Package ‘raveio’

October 14, 2022

**Type** Package

**Title** File-System Toolbox for RAVE Project

**Version** 0.0.9

**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** 7.2.1

**SystemRequirements** little-endian platform

**biocViews** Infrastructure, DataImport

**Imports** utils, data.table, edfReader (>= 1.2.1), dipSAUS, filearray (>= 0.1.3), fst (>= 0.9.2), glue, globals, hdf5r (>= 1.3.4), 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 (>= 2.1.2), promises (>= 1.2.0), threeBrain (>= 0.2.5)

**Suggests** rpymat, visNetwork, testthat, reticulate, knitr, rmarkdown, bs4Dash, clustermq, shiny, shinybusy, shinyWidgets, arrow, ravetools, later (>= 1.3.0)
R topics documented:

as_rave_project ........................................ 4
as_rave_subject ........................................ 4
as_rave_unit ........................................... 5
auto_process_blackrock .................................. 5
backup_file ............................................. 6
BlackrockFile ........................................... 7
cache_path ............................................... 9
catgl ..................................................... 10
cmd_run_3dAllineate ................................... 11
collapse2 ............................................... 14
collapse_power ......................................... 15
configure_knit ......................................... 16
convert_blackrock ........................................ 17
dir_create2 ............................................ 18
ECoGTensor ............................................. 18
find_path ................................................ 20
generate_reference ..................................... 21
get_projects ............................................ 22
get_val2 .................................................. 22
h5_names ................................................ 23
h5_valid .................................................. 24
import_electrode_table .................................. 25
install_modules ........................................ 25
is.blank .................................................. 26
is.zerolenth ............................................. 26
is_valid_ish ............................................. 27
join_tensors ............................................. 28
LazyFST ................................................. 29
LazyH5 .................................................. 31
LFP_electrode ........................................... 34
LFP_reference ........................................... 38
load_bids_ieeg_header .................................. 42
load_fst_or_h5 ......................................... 43
load_h5 .................................................. 44
load_meta2 ............................................. 45
load_yaml ............................................... 46
module_add ............................................. 47
### as_rave_project

**Convert character to RAVEProject instance**

**Description**
Convert character to RAVEProject instance

**Usage**
```r
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**
```r
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

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 = '
')

auto_process_blackrock

Monitors 'BlackRock' output folder and automatically import data into 'RAVE'

Description
Automatically import 'BlackRock' files from designated folder and perform 'Notch' filters, 'Wavelet' transform; also generate epoch, reference files.
Usage

auto_process_blackrock(
  watch_path,
  project_name = "automated",
  task_name = "RAVEWatchDog",
  scan_interval = 10,
  time_threshold = Sys.time(),
  max_jobs = 1L,
  as_job = NA,
  dry_run = FALSE,
  config_open = dry_run
)

Arguments

  watch_path        the folder to watch
  project_name      the project name to generate
  task_name         the watcher's name
  scan_interval     scan the directory every scan_interval seconds, cannot be lower than 1
  time_threshold    time-threshold of files: all files with modified time prior to this threshold will be ignored; default is current time
  max_jobs          maximum concurrent imports, default is 1
  as_job            whether to run in 'RStudio' background job or to block the session when monitoring; default is auto-detected
  dry_run           whether to dry-run the code (instead of executing the scripts, return the watcher's instance and open the settings file); default is false
  config_open       whether to open the pipeline configuration file; default is equal to dry_run

Value

When dry_run is true, then the watcher's instance will be returned; otherwise nothing will be returned.

backup_file

Back up and rename the file or directory

Description

Back up and rename the file or directory

Usage

backup_file(path, remove = FALSE, quiet = FALSE)
Arguments

path path to a file or a directory
remove whether to remove the original path; default is false
quiet whether not to verbose the messages; default is false

Value

FALSE if nothing to back up, or the back-up path if path exists

Examples

```r
path <- tempfile()
file.create(path)

path2 <- backup_file(path, remove = TRUE)

file.exists(c(path, path2))
unlink(path2)
```

---

BlackrockFile

*Class definition to load data from 'BlackRock' 'Micro-systems' files*

Description

Currently only supports minimum file specification version 2.3. Please contact the package maintainer or 'RAVE' team if older specifications are needed.

Public fields

- block character, session block ID

Active bindings

- base_path absolute base path to the file
- version 'NEV' specification version
- electrode_table electrode table
- sample_rate_nev_timestamp sample rate of 'NEV' data packet time-stamps
- has_nsx named vector of 'NSx' availability
- recording_duration recording duration of each 'NSx'
- sample_rates sampling frequencies of each 'NSx' file
Methods

Public methods:

• BlackrockFile$print()
• BlackrockFile$new()
• BlackrockFile$nev_path()
• BlackrockFile$nsx_paths()
• BlackrockFile$refresh_data()
• BlackrockFile$get_epoch()
• BlackrockFile$get_waveform()
• BlackrockFile$get_electrode()
• BlackrockFile$clone()

Method print(): print user-friendly messages

Usage:
BlackrockFile$print()

Method new(): constructor

Usage:
BlackrockFile$new(path, block, nev_data = TRUE)

Arguments:
path  the path to ‘BlackRock’ file, can be with or without file extensions
block  session block ID; default is the file name
nev_data  whether to load comments and ‘waveforms’

Method nev_path(): get ‘NEV’ file path

Usage:
BlackrockFile$nev_path()

Returns: absolute file path

Method nsx_paths(): get ‘NSx’ file paths

Usage:
BlackrockFile$nsx_paths(which = NULL)

Arguments:
which  which signal file to get, or NULL to return all available paths, default is NULL; must be integers

Returns: absolute file paths

Method refresh_data(): refresh and load ‘NSx’ data

Usage:
BlackrockFile$refresh_data(force = FALSE, verbose = TRUE, nev_data = FALSE)

Arguments:
force  whether to force reload data even if the data has been loaded and cached before
cache_path

Manipulate cached data on the file systems

Description

Manipulate cached data on the file systems

Usage

```r
cache_root(check = FALSE)

clearCachedFiles(subject_code, quiet = FALSE)
```
Arguments

- **check**
  whether to ensure the cache root path

- **subject_code**
  subject code to remove; default is missing. If subject_code is provided, then only this subject-related cache files will be removed.

- **quiet**
  whether to suppress the message

Details

'RAVE' intensively uses cache files. If running on personal computers, the disk space might be filled up very quickly. These cache files are safe to remove if there is no 'RAVE' instance running. Function clear_cached_files is designed to remove these cache files. To run this function, please make sure that all 'RAVE' instances are shutdown.

Value

cache_root returns the root path that stores the 'RAVE' cache data; clear_cached_files returns nothing

Examples

```r
cache_root()
```

---

**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.
cmd_run_3dAllineate

Value

The message as characters

Examples

```r
# ------------------ Basic Styles ---------------------
# Temporarily change verbose level for example
raveio_setopt('verbose_level', 'DEBUG', FALSE)

# debug
catgl('Debug message', level = 'DEBUG')

# default
catgl('Default message', level = 'DEFAULT')

# info
catgl('Info message', level = 'INFO')

# warning
catgl('Warning message', level = 'WARNING')

# error
catgl('Error message', level = 'ERROR')

try({
  # fatal, will call stop and raise error
  catgl('Error message', level = 'FATAL')
}, silent = TRUE)

# ------------------ Muffle messages ---------------------
# Temporarily change verbose level to 'WARNING'
raveio_setopt('verbose_level', 'WARNING', FALSE)

# default, muffled
catgl('Default message')

# message printed for level >= Warning
  catgl('Default message', level = 'WARNING')
catgl('Default message', level = 'ERROR')
```
cmd_run_3dAllineate

Description

These shell commands are for importing 'DICOM' images to 'Nifti' format, reconstructing cortical surfaces, and align the CT to 'MRI'. The commands are only tested on 'MacOS' and 'Linux'. On 'Windows' machines, please use the 'WSL2' system.

Usage

```r
cmd_run_3dAllineate(
  subject,
  mri_path,
  ct_path,
  overwrite = FALSE,
  command_path = NULL,
  dry_run = FALSE,
  verbose = dry_run
)
```

```r
cmd_execute(
  script,
  script_path,
  command = "bash",
  dry_run = FALSE,
  backup = TRUE,
  ...
)
```

```r
cmd_run_dcm2niix(
  subject,
  src_path,
  type = c("MRI", "CT"),
  merge = c("Auto", "No", "Yes"),
  float = c("Yes", "No"),
  crop = c("No", "Yes", "Ignore"),
  overwrite = FALSE,
  command_path = NULL,
  dry_run = FALSE,
  verbose = dry_run
)
```

```r
cmd_run_flirt(
  subject,
  mri_path,
  ct_path,
  overwrite = FALSE,
  command_path = NULL,
  dry_run = FALSE,
  verbose = dry_run
)
cmd_run_recon_all(
    subject,
    mri_path,
    args = c("-all", "-autorecon1", "-autorecon2", "-autorecon3", "-autorecon2-cp",
             "-autorecon2-wm", "-autorecon2-pial"),
    work_path = NULL,
    overwrite = FALSE,
    command_path = NULL,
    dry_run = FALSE,
    verbose = dry_run
)

Arguments

subject characters or a RAVESubject instance
mri_path the absolute to 'MRI' volume; must in 'Nifti' format
tc_path the absolute to 'CT' volume; must in 'Nifti' format
overwrite whether to overwrite existing files; default is false
command_path command line path if 'RAVE' cannot find the command binary files
dry_run whether to run in dry-run mode; under such mode, the shell command will not execute. This is useful for debugging scripts; default is false
verbose whether to print out the command script; default is true under dry-run mode, and false otherwise
script the shell script
script_path path to run the script
command which command to invoke; default is 'bash'
backup whether to back up the script file immediately; default is true
... passed to system2
src_path source of the 'DICOM' or 'Nifti' image (absolute path)
type type of the 'DICOM' or 'Nifti' image; choices are 'MRI' and 'CT'
merge, float, crop 'dcm2niix' conversion arguments; ignored when the source is in 'Nifti' format
args further arguments in the shell command, especially the 'FreeSurfer' reconstruction command
work_path work path for 'FreeSurfer' command;

Value

A list of data containing the script details:

script script details
script_path where the script should/will be saved
dry_run whether dry-run mode is turned on
log_file path to the log file
execute a function to execute the script
collapse2  

**Collapse high-dimensional tensor array**

Description

Collapse high-dimensional tensor array

Usage

```r
collapse2(x, keep, method = c("mean", "sum"), ...)
```

## S3 method for class 'FileArray'

```r
collapse2(x, keep, method = c("mean", "sum"), ...)
```

## S3 method for class 'Tensor'

```r
collapse2(x, keep, method = c("mean", "sum"), ...)
```

## S3 method for class 'array'

```r
collapse2(x, keep, method = c("mean", "sum"), ...)
```

Arguments

- `x`: R array, `FileArray-class`, or `Tensor` object
- `keep`: integer vector, the margins to keep
- `method`: character, calculates mean or sum of the array when collapsing
- `...`: passed to other methods

Value

A collapsed array (or a vector or matrix), depending on `keep`

See Also

- `collapse`

Examples

