Arguments

- **path**
  - an existing directory where to test speed, default is temporary local directory.
- **file_size**
  - in bytes, default is 1 MB.
- **quiet**
  - should verbose messages be suppressed?
- **abort_if_slow**
  - abort test if hard drive is too slow. This usually happens when the hard drive is connected via slow internet: if the write speed is less than 0.1 MB per second.
- **use_cache**
  - if hard drive speed was tested before, abort testing and return cached results or not; default is false.
Value

A vector of two: writing and reading speed in MB per seconds.

---

time_diff2

*Calculate time difference in seconds*

Description

Calculate time difference in seconds

Usage

```
time_diff2(start, end, units = "secs", label = ")
```

Arguments

- `start`, `end`: start and end of timer
- `units`: passed to `time_delta`
- `label`: `rave-units` label for display purpose.

Value

A number inherits `rave-units` class.

See Also

`as_rave_unit`

Examples

```
start <- Sys.time()
Sys.sleep(0.1)
end <- Sys.time()
dif <- time_diff2(start, end, label = 'Running ')
print(dif, digits = 4)

is.numeric(dif)

dif + 1
```
Index

* datasets
  rave-raw-validation, 33

as_rave_project, 3
as_rave_subject, 4
as_rave_unit, 4, 67

cat2, 5
catgl, 5
configure_knitr, 6

dir.create, 7
dir_create2, 7
drop, 17, 20, 64

ECoGTensor, 8

fastmap2, 30, 61
find_path, 9
fread, 53
freesurfer_brain2, 48

get_projects, 10
get_val2, 11

h5_names, 12
h5_valid, 12

import_formats (rave-raw-validation), 33

is.blank, 13
is.zerolenth, 14
is_valid_ish, 11, 14

join_tensors, 15

LazyFST, 16, 28
LazyH5, 16, 17, 18, 28
LFP_electrode, 21
load bids ieeg header, 26

load_fst (read-write-fst), 53
load_fst_or_h5, 27
load_h5, 28, 56, 60
load_meta2, 29, 47, 48
load_targets (rave-pipeline), 30
load_yaml, 30, 61

mode, 14, 20

pipeline_attach (rave-pipeline), 30
pipeline_build (rave-pipeline), 30
pipeline_create_subject_pipeline (rave-pipeline), 30
pipeline_create_template, 6
pipeline_create_template (rave-pipeline), 30
pipeline_debug (rave-pipeline), 30
pipeline_description (rave-pipeline), 30
pipeline_find (rave-pipeline), 30
pipeline_fork (rave-pipeline), 30
pipeline_hasname (rave-pipeline), 30
pipeline_list (rave-pipeline), 30
pipeline_progress (rave-pipeline), 30
pipeline_read (rave-pipeline), 30
pipeline_root (rave-pipeline), 30
pipeline_run (rave-pipeline), 30
pipeline_vartable (rave-pipeline), 30
pipeline_visualize (rave-pipeline), 30
pipeline_watch (rave-pipeline), 30

rave-pipeline, 30
rave-raw-validation, 33
rave_brain, 48
rave_import, 49
RAVEAbstractElectrode, 22, 35
RAVEEpoch, 35, 36, 38
raveio-option, 40
raveio::RAVEAbstractElectrode, 21
raveio::Tensor, 8
raveio::confpath (raveio-option), 40
raveio_getopt (raveio-option), 40
raveio_resetopt (raveio-option), 40
raveio_setopt (raveio-option), 40
RAVEPreprocessSettings, 41, 46
RAVEProject, 3, 44, 46
RAVESubject, 4, 33, 35, 36, 41, 42, 46, 48
read-brainvision-eeg, 51
read-write-fst, 53
read_csv_ieeg, 53
read_edf_header, 54
read_edf_signal, 55
read_eeg_data (read-brainvision-eeg), 51
read_eeg_header (read-brainvision-eeg), 51
read_mat, 55
read_yaml, 30, 61
readEdfHeader, 54
readMat, 56
safe_read_csv, 56
safe_write_csv, 58
save_fst (read-write-fst), 53
save_h5, 29, 59
save_meta2, 60
save_yaml, 30, 61
storage.mode, 54
tar_progress_summary, 32
tar_read, 33
Tensor, 8, 9, 15, 16, 62
test_hdspeed, 66
time_delta, 67
time_diff2, 67
validate_raw_file
   (rave-raw-validation), 33
write_fst, 53, 64
write_yaml, 30, 61