```r
x <- array(1:16, rep(2, 4))
collapse2(x, c(3, 2))

# Alternative method, but slower when 'x' is a large array
apply(x, c(3, 2), mean)
```
**Description**

Collapse power array with given analysis cubes

**Usage**

```r
collapse_power(x, analysis_index_cubes)
```

### S3 method for class 'array'

```r
collapse_power(x, analysis_index_cubes)
```

### S3 method for class 'FileArray'

```r
collapse_power(x, analysis_index_cubes)
```

**Arguments**

- `x`  
  A `FileArray-class` array, must have 4 modes in the following sequence: Frequency, Time, Trial, and Electrode

- `analysis_index_cubes`  
  A list of analysis indices for each mode

**Value**

A list of collapsed (mean) results:

- `freq_trial_elec`  
  Collapsed over time-points

- `freq_time_elec`  
  Collapsed over trials

- `time_trial_elec`  
  Collapsed over frequencies

- `freq_time`  
  Collapsed over trials and electrodes

- `freq_elec`  
  Collapsed over trials and time-points

- `freq_trial`  
  Collapsed over time-points and electrodes

- `time_trial`  
  Collapsed over frequencies and electrodes

- `time_elec`  
  Collapsed over frequencies and trials

- `trial_elec`  
  Collapsed over frequencies and time-points

- `freq`  
  Power per frequency, averaged over other modes

- `time`  
  Power per time-point, averaged over other modes

- `trial`  
  Power per trial, averaged over other modes
Examples

```r
# Generate a 4-mode tensor array
x <- filearray::filearray_create(
  tempfile(), dimension = c(16, 100, 20, 5),
  partition_size = 1
)
x[] <- rnorm(160000)
dnames <- list(
  Frequency = 1:16,
  Time = seq(0, 1, length.out = 100),
  Trial = 1:20,
  Electrode = 1:5
)
dimnames(x) <- dnames

# Collapse array
results <- collapse_power(x, list(
  overall = list(),
  A = list(Trial = 1:5, Frequency = 1:6),
  B = list(Trial = 6:10, Time = 1:50)
))

# Plot power over frequency and time
groupB_result <- results$B
image(t(groupB_result$freq_time),
  x = dnames$Time[groupB_result$cube_index$Time],
  y = dnames$Frequency[groupB_result$cube_index$Frequency],
  xlab = "Time (s)",
  ylab = "Frequency (Hz)",
  xlim = range(dnames$Time))
```

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

```r
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

---

**convert_blackrock**

*Convert 'BlackRock' 'NEV/NSx' files*

---

**Description**

Convert 'BlackRock' 'NEV/NSx' files

**Usage**

```r
convert_blackrock(
  file,
  block = NULL,
  subject = NULL,
  to = NULL,
  comments = TRUE,
  format = c("mat", "hdf5")
)
```

**Arguments**

- **file**: path to any 'NEV/NSx' file
- **block**: the block name, default is file name
- **subject**: subject code to save the files; default is NULL
- **to**: save to path, must be a directory; default is under the file path. If subject is provided, then the default is subject raw directory path
- **comments**: whether to extract comment section as epoch; default is true
- **format**: output format, choices are 'mat' or 'hdf5'

**Value**

The results will be stored in directory specified by `to`. Please read the output message carefully.
**dir_create2**  
*Force creating directory with checks*

**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**  
```r
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, must match with data
dimnames list of dimension names, equal length as dim
varnames names of dimension names, 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

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'))
generate_reference

Generate common average reference signal for 'RAVE' subjects

Description
To properly run this function, please install ravetools package.

Usage
generate_reference(subject, electrodes)

Arguments
subject subject ID or RAVESubject instance
electrodes electrodes to calculate the common average; these electrodes must run through 'Wavelet' first

Details
The goal of generating common average signals is to capture the common movement from all the channels and remove them out from electrode signals.

The common average signals will be stored at subject reference directories. Two exact same copies will be stored: one in 'HDF5' format such that the data can be read universally by other programming languages; one in filearray format that can be read in R with super fast speed.

Value
A reference instance returned by new_reference with signal type determined automatically.
get_projects  Get all possible projects in 'RAVE' directory

Description
Get all possible projects in 'RAVE' directory

Usage
get_projects(refresh = TRUE)

Arguments
refresh  whether to refresh the cache; default is true

Value
characters of project names

get_val2  Get value or return default if invalid

Description
Get value or return default if invalid

Usage
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 ------------------------
# no key, returns x if x is valid
get_val2(x)

get_val2(x, 'a', 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**

*Returns all names contained in 'HDF5' file*

**Description**

Returns all names contained in 'HDF5' file

**Usage**

```r
h5_names(file)
```

**Arguments**

- `file`, 'HDF5' file path

**Value**

characters, data set names
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
h5_valid(file, mode = c("r", "w"), close_all = FALSE)

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
logical whether the file can be opened.

Examples

```r
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
```
**import_electrode_table**

*Import electrode table into subject meta folder*

**Description**

Import electrode table into subject meta folder

**Usage**

import_electrode_table(path, subject, use_fs = NA, dry_run = FALSE, ...)

**Arguments**

- **path** path of table file, must be a `csv` file
- **subject** 'RAVE' subject ID or instance
- **use_fs** whether to use 'FreeSurfer' files to calculate other coordinates
- **dry_run** whether to dry-run the process; if true, then the table will be generated but not saved to subject’s meta folder
- **...** passed to `read.csv`

**Value**

Nothing, the electrode information will be written directly to the subject’s meta directory

---

**install_modules**

*Install 'RAVE' modules*

**Description**

Install 'RAVE' modules

**Usage**

install_modules(modules, dependencies = FALSE)

**Arguments**

- **modules** a vector of characters, repository names; default is to automatically determined from a public registry
- **dependencies** whether to update dependent packages; default is false

**Value**

nothing
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 == ""

---

is.zerolenth  
*Check If Input Has Zero Length*

**Description**
Check If Input Has Zero Length

**Usage**
is.zerolenth(x)

**Arguments**
- **x**  
  input data: a vector, list, or array

**Value**
whether x has zero length
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**

```r
# 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

```r
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

```r
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(...)

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

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

# Check whether the data is identical
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$new()
- LazyH5$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
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, ...)
LazyH5

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

Author(s)
Zhengjia Wang

Examples
# 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 Definitions of reference with 'LFP' signal type

Description

Please use a safer new_electrode function to create instances. This documentation is to describe the member methods of the electrode class LFP_electrode

Super class

raveio::RAVEAbstarctElectrode -> LFP_electrode

Active bindings

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
power_sample_rate power/phase sample rate
preprocess_info preprocess information
power_file path to power 'HDF5' file
phase_file path to phase 'HDF5' file
voltage_file path to voltage 'HDF5' file

Methods

Public methods:
- LFP_electrode$print()
- LFP_electrode$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_raw_voltage()
LFP_electrode

- LFP_electrode$load_data()
- LFP_electrode$load_blocks()
- LFP_electrode$clear_cache()
- LFP_electrode$clear_memory()
- LFP_electrode$clone()

Method `print()`: print electrode summary

Usage:
LFP_electrode$print()

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, quiet = FALSE)

Arguments:
subject, number, quiet see constructor in RAVEAbstractElectrode

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

Usage:
LFP_electrode$.load_noref_wavelet(reload = FALSE)

Arguments:
reload whether to reload cache

Returns: if the reference number if NULL or 'noref', then returns 0, otherwise returns a FileArray-class

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

Usage:
LFP_electrode$.load_noref_voltage(reload = FALSE)

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

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

Usage:
LFP_electrode$.load_wavelet(
  type = c("power", "phase", "wavelet-coefficient"),
  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_raw_voltage(): load raw voltage (no process)

Usage:
LFP_electrode$.load_raw_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", "raw-voltage")
)

Arguments:
type  data type such as "power", "phase", "voltage", "wavelet-coefficient", and "raw-voltage". For "power", "phase", and "wavelet-coefficient", 'Wavelet' transforms are required. For "voltage", 'Notch' filters must be applied. All these types except for "raw-voltage" will be referenced. For "raw-voltage", no reference will be performed since the data will be the "raw" signal (no processing).

Method load_blocks(): load electrode block-wise data (with no reference), useful when epoch is absent

Usage:
LFP_electrode$load_blocks(
  blocks,
  type = c("power", "phase", "voltage", "wavelet-coefficient", "raw-voltage"),
  simplify = TRUE
)

Arguments:
blocks  session blocks
type  data type such as "power", "phase", "voltage", "raw-voltage" (with no filters applied, as-is from imported), "wavelet-coefficient". Note that if type is "raw-voltage", then the data only needs to be imported; for "voltage" data, 'Notch' filters must be applied; for all other types, 'Wavelet' transforms are required.
simplify  whether to simplify the result
Returns: If simplify is enabled, and only one block is loaded, then the result will be a vector (type="voltage") or a matrix (others), otherwise the result will be a named list where the names are the blocks.

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

# Download subject demo/DemoSubject
subject <- as_rave_subject("demo/DemoSubject", strict = FALSE)
if(dir.exists(subject$path)) {

# Electrode 14 in demo/DemoSubject
e <- new_electrode(subject = subject, number = 14, signal_type = "LFP")

# Load CAR reference "ref_13-16,24"
ref <- new_reference(subject = subject, number = "ref_13-16,24",
signal_type = "LFP")
e$set_reference(ref)

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

# Set loading window
e$trial_intervals <- list(c(-1, 2))

# Preview
print(e)
# Now epoch power
power <- e$load_data("power")
names(dimnames(power))

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

# Draw baseline
tempfile <- tempfile()
bl <- power_baseline(power, baseline_windows = c(-1, 0),
                      method = "decibel", filebase = tempfile)
collapsed_power <- collapse2(bl, keep = c(2,1))
# Visualize
dname <- dimnames(bl)
image(collapsed_power, x = dname$Time, y = dname$Frequency,
      xlab = "Time (s)", ylab = "Frequency (Hz)",
      main = "Mean power over trial (Baseline: -1~0 seconds)",
      sub = glue(\"Electrode \{e$number\} (Reference: \{ref$number\})\"))
abline(v = 0, lty = 2, col = "blue")
text(x = 0, y = 20, "Audio onset", col = "blue", cex = 0.6)

# clear cache on hard disk
e$clear_cache()
ref$clear_cache()

---

**LFP_reference**  
*Definitions of reference with 'LFP' signal type*

**Description**

Please use a safer `new_reference` function to create instances. This documentation is to describe the member methods of the electrode class `LFP_reference`.

**Super class**

`raveio::RAVEAbstractElectrode` -> `LFP_reference`

**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
- `power_sample_rate`: power/phase sample rate
preprocess_info preprocess information
power_file path to power 'HDF5' file
phase_file path to phase 'HDF5' file
voltage_file path to voltage 'HDF5' file

Methods

Public methods:

• LFP_reference$print()
• LFP_reference$set_reference()
• LFP_reference$new()
• LFP_reference$.load_noref_wavelet()
• LFP_reference$.load_noref_voltage()
• LFP_reference$.load_wavelet()
• LFP_reference$.load_voltage()
• LFP_reference$load_data()
• LFP_reference$load_blocks()
• LFP_reference$clear_cache()
• LFP_reference$clear_memory()
• LFP_reference$clone()

Method print(): print reference summary

Usage:
LFP_reference$print()

Method set_reference(): set reference for current electrode

Usage:
LFP_reference$set_reference(reference)

Arguments:
reference either NULL or LFP_electrode instance

Method new(): constructor

Usage:
LFP_reference$new(subject, number, quiet = FALSE)

Arguments:
subject, number, quiet see constructor in RAVEAbstractElectrode

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

Usage:
LFP_reference$.load_noref_wavelet(reload = FALSE)

Arguments:
reload whether to reload cache
Returns: if the reference number is NULL or 'noref', then returns 0, otherwise returns a FileArray-class

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

Usage:
LFP_reference$.load_noref_voltage(reload = FALSE)

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

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

Usage:
LFP_reference$.load_wavelet(
  type = c("power", "phase", "wavelet-coefficient"),
  reload = FALSE
)

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

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

Usage:
LFP_reference$.load_voltage(reload = FALSE)

Arguments:
reload whether to reload cache

Method load_data(): method to load electrode data

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

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

Method load_blocks(): load electrode block-wise data (with reference), useful when epoch is absent

Usage:
LFP_reference$.load_blocks(
  blocks,
  type = c("power", "phase", "voltage", "wavelet-coefficient"),
  simplify = TRUE
)

Arguments:
blocks session blocks
type data type such as "power", "phase", "voltage", "wavelet-coefficient". Note that if type is voltage, then 'Notch' filters must be applied; otherwise 'Wavelet' transforms are required.

simplify whether to simplify the result

Returns: If simplify is enabled, and only one block is loaded, then the result will be a vector (type="voltage") or a matrix (others), otherwise the result will be a named list where the names are the blocks.

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

*Usage:*

LFP_reference$clear_cache(...)

*Arguments:*

... ignored

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

*Usage:*

LFP_reference$clear_memory(...)

*Arguments:*

... ignored

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

*Usage:*

LFP_reference$clone(deep = FALSE)

*Arguments:*

depth Whether to make a deep clone.

**Examples**

```r
## Not run:

# Download subject demo/DemoSubject

subject <- as_rave_subject("demo/DemoSubject")

# Electrode 14 as reference electrode (Bipolar referencing)
e <- new_reference(subject = subject, number = "ref_14",
signal_type = "LFP")

# Reference "ref_13-16,24" (CAR or white-matter reference)
ref <- new_reference(subject = subject, number = "ref_13-16,24",
signal_type = "LFP")

ref

# Set epoch
e$set_epoch(epoch = 'auditory_onset')
```
# Set loading window
e$trial_intervals <- list(c(-1, 2))

# Preview
print(e)

# Now epoch power
power <- e$load_data("power")
names(dimnames(power))

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

# clear cache on hard disk
e$clear_cache()

## End(Not run)

---

**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

# 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_need_transpose`: 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.

---

```r
load_h5(file, name, read_only = TRUE, ram = FALSE, quiet = FALSE)
```

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)
```
load_meta2

Load 'RAVE' subject meta data

Description

Load 'RAVE' subject meta data

Usage

load_meta2(meta_type, project_name, subject_code, subject_id, meta_name)
load_yaml

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

fastmap2, save_yaml, read_yaml, write_yaml
**module_add**

Add new ‘RAVE’ (2.0) module to current project

**Usage**

```
module_add(
    module_id,
    module_label,
    path = ".",
    type = c("default", "bare"),
    ...,  # additional configurations to the module such as 'order', 'group', 'badge'
    pipeline_name = module_id,
    overwrite = FALSE
)
```

**Arguments**

- `module_id`  
  module ID to create, must be unique
- `module_label`  
  a friendly label to display in the dashboard
- `path`  
  project root path; default is current directory
- `type`  
  template to choose, options are 'default' and 'bare'
- `pipeline_name`  
  the pipeline name to create along with the module; default is identical to `module_id`
- `overwrite`  
  whether to overwrite existing module if module with same ID exists; default is `false`

**Value**

Nothing.

---

**module_registry**

'RAVE' module registry

**Description**

Create, view, or reserve the module registry
Usage

```r
module_registry(
  title,
  repo,
  modules,
  authors,
  url = sprintf("https://github.com/%s", repo)
)
```

```r
module_registry2(repo, description)
```

```r
get_modules_registries(update = NA)
```

```r
add_module_registry(title, repo, modules, authors, url, dry_run = FALSE)
```

Arguments

- **title**: title of the registry, usually identical to the description title in 'DESCRIPTION' or RAVE-CONFIG file
- **repo**: 'Github' repository
- **modules**: characters of module ID, must only contain letters, digits, underscore, dash; must not be duplicated with existing registered modules
- **authors**: a list of module authors; there must be one and only one author with 'cre' role (see `person`). This author will be considered maintainer, who will be in charge if editing the registry
- **url**: the web address of the repository
- **description**: path to 'DESCRIPTION' or RAVE-CONFIG file
- **update**: whether to force updating the registry
- **dry_run**: whether to generate and preview message content instead of opening an email link

Details

A 'RAVE' registry contains the following data entries: repository title, name, 'URL', authors, and a list of module IDs. 'RAVE' requires that each module must use a unique module ID. It will cause an issue if two modules share the same ID. Therefore 'RAVE' maintains a public registry list such that the module maintainers can register their own module ID and prevent other people from using it.

To register your own module ID, please use `add_module_registry` to validate and send an email to the 'RAVE' development team.

Value

- a registry object, or a list of registries
new_electrode

Examples

library(raveio)

# get current registries
get_modules_registries(FALSE)

# create your own registry
module_registry(
  repo = "dipterix/rave-pipelines",
  title = "A Collection of 'RAVE' Builtin Pipelines",
  authors = list(
    list("Zhengjia", "Wang", role = c("cre", "aut"),
      email = "dipterix@rave.wiki"),
    modules = "brain_viewer"
  )
)

# If your repository is on Github and RAVE-CONFIG file exists
module_registry2("dipterix/rave-pipelines")

# send a request to add your registry
if(interactive()) {
  reg <- module_registry2("dipterix/rave-pipelines")
  add_module_registry(reg)
}

new_electrode(subject, number, signal_type, ...)
new_reference(subject, number, signal_type, ...)

Arguments

subject characters, or a RAVESubject instance
number integer in new_electrode, or characters in new_reference; see 'Details' and 'Examples'

Description

Create new electrode channel instance or a reference signal instance

Usage

new_electrode(subject, number, signal_type, ...)
new_reference(subject, number, signal_type, ...)

new_electrode

Create new electrode channel instance or a reference signal instance

Description

Create new electrode channel instance or a reference signal instance

Usage

new_electrode(subject, number, signal_type, ...)
new_reference(subject, number, signal_type, ...)

Arguments

subject characters, or a RAVESubject instance
number integer in new_electrode, or characters in new_reference; see 'Details' and 'Examples'
new_electrode

signal_type  signal type of the electrode or reference; can be automatically inferred, but it is highly recommended to specify a value; see SIGNAL_TYPES

... other parameters passed to class constructors, respectively

Details

In new_electrode, number should be a positive valid integer indicating the electrode number. In new_reference, number can be one of the followings:

'noref', or NULL no reference is needed

'ref_X' where 'X' is a single number, then the reference is another existing electrode; this could occur in bipolar-reference cases

'ref_XXX' 'XXX' is a combination of multiple electrodes that can be parsed by parse_svec. This could occur in common average reference, or white matter reference. One example is 'ref_13-16,24', meaning the reference signal is an average of electrode 13, 14, 15, 16, and 24.

Value

Electrode or reference instances that inherit RAVEAbstractElectrode class

Examples

```r
## Not run:

# Download subject demo/DemoSubject (~500 MB)
# Electrode 14 in demo/DemoSubject
subject <- as_rave_subject("demo/DemoSubject")
e <- new_electrode(subject = subject, number = 14, signal_type = "LFP")

# Load CAR reference "ref_13-16,24"
ref <- new_reference(subject = subject, number = "ref_13-16,24",
 signal_type = "LFP")
e$set_reference(ref)

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

# Set loading window
e$trial_intervals <- list(c(-1, 2))

# Preview
print(e)

# Now epoch power
power <- e$load_data("power")
names(dimnames(power))

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

# Draw baseline
tempfile <- tempfile()
bl <- power_baseline(power, baseline_windows = c(-1, 0),
    method = "decibel", filebase = tempfile)
collapsed_power <- collapse2(bl, keep = c(2,1))
# Visualize
dname <- dimnames(bl)
image(collapsed_power, x = dname$Time, y = dname$Frequency,
    xlab = "Time (s)", ylab = "Frequency (Hz)",
    main = "Mean power over trial (Baseline: -1~0 seconds)",
    sub = glue('Electrode {e$number} (Reference: {ref$number})'))
abline(v = 0, lty = 2, col = "blue")
text(x = 0, y = 20, "Audio onset", col = "blue", cex = 0.6)

# clear cache on hard disk
e$clear_cache()
ref$clear_cache()

## End(Not run)

pipeline

Creates 'RAVE' pipeline instance

Description

Set pipeline inputs, execute, and read pipeline outputs

Usage

pipeline(
    pipeline_name,
    settings_file = "settings.yaml",
    paths = pipeline_root()
)

Arguments

pipeline_name the name of the pipeline, usually title field in the 'DESCRIPTION' file, or the pipeline folder name (if description file is missing)
settings_file the name of the settings file, usually stores user inputs
paths the paths to search for the pipeline, usually the parent directory of the pipeline; default is pipeline_root, which only search for pipelines that are installed or in current working directory.
Value

A `PipelineTools` instance

Examples

```r
if(interactive()) {

library(raveio)

# ------------ Set up a bare minimal example pipeline ------------
pipeline_path <- pipeline_create_template(
  root_path = tempdir(), pipeline_name = "raveio_demo",
  overwrite = TRUE, activate = FALSE, template_type = "rmd-bare")

save_yaml(list(
  n = 100, pch = 16, col = "steelblue"
), file = file.path(pipeline_path, "settings.yaml"))

pipeline_build(pipeline_path)

rmarkdown::render(input = file.path(pipeline_path, "main.Rmd"),
  output_dir = pipeline_path,
  knit_root_dir = pipeline_path,
  intermediates_dir = pipeline_path, quiet = TRUE)

utils::browseURL(file.path(pipeline_path, "main.html"))

# --------------------- Example starts ------------------------

pipeline <- pipeline("raveio_demo", paths = tempdir())

pipeline$run("plot_data")

# Run again and you will see some targets are skipped
pipeline$set_settings(pch = 2)

pipeline$run("plot_data")

head(pipeline$read("input_data"))

# or use
pipeline[c("n", "pch", "col")]

pipeline[-c("input_data")]

pipeline$target_table

pipeline$result_table

pipeline$progress("details")

# --------------------- Clean up ------------------------
unlink(pipeline_path, recursive = TRUE)
```
PipelineResult

} 

PipelineResult  

Pipeline result object

Description
 Pipeline result object
 Pipeline result object

Public fields

progressor  progress bar object, usually generated from progress2
promise   a promise instance that monitors the pipeline progress
verbose   whether to print warning messages
names  names of the pipeline to build
async_callback  function callback to call in each check loop; only used when the pipeline is running in async=TRUE mode
check_interval  used when async=TRUE in pipeline_run, interval in seconds to check the progress

Active bindings

variables  target variables of the pipeline
variable_descriptions  readable descriptions of the target variables
valid  logical true or false whether the result instance hasn’t been invalidated
status  result status, possible status are ‘initialize’, ‘running’, ‘finished’, ‘canceled’, and ‘errored’. Note that ‘finished’ only means the pipeline process has been finished.
process  (read-only) process object if the pipeline is running in ‘async’ mode, or NULL; see r_bg.

Methods

Public methods:

• PipelineResult$validate()
• PipelineResult$invalidate()
• PipelineResult$get_progress()
• PipelineResult$new()
• PipelineResult$run()
• PipelineResult$await()
• PipelineResult$print()
• PipelineResult$get_values()
• PipelineResult$clone()
Method `validate()`: check if result is valid, raises errors when invalidated

Usage:
PipelineResult$validate()

Method `invalidate()`: invalidate the pipeline result

Usage:
PipelineResult$invalidate()

Method `get_progress()`: get pipeline progress

Usage:
PipelineResult$get_progress()

Method `new()`: constructor (internal)

Usage:
PipelineResult$new(path = character(0L), verbose = FALSE)

Arguments:
path  pipeline path
verbose whether to print warnings

Method `run()`: run pipeline (internal)

Usage:
PipelineResult$run(
  expr,
  env = parent.frame(),
  quoted = FALSE,
  async = FALSE,
  process = NULL
)

Arguments:
expr  expression to evaluate
env  environment of expr
quoted  whether expr has been quoted
async  whether the process runs in other sessions
process  the process object inherits process, will be inferred from expr if process=NULL, and will raise errors if cannot be found

Method `await()`: wait until some targets get finished

Usage:
PipelineResult$await(names = NULL, timeout = Inf)

Arguments:
names  target names to wait, default is NULL, i.e. to wait for all targets that have been scheduled
timeout  maximum waiting time in seconds

Returns:  TRUE if the target is finished, or FALSE if timeout is reached
**Method print()**: print method

*Usage:*

```r
PipelineResult$print()
```

**Method get_values()**: get results

*Usage:*

```r
PipelineResult$get_values(names = NULL, ...)
```

*Arguments:*

- `names` the target names to read
- `...` passed to `codelinkpipeline_read`

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

*Usage:*

```r
PipelineResult$clone(deep = FALSE)
```

*Arguments:*

- `deep` Whether to make a deep clone.

---

**PipelineTools**

*Class definition for pipeline tools*

---

**Description**

Class definition for pipeline tools

Class definition for pipeline tools

**Active bindings**

- `settings_path` absolute path to the settings file
- `target_table` table of target names and their descriptions
- `result_table` summary of the results, including signatures of data and commands
- `pipeline_path` the absolute path of the pipeline
- `pipeline_name` the code name of the pipeline

**Methods**

**Public methods:**

- `PipelineTools$new()`
- `PipelineTools$set_settings()`
- `PipelineTools$get_settings()`
- `PipelineTools$read()`
- `PipelineTools$run()`
- `PipelineTools$eval()`
- `PipelineTools$progress()`
- `PipelineTools$attach()`
- `PipelineTools$clean()`
- `PipelineTools$save_data()`
- `PipelineTools$load_data()`
- `PipelineTools$clone()`

**Method new()**: construction function

**Usage:**
```
PipelineTools$new(
    pipeline_name,
    settings_file = "settings.yaml",
    paths = pipeline_root()
)
```

**Arguments:**
- `pipeline_name` name of the pipeline, usually in the pipeline 'DESCRIPTION' file, or pipeline folder name
- `settings_file` the file name of the settings file, where the user inputs are stored
- `paths` the paths to find the pipeline, usually the parent folder of the pipeline; default is `pipeline_root()`

**Method set_settings()**: set inputs

**Usage:**
```
PipelineTools$set_settings(..., .list = NULL)
```

**Arguments:**
- `...` named list of inputs; all inputs should be named, otherwise errors will be raised
- `.list` named list of inputs; all inputs should be named, otherwise errors will be raised

**Method get_settings()**: get current inputs

**Usage:**
```
PipelineTools$get_settings(key, default = NULL, constraint)
```

**Arguments:**
- `key` the input name; default is missing, i.e., to get all the settings
- `default` default value if not found
- `constraint` the constraint of the results; if input value is not from `constraint`, then only the first element of `constraint` will be returned.

**Returns**: The value of the inputs, or a list if `key` is missing

**Method read()**: read intermediate variables

**Usage:**
```
PipelineTools$read(var_names, ifnotfound = NULL, ...)
```

**Arguments:**
- `var_names` the target names, can be obtained via `x$target_table` member; default is missing, i.e., to read all the intermediate variables
- `ifnotfound` variable default value if not found
... other parameters passing to pipeline_read

**Returns:** The values of the targets

**Method** run(): run the pipeline

**Usage:**

```r
PipelineTools$run(
  names = NULL,
  async = FALSE,
  as.promise = async,
  scheduler = c("none", "future", "clustermq"),
  type = c("smart", "callr", "vanilla"),
  envir = new.env(parent = globalenv()),
  callr_function = NULL,
  ...
)
```

**Arguments:**

- **names**  pipeline variable names to calculate; default is to calculate all the targets
- **async**  whether to run asynchronous in another process
- **as.promise**  whether to return a `PipelineResult` instance
- **scheduler, type, envir, callr_function, ...**  passed to `pipeline_run` if `as.promise` is true, otherwise these arguments will be passed to `pipeline_run_bare`

**Returns:** A `PipelineResult` instance if `as.promise` or `async` is true; otherwise a list of values for input names

**Method** eval(): run the pipeline in order; unlike `$run()`, this method does not use the targets infrastructure, hence the pipeline results will not be stored, and the order of names will be respected.

**Usage:**

```r
PipelineTools$eval(names, env = parent.frame(), clean = TRUE)
```

**Arguments:**

- **names**  pipeline variable names to calculate; must be specified
- **env**  environment to evaluate and store the results
- **clean**  whether to evaluate without polluting `env`

**Method** progress(): get progress of the pipeline

**Usage:**

```r
PipelineTools$progress(method = c("summary", "details"))
```

**Arguments:**

- **method**  either 'summary' or 'details'

**Returns:** A table of the progress

**Method** attach(): attach pipeline tool to environment (internally used)

**Usage:**

```r
PipelineTools$attach(env)
```
Arguments:
env an environment

Method clean(): clean all or part of the data store

Usage:
PipelineTools$clean(
  destroy = c("all", "cloud", "local", "meta", "process", "progress", "objects", 
    "scratch", "workspaces"),
  ask = FALSE
)

Arguments:
destroy, ask see tar_destroy

Method save_data(): save data to pipeline data folder

Usage:
PipelineTools$save_data(
  data,
  name,
  format = c("json", "yaml", "csv", "fst", "rds"),
  overwrite = FALSE,
  ...
)

Arguments:
data R object
name the name of the data to save, must start with letters
format serialize format, choices are 'json', 'yaml', 'csv', 'fst', 'rds'; default is 'json'.
  To save arbitrary objects such as functions or environments, use 'rds'
overwrite whether to overwrite existing files; default is no
... passed to saver functions

Returns: the saved file path

Method load_data(): load data from pipeline data folder

Usage:
PipelineTools$load_data(
  name,
  error_if_missing = TRUE,
  default_if_missing = NULL,
  format = c("auto", "json", "yaml", "csv", "fst", "rds"),
  ...
)

Arguments:
name the name of the data
error_if_missing whether to raise errors if the name is missing
default_if_missing default values to return if the name is missing
format  the format of the data, default is automatically obtained from the file extension
... passed to loader functions

Returns:  the data if file is found or a default value

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

Usage:
PipelineTools$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.

See Also
pipeline

pipeline_install  Install 'RAVE' pipelines

Description
Install 'RAVE' pipelines

Usage

pipeline_install_local(
  src,
  to = c("default", "custom", "workdir", "tempdir"),
  upgrade = FALSE,
  force = FALSE,
  ...
)
pipeline_install_github(
  repo,
  to = c("default", "custom", "workdir", "tempdir"),
  upgrade = FALSE,
  force = FALSE,
  ...
)

Arguments

src  pipeline directory

to  installation path; choices are 'default', 'custom', 'workdir', and 'tempdir'. Please specify pipeline root path via pipeline_root when 'custom' is used.

upgrade  whether to upgrade the dependence; default is FALSE for stability, however, it is highly recommended to upgrade your dependencies
Pipeline settings get set

### Description

Get or change pipeline input parameter settings

### Usage

```r
description

pipeline_settings_set(..., pipeline_settings_path = file.path(Sys.getenv("RAVE_PIPELINE", "."), "settings.yaml")

pipeline_settings_get(key, default = NULL, constraint = NULL, pipeline_settings_path = file.path(Sys.getenv("RAVE_PIPELINE", "."), "settings.yaml")
```

### Arguments

- `pipeline_settings_path`: the settings file of the pipeline, must be a `yaml` file; default is `settings.yaml` in the current pipeline
- `key, ...`: the character key(s) to get or set
- `default`: the default value is key is missing
- `constraint`: the constraint of the resulting value; if not NULL, then result must be within the constraint values, otherwise the first element of `constraint` will be returned. This is useful to make sure the results stay within given options

### Value

- `pipeline_settings_set` returns a list of all the settings.
- `pipeline_settings_get` returns the value of given key.
power_baseline

Calculate power baseline

Description

Calculate power baseline

Usage

power_baseline(
  x,
  baseline_windows,
  method = c("percentage", "sqrt_percentage", "decibel", "zscore", "sqrt_zscore"),
  units = c("Trial", "Frequency", "Electrode"),
  ...
)

## S3 method for class 'rave_prepare_power'

power_baseline(
  x,
  baseline_windows,
  method = c("percentage", "sqrt_percentage", "decibel", "zscore", "sqrt_zscore"),
  units = c("Frequency", "Trial", "Electrode"),
  electrodes,
  ...
)

## S3 method for class 'FileArray'

power_baseline(
  x,
  baseline_windows,
  method = c("percentage", "sqrt_percentage", "decibel", "zscore", "sqrt_zscore"),
  units = c("Frequency", "Trial", "Electrode"),
  filebase = NULL,
  ...
)

## S3 method for class 'array'

power_baseline(
  x,
  baseline_windows,
  method = c("percentage", "sqrt_percentage", "decibel", "zscore", "sqrt_zscore"),
  units = c("Trial", "Frequency", "Electrode"),
  ...
)

## S3 method for class 'ECoGTensor'

...
power_baseline(
  x,
  baseline_windows,
  method = c("percentage", "sqrt_percentage", "decibel", "zscore", "sqrt_zscore"),
  units = c("Trial", "Frequency", "Electrode"),
  filebase = NULL,
  hybrid = TRUE,
  ...
)

Arguments

x R array, filearray, ECoGTensor, or 'rave_prepare_power' object created by
  prepare_subject_power.

baseline_windows list of baseline window (intervals)

method baseline method; choices are 'percentage', 'sqrt_percentage', 'decibel',
  'zscore', 'sqrt_zscore'; see 'Details' in baseline_array

units the unit of the baseline; see 'Details'

... passed to other methods

electrodes the electrodes to be included in baseline calculation; for power repository object
  produced by prepare_subject_power only; default is all available electrodes
  in each of signal_types

filebase where to store the output; default is NULL and is automatically determined

hybrid whether the array will be

Details

The arrays must be four-mode tensor and must have valid named dimnames. The dimension names
must be 'Trial', 'Frequency', 'Time', 'Electrode', case sensitive.

The baseline_windows determines the baseline windows that are used to calculate time-points of
baseline to be included. This can be one or more intervals and must pass the validation function
validate_time_window.

The units determines the unit of the baseline. It can be one or more of 'Trial', 'Frequency',
'Electrode'. The default value is all of them, i.e., baseline for each combination of trial, fre-
quency, and electrode. To share the baseline across trials, please remove 'Trial' from units. To
calculate baseline that should be shared across electrodes (e.g. in some mini-electrodes), remove
'Electrode' from the units.

Value

Usually the same type as the input: for arrays, filearray, or ECoGTensor, the outputs are also the
same type with the same dimensions; for 'rave_prepare_power' repositories, the results will be
stored in its 'baselined' element; see 'Examples'.
Examples

```r
## Not run:
# The following code need to download additional demo data
# Please see https://rave.wiki/ for more details

library(raveio)
repo <- prepare_subject_power(
  subject = "demo/DemoSubject",
  time_windows = c(-1, 3),
  electrodes = c(14, 15))

##### Direct baseline on the repository
power_baseline(x = repo, method = "decibel",
  baseline_windows = list(c(-1, 0), c(2, 3)))
power_mean <- repo$power$baselined$collapse(
  keep = c(2,1), method = "mean")
image(power_mean, x = repo$time_points, y = repo$frequency,
  xlab = "Time (s)", ylab = "Frequency (Hz)",
  main = "Mean power over trial (Baseline: -1~0 & 2~3)"
abline(v = 0, lty = 2, col = "blue")
text(x = 0, y = 20, "Aud-Onset", col = "blue", cex = 0.6)

##### Alternatively, baseline on electrode instances
baselined <- lapply(repo$power$data_list, function(inst) {
  re <- power_baseline(inst, method = "decibel",
    baseline_windows = list(c(-1, 0), c(2, 3)))
  collapse2(re, keep = c(2,1), method = "mean")
})
power_mean2 <- (baselined[[1]] + baselined[[2]]) / 2

# Same with precision difference
max(abs(power_mean2 - power_mean)) < 1e-6

## End(Not run)
```

Description

Prepare 'RAVE' single-subject data

Usage

```r
prepare_subject_bare0()
```
prepare_subject_bare0

```r
prepare_subject_bare(
  subject,
  electrodes,
  reference_name,
  ..., 
  quiet = TRUE,
  repository_id = NULL
)
```

```r
prepare_subject_with_epoch(
  subject,
  electrodes,
  reference_name,
  ..., 
  epoch_name,
  time_windows,
  env = parent.frame(),
  ...
)
```

```r
prepare_subject_power(
  subject,
  electrodes,
  reference_name,
  epoch_name,
  time_windows,
  signal_type = c("LFP"),
  env = parent.frame(),
  verbose = TRUE,
  ...
)
```

```r
prepare_subject_with_blocks(
  subject,
  electrodes,
  reference_name,
  blocks,
  signal_type = "LFP",
  time_frequency = signal_type == "LFP",
  env = parent.frame(),
  repository_id = NULL,
  ...
)
```
prepare_subject_raw_voltage_with_epoch(
    subject,
    electrodes,
    epoch_name,
    time_windows,
    ..., quiet = TRUE,
    repository_id = NULL
)

prepare_subject_voltage_with_epoch(
    subject,
    electrodes,
    epoch_name,
    time_windows,
    reference_name,
    ..., quiet = TRUE,
    repository_id = NULL
)

Arguments

subject character of project and subject, such as "demo/YAB", or RAVESubject instance
electrodes integer vector of electrodes, or a character that can be parsed by parse_svec
reference_name reference name to be loaded
... ignored
quiet whether to quietly load the data
repository_id used internally
epoch_name epoch name to be loaded, or a RAVEEpoch instance
time_windows a list of time windows that are relative to epoch onset time; need to pass the validation validate_time_window
env environment to evaluate
signal_type electrode signal type (length of one) to be considered; default is 'LFP'. This option rarely needs to change unless you really want to check the power data from other types. For other signal types, check SIGNAL_TYPES
verbose whether to show progress
blocks one or more session blocks to load
time_frequency whether to load time-frequency data when preparing block data

Value

A fastmap2 (basically a list) of objects. Depending on the functions called, the following items may exist in the list:
subject A `RAVESubject` instance
epoch_name Same as input `epoch_name`
epoch A `RAVEEpoch` instance
reference_name Same as input `reference_name`
reference_table A data frame of reference
electrode_table A data frame of electrode information
frequency A vector of frequencies
time_points A vector of time-points
power_list A list of power data of the electrodes
power_dimnames A list of trial indices, frequencies, time points, and electrodes that are loaded

---

**progress_with_logger**  *Enhanced progress with logger message*

**Description**
For best performance, please install ‘ravedash’. This function can replace `progress2`.

**Usage**

```
progress_with_logger(
  title,
  max = 1,
  ..., quiet = FALSE,
  session = shiny::getDefaultReactiveDomain(),
  shiny_auto_close = FALSE,
  outputId = NULL,
  log
)
```

**Arguments**
- `title`, `max`, `...`, `quiet`, `session`, `shiny_auto_close`
  - see `progress2`
- `outputId` will be used if package 'shidashi' is installed, otherwise will be ignored
- `log` function, `NULL`, or missing; default is missing, which will use logger function in the package 'ravedash', or `cat2` if 'ravedash' is not installed. If `log=NULL`, then the message will be suppressed in 'shiny' applications. If a function provided, then the function will be called.

**Value**
A list, see `progress2`
Description

Utility functions for 'RAVE' pipelines, currently designed for internal development use. The infrastructure will be deployed to 'RAVE' in the future to facilitate the "self-expanding" aim. Please check the official 'RAVE' website.

Usage

```r
pipeline_root(root_path)

pipeline_list(root_path = pipeline_root())

pipeline_find(name, root_path = pipeline_root())

pipeline_attach(name, root_path = pipeline_root())

pipeline_run(
  pipe_dir = Sys.getenv("RAVE_PIPELINE", "."),
  scheduler = c("none", "future", "clustermq"),
  type = c("smart", "callr", "vanilla"),
  envir = new.env(parent = globalenv()),
  callr_function = NULL,
  names = NULL,
  async = FALSE,
  check_interval = 0.5,
  progress_quiet = !async,
  progress_max = NA,
  progress_title = "Running pipeline",
  ...
)

pipeline_clean(
  pipe_dir = Sys.getenv("RAVE_PIPELINE", "."),
  destroy = c("all", "cloud", "local", "meta", "process", "progress", "objects",
             "scratch", "workspaces"),
  ask = FALSE
)

pipeline_run_bare(
  pipe_dir = Sys.getenv("RAVE_PIPELINE", "."),
  scheduler = c("none", "future", "clustermq"),
  type = c("smart", "callr", "vanilla"),
  envir = new.env(parent = globalenv()),
  callr_function = NULL,
```
... names = NULL,
)
)
load_targets(...)

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

pipeline_debug(
  quick = TRUE,
  env = parent.frame(),
  pipe_dir = Sys.getenv("RAVE_PIPELINE", "."),
  skip_names
)

pipeline_eval(
  names,
  env = new.env(parent = parent.frame()),
  pipe_dir = Sys.getenv("RAVE_PIPELINE", ".")
)

pipeline_visualize(
  pipe_dir = Sys.getenv("RAVE_PIPELINE", "."),
  glimpse = FALSE,
  targets_only = TRUE,
  shortcut = FALSE,
  zoom_speed = 0.1,
  ...
)

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 =
    "(^data|R|\.R|\.yaml|\.txt|\.csv|\.fst|\.conf|\.json|\.rds)$",
  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", "rmd-bare")
)

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

pipeline_description(file)

pipeline_load_extdata(
  name,
format = c("auto", "json", "yaml", "csv", "fst", "rds"),
error_if_missing = TRUE,
default_if_missing = NULL,
pipe_dir = Sys.getenv("RAVE_PIPELINE", "."),
...
)

pipeline_save_extdata(
data,
name,
format = c("json", "yaml", "csv", "fst", "rds"),
overwrite = FALSE,
pipe_dir = Sys.getenv("RAVE_PIPELINE", "."),
...)

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
pipe_dir where the pipeline directory is; can be set via system environment Sys.getenv("RAVE_PIPELINE"=...)
scheduler how to schedule the target jobs: default is 'none', which is sequential. If you have multiple heavy-weighted jobs that can be scheduled at the same time, you can choose 'future' or 'clustermq'
type how the pipeline should be executed; current choices are "smart" to enable 'future' package if possible, 'callr' to use r, or 'vanilla' to run everything sequentially in the main session.
callr_function function that will be passed to tar_make; will be forced to be NULL if type='vanilla', or r if type='callr'
names the names of pipeline targets that are to be executed; default is NULL, which runs all targets; use pipeline_target_names to check all your available target names.
async whether to run pipeline without blocking the main session
check_interval when running in background (non-blocking mode), how often to check the pipeline progress
progress_title, progress_max, progress_quiet control the progress, see progress2.
... other parameters, targets, etc.
destroy what part of data repository needs to be cleaned
ask whether to ask
quick whether to skip finished targets to save time
env, envir environment to execute the 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
glimpse whether to hide network status when visualizing the pipelines
targets_only whether to return the variable table for targets only; default is true
shortcut whether to display shortcut targets
zoom_speed zoom speed when visualizing the pipeline dependence
method how the progress should be presented; choices are "summary", "details", "custom". If custom method is chosen, then func will be called
rave-pipeline

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 `tar_read`

ifnotfound default values to return if variable is not found

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 `RAVESubject` instance

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

format format of the extended data, default is `json`, other choices are `yaml`, `fst`, `csv`, `rds`

error_if_missing, default_if_missing what to do if the extended data is not found

data extended data to be saved

Value

pipeline_root the root directories of the pipelines

pipeline_list the available pipeline names under `pipeline_root`

pipeline_find the path to the pipeline

pipeline_run a `PipelineResult` instance

load_targets a list of targets to build

pipeline_target_names a vector of characters indicating the pipeline target names

pipeline_visualize a widget visualizing the target dependence structure

pipeline_progress a table of building progress

pipeline_fork a normalized path of the forked pipeline directory

pipeline_read the value of corresponding `var_names`, or a named list if `var_names` has more than one element

pipeline_vartable a table of summaries of the variables; can raise errors if pipeline has never been executed

pipeline_hasname logical, whether the pipeline has variable built

pipeline_watch a basic shiny application to monitor the progress

pipeline_description the list of descriptions of the pipeline or pipeline collection
Validate raw files in 'rave' directory

Description

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

Usage

validate_raw_file(
  subject_code,
  blocks,
  electrodes,
  format,
  data_type = c("continuous"),
  ...
)

IMPORT_FORMATS

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 7.

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' data file 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'+'.eeg/.dat' extensions

**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.
Install and configure 'RAVE' server as background service using shiny-server

Description

Works on 'Linux' and 'Mac' only.

Usage

```r
rave_server_install(
  url = "https://github.com/rstudio/shiny-server/archive/refs/tags/v1.5.18.987.zip"
)

rave_server_configure(
  ports = 17283,
  user = Sys.info()[["user"],
  rave_version = c("1", "2")
)
```

Arguments

- `url` 'URL' to shiny-server 'ZIP' file to download
- `ports` integer vectors or character, indicating the port numbers to host 'RAVE' instances a valid port must be within the range from 1024 to 65535.
- `user` user to run the service as; default is the login user
- `rave_version` internally used; might be deprecated in the future

Value

nothing

Examples

```r
## Not run:
# OS-specific. Please install R package `rpymat` first

# Install rave-server
rave_server_install()

# Let port 17283-17290 to host RAVE instance
rave_server_configure(ports = "17283-17290")
```

## End(Not run)
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

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

Active bindings

type signal type of the electrode, such as 'LFP', 'Spike', and 'EKG'; default is 'Unknown'
power_enabled whether the electrode can be used in power analyses such as frequency, or frequency-time analyses; this usually requires transforming the electrode raw voltage signals using signal processing methods such as 'Fourier', 'wavelet', 'Hilbert', 'multi-taper', etc. If an electrode has power data, then it's power data can be loaded via prepare_subject_power method.
is_reference whether this instance is a reference electrode
location location type of the electrode, see LOCATION_TYPES for details
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
trial_intervals trial intervals relative to epoch onset

Methods

Public methods:

• RAVEAbstarctElectrode$new()
• RAVEAbstarctElectrode$set_reference()
• RAVEAbstarctElectrode$set_epoch()
• RAVEAbstarctElectrode$clear_cache()
• RAVEAbstarctElectrode$clear_memory()
• RAVEAbstractElectrode$load_data()
• RAVEAbstractElectrode$load_blocks()
• RAVEAbstractElectrode$clone()

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

*Usage:*
RAVEAbstractElectrode$new(subject, number, quiet = FALSE)

*Arguments:*
- `subject` character or `RAVESubject` instance
- `number` current electrode number or reference ID
- `quiet` reserved, whether to suppress warning messages

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

*Usage:*
RAVEAbstractElectrode$set_reference(reference)

*Arguments:*
- `reference` NULL or `RAVEAbstractElectrode` 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 `load_blocks()`: load electrode block-wise data (with reference), useful when epoch is absent.

Usage:
RAVEAbstractElectrode$load_blocks(blocks, type, simplify = TRUE)

Arguments:
blocks  session blocks
type  data type such as "power", "phase", "voltage", "wavelet-coefficient".
simplify  whether to simplify the result

Returns: If simplify is enabled, and only one block is loaded, then the result will be a vector (type="voltage") or a matrix (others), otherwise the result will be a named list where the names are the blocks.

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

Usage:
RAVEAbstractElectrode$clone(deep = FALSE)

Arguments:
deep  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
```
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 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(epoch$table)

# query specific trial
old_trial1 <- epoch$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-constants**

*The constant variables*

**Description**

The constant variables

**Usage**

SIGNALTYPES

LOCATIONTYPES

MNI305_to_MNI152

**Format**

An object of class character of length 6.
An object of class character of length 5.
An object of class matrix (inherits from array) with 4 rows and 4 columns.

**Details**

SIGNALTYPES has the following options: 'LFP', 'Spike', 'EKG', 'Audio', 'Photodiode', or 'Unknown'. As of 'raveio' 0.0.6, only 'LFP' (see LFP_electrode) signal type is supported.

LOCATIONTYPES is a list of the electrode location types: 'iEEG' (this includes the next two), 'sEEG' (stereo), 'ECoG' (surface), 'EEG' (scalp), 'Others'. See field 'location' in RAVEAbstractElectrode
MNI305_to_MNI152 is a 4-by-4 matrix converting 'MNI305' coordinates to 'MNI152' space. The difference of these two spaces is: 'MNI305' is an average of 305 human subjects, while 'MNI152' is the average of 152 people. These two coordinates differs slightly. While most of the 'MNI' coordinates reported by 'RAVE' and 'FreeSurfer' are in the 'MNI305' space, many other programs are expecting 'MNI152' coordinates.

---

**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.
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
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
RAVEPreprocessSettings

@freeze_blocks whether to free block, internally used
@freeze_lfp_ecog whether to freeze electrodes that record 'LFP' signals, internally used
@lfp_ecog_sample_rate 'LFP' sample rates, internally used
all_blocks characters, all possible blocks even not included in some projects
raw_path raw data path
raw_path_type raw data path type, 'native' or 'bids'

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 = SIGNAL_TYPES,
add = FALSE
)

Arguments:
electrodes integer vectors
type signal type of electrodes, see `SIGNAL_TYPES`
add whether to add to current settings

Method `set_sample_rates()`: set sample frequency

Usage:
RAVEPreprocessSettings$set_sample_rates(srate, type = SIGNAL_TYPES)

Arguments:
srate sample rate, must be positive number
type electrode type to set sample rate. In ‘rave’, all electrodes with the same signal 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

# 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:**
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
- note_path: path that stores 'RAVE' related subject notes
- 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$meta_data()
- RAVESubject$valid_electrodes()
- RAVESubject$initialize_paths()
- RAVESubject$set_default()
- RAVESubject$get_default()
- RAVESubject$get_epoch()
- RAVESubject$get_reference()
- RAVESubject$get_electrode_table()
- RAVESubject$get_frequency()
- RAVESubject$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
RAVESubject

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 set_default(): set default key-value pair for the subject, used by 'RAVE' modules

Usage:
RAVESubject$set_default(key, value, namespace = "default")

Arguments:
key character
value value of the key
namespace file name of the note (without post-fix)

Returns: The same as value

Method get_default(): get default key-value pairs for the subject, used by 'RAVE' modules

Usage:
RAVESubject$get_default(
  ..., 
  default_if_missing = NULL, 
  simplify = TRUE, 
  namespace = "default"
)

Arguments:
... single key, or a vector of character keys
default_if_missing default value is any key is missing
simplify whether to simplify the results if there is only one key to fetch; default is TRUE
namespace file name of the note (without post-fix)

Returns: A named list of key-value pairs, or if one key is specified and simplify=TRUE, then only the value will be returned.

Method get_epoch(): check and get subject's epoch information

Usage:
RAVESubject$get_epoch(epoch_name, as_table = FALSE, trial_starts = 0)
Arguments:
ePOCH_NAME epoch name, depending on the subject's meta files
as_table whether to convert to data.frame; default is false
trial_starts the start of the trial relative to epoch time; default is 0

Returns: If as_table is FALSE, then returns as RAVEEpoch instance; otherwise returns epoch table; will raise errors when file is missing or the epoch is invalid.

Method get_reference(): check and get subject's reference information

Usage:
RAVESubject$get_reference(reference_name, simplify = FALSE)

Arguments:
reference_name reference name, depending on the subject's meta file settings
simplify whether to only return the reference column

Returns: If simplify is true, returns a vector of reference electrode names, otherwise returns the whole table; will raise errors when file is missing or the reference is invalid.

Method get_electrode_table(): check and get subject's electrode table with electrodes that are load-able

Usage:
RAVESubject$get_electrode_table(
electrodes,
reference_name,
subset = FALSE,
simplify = FALSE
)

Arguments:
electrodes characters indicating integers such as "1-14,20-30", or integer vector of electrode numbers
reference_name see method get_reference
subset whether to subset the resulting data table
simplify whether to only return electrodes

Returns: If simplify is true, returns a vector of electrodes that are valid (or won't be excluded) under given reference; otherwise returns a table. If subset is true, then the table will be subset and only rows with electrodes to be loaded will be kept.

Method get_frequency(): check and get subject's frequency table, time-frequency decomposition is needed.

Usage:
RAVESubject$get_frequency(simplify = TRUE)

Arguments:
simplify whether to simplify as vector

Returns: If simplify is true, returns a vector of frequencies; otherwise returns a table.

Method clone(): The objects of this class are cloneable with this method.
**Usage:**
RAVESubject$clone(deep = FALSE)

**Arguments:**
deep Whether to make a deep clone.

**See Also**
load_meta2

---

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

**Usage**
```r
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()
}
```

**Arguments**

- **path**: path to normalize
- **type**: type of command
- **unset**: default to return if the command is not found
- **error_on_missing**: whether to raise errors if command is missing

**Value**

Normalized path to the command, or unset if command is missing.

---

**rave_directories**  
*Returns a list of 'RAVE' directories*

**Description**

This function is internally used and should not be called directly.

**Usage**

```r
rave_directories(
  subject_code,
  project_name,
  blocks = NULL,
  .force_format = c("", "native", "BIDS")
)
```

**Arguments**

- **subject_code**: 'RAVE' subject code
- **project_name**: 'RAVE' project name
- **blocks**: session or block names, optional
- **.force_format**: format of the data, default is automatically detected.

**Value**

A list of directories
rave_import  
Import data into 'rave' projects

Description

Import files with predefined structures. 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 for tutorials.

Usage

```r
rave_import(
  project_name,
  subject_code,
  blocks,
  electrodes,
  format,
  sample_rate,
  conversion = NA,
  data_type = "LFP",
  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.
- `conversion`: physical unit conversion, choices are NA, V, mV, uV.
- `data_type`: electrode signal type; see `SIGNAL_TYPES`.
- `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.

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_ieeg.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.

---

rave_subject_format_conversion

*Compatibility support for 'RAVE' 1.0 format*

**Description**

Convert 'RAVE' subject generated by 2.0 pipeline such that 1.0 modules can use the data. The subject must have valid electrodes. The data must be imported, with time-frequency transformed to pass the validation before converting.

**Usage**

```
rave_subject_format_conversion(subject, verbose = TRUE, ...)
```

**Arguments**

- `subject` 'RAVE' subject characters, such as 'demo/YAB', or a subject instance generated from RAVESubject
- `verbose` whether to verbose the messages
- `...` ignored, reserved for future use

**Value**

Nothing

---

read-brainvision-eeg  
*Load from 'BrainVision' file*

**Description**

Read in 'eeg' or 'ieeg' data from 'BrainVision' files with .eeg or .dat extensions.
read-brainvision-eeg

Usage

read_eeg_header(file)

read_eeg_data(header, path = NULL)

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

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**

`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**

`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

```r
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](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)
read_mat2(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[]
}
```

---

**read_nsx_nev**

Read 'BlackRock' event and signal files
Description

Current implementation supports minimum 2.3 file specification version. Please contact the package maintainer to add specification configurations if you want us to support older versions.

Usage

```r
read_nsx_nev(
  paths,
  nev_path = NULL,
  header_only = FALSE,
  nev_data = TRUE,
  verbose = TRUE,
  ram = FALSE,
  force_update = FALSE,
  temp_path = file.path(tempdir(), "blackrock-temp")
)
```

Arguments

- **paths**: 'NSx' signal files, usually with file extensions such as '.ns1', '.ns2', '.ns3', '.ns4', '.ns5'.
- **nev_path**: 'NEV' event files, with file extension '.nev'
- **header_only**: whether to load header information only and avoid reading signal arrays
- **nev_data**: whether to load '.nev' comments and 'waveforms'
- **verbose**: whether to print out progress when loading signal array
- **ram**: whether to load signals into the memory rather than storing with `filearray`; default is false
- **force_update**: force updating the channel data even if the headers haven’t changed
- **temp_path**: temporary directory to store the channel data

---

**safe_read_csv**

Read comma separated value files with given column classes

Description

Read comma separated value files with given column classes

Usage

```r
safe_read_csv(
  file,
  header = TRUE,
  sep = ",",
  colClasses = NA,
  skip = 0,
)"
safe_read_csv

quote = "\"",

..., stringsAsFactors = FALSE)

Arguments

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

passed to read.csv

Details

Reading a comma separated value file using builtin function read.csv might result in some unexpected behavior. safe_read_csv does some preprocessing on the format so that it take cares 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)
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
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

Examples
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
- **...**: passed to other LazyH5$save

**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_json

Save or load R object in 'JSON' format

### Description

Save or load R object in 'JSON' format

### Usage

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

```r
load_json(con, ..., map = NULL)
```

### Arguments

- **x**: R object to save
- **con**: file or connection
- **...**: other parameters to pass into `toJSON` or `fromJSON`
- **digits**: number of digits to save
- **pretty**: whether the output should be pretty
- **serialize**: whether to save a serialized version of `x`; see 'Examples'.
- **map**: a map to save the results

### Value

`save_json` returns nothing; `load_json` returns an R object.
Examples

# Serialize
save_json(list(a = 1, b = function(){}))

# use toJSON
save_json(list(a = 1, b = function(){}), serialize = FALSE)

# Demo of using serializer
f1 <- tempfile(fileext = ".json")
save_json(x ~ y + 1, f1)
load_json(f1)
unlink(f1)

save_meta2(data, meta_type, project_name, subject_code)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>data table</td>
</tr>
<tr>
<td>meta_type</td>
<td>see load meta</td>
</tr>
<tr>
<td>project_name</td>
<td>project name</td>
</tr>
<tr>
<td>subject_code</td>
<td>subject code</td>
</tr>
</tbody>
</table>

Value

Either none if no meta matched or the absolute path of file saved.
save_yaml  Write named list to file

Description
Write named list to file

Usage
save_yaml(x, file, ..., sorted = FALSE)

Arguments
x          a named list, fastmap2, or anything that can be transformed into named list via as.list
file, ...  passed to write_yaml
sorted     whether to sort the results by name; default is false

Value
Normalized file path

See Also
fastmap2, load_yaml, read_yaml, write_yaml

Examples

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$subset()
• Tensor$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:*

```r
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:*

```r
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 = 1:10000000, c(3000,300,20),
    dimnames = list(A = 1:3000, B = 1:300, C = 1:20),
    varnames = c("Var A", "Var B", "Var 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
}
```
test_hdspeed

# 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
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)

---

**test_hdspeed**

*Simple hard disk speed test*

**Description**

Simple hard disk speed test

**Usage**

```r
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

\[
time\_diff2(\text{start, end, units = "secs", label = ""})
\]

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

```r
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
```

url_neurosynth

Get 'Neurosynth' website address using 'MNI152' coordinates

Description

Get 'Neurosynth' website address using 'MNI152' coordinates

Usage

url_neurosynth(x, y, z)

Arguments

x, y, z  numerical values: the right-anterior-superior 'RAS' coordinates in 'MNI152' space

Value

'Neurosynth' website address

validate_subject

Validate subject data integrity

Description

Check against existence, validity, and consistency

Arguments

subject  subject ID (character), or RAVESubject instance
method  validation method, choices are 'normal' (default) or 'basic' for fast checks; if set to 'normal', four additional validation parts will be tested (see parts with * in Section 'Value').
verbose  whether to print out the validation messages
version  data version, choices are 1 for 'RAVE' 1.0 data format, and 2 ('RAVE' 2.0 data format); default is 2
Valide_subject

Value

A list of nested validation results. The validation process consists of the following parts in order:

Data paths (paths)
- path the subject’s root folder
- path the subject’s ‘RAVE’ folder (the ‘rave’ folder under the root directory)
- raw_path the subject’s raw data folder
- data_path a directory storing all the voltage, power, phase data (before reference)
- meta_path meta directory containing all the electrode coordinates, reference table, epoch information, etc.
- reference_path a directory storing calculated reference signals
- preprocess_path a directory storing all the preprocessing information
- cache_path (low priority) data caching path
- freesurfer_path (low priority) subject’s ‘FreeSurfer’ directory
- note_path (low priority) subject’s notes
- pipeline_path (low priority) a folder containing all saved pipelines for this subject

Preprocessing information (preprocess)
- electrodes_set whether the subject has a non-empty electrode set
- blocks_set whether the session block length is non-zero
- sample_rate_set whether the raw sampling frequency is set to a valid, proper positive number
- data_imported whether all the assigning electrodes have been imported
- notch_filtered whether all the ‘LFP’ and ‘EKG’ signals have been ‘Notch’ filtered
- has_wavelet whether all the ‘LFP’ signals are wavelet-transformed
- has_reference at least one reference has been generated in the meta folder
- has_epoch at least one epoch file has been generated in the meta folder
- has_electrode_file meta folder has electrodes.csv file

Meta information (meta)
- meta_data_valid this item only exists when the previous preprocess validation is failed or incomplete
- meta_electrode_table the electrodes.csv file in the meta folder has correct format and consistent electrodes numbers to the preprocess information
- meta_reference_xxx (xxx will be replaced with actual reference names) checks whether the reference table contains all electrodes and whether each reference data exists
- meta_epoch_xxx (xxx will be replaced with actual epoch names) checks whether the epoch table has the correct formats and whether there are missing blocks indicated in the epoch files

Voltage data (voltage_data*)
- voltage_preprocessing whether the raw preprocessing voltage data are valid. This includes data lengths are the same within the same blocks for each signal type
- voltage_data whether the voltage data (after ‘Notch’ filters) exist and readable. Besides, the lengths of the data must be consistent with the raw signals
Spectral power and phase (power_phase_data*)
power_data whether the power data exists for all 'LFP' signals. Besides, to pass the validation process, the frequency and time-point lengths must be consistent with the preprocess record.

epoch_tables* One or more sub-items depending on the number of epoch tables. To pass the validation, the event time for each session block must not exceed the actual signal duration. For example, if one session lasts for 200 seconds, it will invalidate the result if a trial onset time is later than 200 seconds.

Reference table (reference_tables*) One or more sub-items depending on the number of reference tables. To pass the validation, the reference data must be valid. The inconsistencies, for example, missing file, wrong frequency size, invalid time-point lengths will result in failure.

validate_time_window Validate time windows to be used

Description
Make sure the time windows are valid intervals and returns a reshaped window list

Usage
validate_time_window(time_windows)

Arguments
time_windows vectors or a list of time intervals

Value
A list of time intervals (ordered, length of 2)

Examples

# Simple time window
validate_time_window(c(-1, 2))

# Multiple windows
validate_time_window(c(-1, 2, 3, 5))

# alternatively
validate_time_window(list(c(-1, 2), c(3, 5)))
validate_time_window(list(list(-1, 2), list(3, 5)))

## Not run:
# Incorrect usage (will raise errors)

# Invalid interval (length must be two for each intervals)
validate_time_window(list(c(-1, 2, 3, 5)))

# Time intervals must be in ascending order
validate_time_window(c(2, 1))

## End(Not run)

table:

<table>
<thead>
<tr>
<th>voltage_baseline</th>
<th>Calculate voltage baseline</th>
</tr>
</thead>
</table>

**Description**

Calculate voltage baseline

**Usage**

```r
voltage_baseline(
  x,
  baseline_windows,
  method = c("percentage", "zscore", "subtract_mean"),
  units = c("Trial", "Electrode"),
  ...
)
```

```r
## S3 method for class 'rave_prepare_subject_raw_voltage_with_epoch'
voltage_baseline(
  x,
  baseline_windows,
  method = c("percentage", "zscore", "subtract_mean"),
  units = c("Trial", "Electrode"),
  electrodes,
  baseline_mean,
  baseline_sd,
  ...
)
```

```r
## S3 method for class 'rave_prepare_subject_voltage_with_epoch'
voltage_baseline(
  x,
  baseline_windows,
  method = c("percentage", "zscore", "subtract_mean"),
  units = c("Trial", "Electrode"),
  ...)
```
voltage_baseline

electrodes,
baseline_mean,
baseline_sd,
...
)

## S3 method for class 'FileArray'
voltage_baseline(  
  x,
  baseline_windows,
  method = c("percentage", "zscore", "subtract_mean"),
  units = c("Trial", "Electrode"),
  filebase = NULL,
  ...
)

## S3 method for class 'array'
voltage_baseline(  
  x,
  baseline_windows,
  method = c("percentage", "zscore", "subtract_mean"),
  units = c("Trial", "Electrode"),
  ...
)

Arguments

x R array, filearray, or 'rave_prepare_power' object created by prepare_subjed_raw_voltage_with_epoch

baseline_windows list of baseline window (intervals)

method baseline method; choices are 'percentage' and 'zscore'; see 'Details' in baseline_array

units the unit of the baseline; see 'Details'

... passed to other methods

electrodes the electrodes to be included in baseline calculation; for power repository object produced by prepare_subject_power only; default is all available electrodes in each of signal_types

baseline_mean, baseline_sd internally used by 'RAVE' repository, provided baseline is not contained in the data. This is useful for calculating the baseline with data from other blocks.

filebase where to store the output; default is NULL and is automatically determined

Details

The arrays must be three-mode tensor and must have valid named dimnames. The dimension names must be 'Trial', 'Time', 'Electrode', case sensitive.
The baseline_windows determines the baseline windows that are used to calculate time-points of baseline to be included. This can be one or more intervals and must pass the validation function validate_time_window.

The units determines the unit of the baseline. It can be either or both of 'Trial', 'Electrode'. The default value is both, i.e., baseline for each combination of trial and electrode.

Value

The same type as the inputs

Examples

```r
## Not run:
# The following code need to download additional demo data
# Please see https://rave.wiki/ for more details

library(raveio)
repo <- prepare_subject_raw_voltage_with_epoch(
  subject = "demo/DemoSubject",
  time_windows = c(-1, 3),
  electrodes = c(14, 15))

##### Direct baseline on repository
voltage_baseline(
  x = repo, method = "zscore",
  baseline_windows = list(c(-1, 0), c(2, 3))
)

voltage_mean <- repo$raw_voltage$baselined$collapse(
  keep = c(1,3), method = "mean")
matplot(voltage_mean, type = "l", lty = 1,
  x = repo$raw_voltage$dimnames$Time,
  xlab = "Time (s)", ylab = "Voltage (z-scored)",
  main = "Mean coltage over trial (Baseline: -1~0 & 2~3)"
)
abline(v = 0, lty = 2, col = "darkgreen")
text(x = 0, y = -0.5, "Aud-Onset ", col = "darkgreen", cex = 0.6, adj = c(1,1))

##### Alternatively, baseline on each electrode channel
voltage_mean2 <- sapply(repo$raw_voltage$data_list, function(inst) {
  re <- voltage_baseline(
    x = inst, method = "zscore",
    baseline_windows = list(c(-1, 0), c(2, 3)))
  rowMeans(re[])
})

# Same with floating difference
max(abs(voltage_mean - voltage_mean2)) < 1e-8

## End(Not run)
```
with_future_parallel  Enable parallel computing provided by 'future' package within the context

Description

Enable parallel computing provided by 'future' package within the context

Usage

with_future_parallel(
    expr,
    env = parent.frame(),
    quoted = FALSE,
    on_failure = "multisession",
    max_workers = NA,
    ...
)

Arguments

expr  the expression to be evaluated
env   environment of the expr
quoted whether expr has been quoted; default is false
on_failure alternative 'future' plan to use if forking a process is disallowed; this usually occurs on 'Windows' machines; see details.
max_workers maximum of workers; default is automatically set by raveio_getopt("max_worker", 1L)
...  additional parameters passing into make_forked_clusters

Details

Some 'RAVE' functions such as prepare_subject_power support parallel computing to speed up. However, the parallel computing is optional. You can enable it by wrapping the function calls within with_future_parallel (see examples).

The default plan is to use 'forked' R sessions. This is a convenient, fast, and relative simple way to create multiple R processes that share the same memories. However, on some machines such as 'Windows' the support has not yet been implemented. In such cases, the plan fall backs to a back-up specified by on_failure. By default, on_failure is 'multisession', a heavier implementation than forking the process, and slightly longer ramp-up time. However, the difference should be marginal for most of the functions.

When parallel computing is enabled, the number of parallel workers is specified by the option raveio_getopt("max_worker", 1L).
**Value**

The evaluation results of `expr`

**Examples**

```r
library(raveio)

demo_subject <- as_rave_subject("demo/DemoSubject", strict = FALSE)

if(dir.exists(demo_subject$path)) {
  with_future_parallel({
    prepare_subject_power("demo/DemoSubject")
  })
}
```
Index

* datasets
  rave-raw-validation, 72
  raveio-constants, 80

add_module_registry (module_registry), 47
as_rave_project, 4
as_rave_subject, 4
as_rave_unit, 5, 114
auto_process_blackrock, 5

backup_file, 6
baseline_array, 62, 119
BlackrockFile, 7

cache_path, 9
cache_root (cache_path), 9
cat2, 10, 66
catgl, 10
clear_cached_files (cache_path), 9
cmd-external (cmd_run_3dAllineate), 11
cmd_afni_home (rave_command_line_path), 92
cmd_dcm2niix (rave_command_line_path), 92
cmd_execute (cmd_run_3dAllineate), 11
cmd_freesurfer_home (rave_command_line_path), 92
cmd_fsl_home (rave_command_line_path), 92
cmd_homebrew (rave_command_line_path), 92
cmd_run_3dAllineate, 11
cmd_run_dcm2niix (cmd_run_3dAllineate), 11
cmd_run_flirt (cmd_run_3dAllineate), 11
cmd_run_recon_all (cmd_run_3dAllineate), 11
collapse, 14
collapse2, 14
collapse_power, 15
cumulate_tuples, 16
convert_blackrock, 17
data.frame, 90
dimnames, 62, 119
dir.create, 18
dir.create2, 18
drop, 30, 33, 111

ECoGTensor, 18, 62
fastmap2, 46, 65, 108
filearray, 21, 62, 102, 119
find_path, 20
fread, 99
freesurfer_brain2, 91
fromJSON, 106
generate_reference, 21
get_modules_registries (module_registry), 47
get_projects, 22
get_val2, 22

h5_names, 23
h5_valid, 24

hdf5r-package, 44
import_electrode_table, 25
IMPORT_FORMATS (rave-raw-validation), 72
install_modules, 25
is.blank, 26
is.zerolenth, 26
is_dry_run (rave_command_line_path), 92
is_valid_ish, 22, 27

join_tensors, 28

LazyFST, 29, 44
LazyH5, 29, 31, 44, 45
LFP_electrode, 34, 80
LFP_reference, 38
load_bids_ieeg_header, 42
load_fst (read-write-fst), 98
load_fst_or_h5, 43
load_h5, 44, 101, 105
load_json (save_json), 106
load_meta2, 45, 89, 91
load_targets (rave-pipeline), 67
load_yaml, 46, 108
LOCATION_TYPES, 75
LOCATION_TYPES (raveio-constants), 80
make_forked_clusters, 121
MNI305_to_MNI152 (raveio-constants), 80
mode, 27, 32
module_add, 47
module_registry, 47
module_registry2 (module_registry), 47
new_electrode, 34, 49
new_reference, 21, 38
new_reference (new_electrode), 49
normalize_commandline_path
  (rave_commandline_path), 92
parse_svec, 50, 65
person, 48
pipeline, 51, 59
pipeline_attach (rave-pipeline), 67
pipeline_build (rave-pipeline), 67
pipeline_clean (rave-pipeline), 67
pipeline_create_subject_pipeline (rave-pipeline), 67
pipeline_create_template, 16
pipeline_create_template (rave-pipeline), 67
pipeline_debug (rave-pipeline), 67
pipeline_description (rave-pipeline), 67
pipeline_eval (rave-pipeline), 67
pipeline_find (rave-pipeline), 67
pipeline_fork (rave-pipeline), 67
pipeline_hasname (rave-pipeline), 67
pipeline_install, 59
pipeline_install_github
  (pipeline_install), 59
pipeline_install_local
  (pipeline_install), 59
pipeline_list (rave-pipeline), 67
pipeline_load_exdata (rave-pipeline), 67
pipeline_progress (rave-pipeline), 67
pipeline_read, 57
pipeline_read (rave-pipeline), 67
pipeline_root, 51, 59
pipeline_root (rave-pipeline), 67
pipeline_run, 53, 57
pipeline_run (rave-pipeline), 67
pipeline_run_bare (rave-pipeline), 67
pipeline_save_exdata (rave-pipeline), 67
pipeline_settings_get
  (pipeline_settings_get_set), 60
pipeline_settings_get_set, 60
pipeline_settings_set
  (pipeline_settings_get_set), 60
pipeline_target_names (rave-pipeline), 67
pipeline_vartable (rave-pipeline), 67
pipeline_visualize (rave-pipeline), 67
pipeline_watch (rave-pipeline), 67
PipelineResult, 53, 57, 71
PipelineTools, 52, 55
power_baseline, 61
prepare_subject_bare
  (prepare_subject_bare0), 63
prepare_subject_bare0, 63
prepare_subject_power, 62, 75, 119, 121
prepare_subject_power
  (prepare_subject_bare0), 63
prepare_subject_raw_voltage_with_epoch, 119
prepare_subject_raw_voltage_with_epoch
  (prepare_subject_bare0), 63
prepare_subject_voltage_with_epoch
  (prepare_subject_bare0), 63
prepare_subject_with_blocks
  (prepare_subject_bare0), 63
prepare_subject_with_epoch
  (prepare_subject_bare0), 63
process, 54
progress2, 53, 66, 70
progress_with_logger, 66
promise, 53
r, 70
r_bg, 53
rave-pipeline, 67
rave-prepare (prepare_subject_bare0), 63
rave-raw-validation, 72
rave-server, 74
rave_brain, 91
rave_command_line_path, 92
rave_directories, 93
rave_import, 94
rave_server_configure (rave-server), 74
rave_server_install (rave-server), 74
rave_subject_format_conversion, 96
RAVEAbstractElectrode, 35, 39, 50, 75, 80
RAVEEpoch, 65, 66, 75, 76, 78, 90
raveio-constants, 80
raveio-option, 81
raveio::RAVEAbstractElectrode, 34, 38
raveio::Tensor, 18
raveio_confpath (raveio-option), 81
raveio_getopt (raveio-option), 81
raveio_resopt (raveio-option), 81
raveio_setopt (raveio-option), 81
RAVEPreprocessSettings, 82, 87
RAVEProject, 4, 85, 87
RAVESubject, 4, 13, 21, 49, 65, 66, 71, 75, 76, 82, 83, 87, 91, 96, 115
read-brainvision-eeg, 96
read-write-fst, 98
read.csv, 25
read_csv_ieeg, 98
read_edf_header, 99
read_edf_signal, 100
read_eeg_data (read-brainvision-eeg), 96
read_eeg_header (read-brainvision-eeg), 96
read_mat, 100
read_mat2 (read_mat), 100
read_nsx_nev, 101
read_yaml, 46, 108
readEdfHeader, 99
readMat, 101
safe_read_csv, 102
safe_write_csv, 104
save_fst (read-write-fst), 98
save_h5, 45, 105
save_json, 106
save_meta2, 107
save_yaml, 46, 108
SIGNAL_TYPES, 50, 65, 84, 94
SIGNAL_TYPES (raveio-constants), 80
storage.mode, 99
system2, 13
tar_destroy, 58
tar_make, 70
tar_progress_summary, 71
tar_read, 71
Tensor, 14, 18, 19, 28, 109
test_hdspeed, 113
time_delta, 114
time_diff2, 114
toJSON, 106
url_neurosynth, 115
validate_raw_file (rave-raw-validation), 72
validate_subject, 115
validate_time_window, 62, 65, 117, 120
voltage_baseline, 118
with_future_parallel, 121
write_fst, 98, 111
write_yaml, 46, 108