Package 'ravecore'

May 9, 2026

Title Core File Structures and Workflows for 'RAVE'

Version 0.1.1

Description Defines storage standard for Read, process, and analyze intracranial electroencephalography and deep-brain stimulation in 'RAVE', a reproducible framework for analysis and visualization of iEEG by Magnotti, Wang, and Beauchamp, (2020, <doi:10.1016/j.neuroimage.2020.117341>). Supports brain imaging data structure (BIDS) <https://bids.neuroimaging.io> and native file structure to ingest signals from 'Matlab' data files, hierarchical data format 5 (HDF5), European data format (EDF), BrainVision core data format (BVCDF), or BlackRock Microsystem (NEV/NSx); process images in Neuroimaging informatics technology initiative (NIfTI) and 'FreeSurfer' formats, providing brain imaging normalization to template brain, facilitating 'threeBrain' package for comprehensive electrode localization via 'YAEL' (your advanced electrode localizer) by Wang, Magnotti, Zhang, and Beauchamp (2023, <doi:10.1523/ENEURO.0328-23.2023>).

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Language en-US

URL https://rave.wiki, http://rave.wiki/ravecore/

BugReports https://github.com/rave-ieeg/ravecore/issues

Imports tools, utils, bidsr, data.table, filearray (>= 0.2.0), fs, ieegio, jsonlite, methods, R6, ravepipeline (>= 0.0.2), ravetools, S7, threeBrain

Suggests rpyANTs, rpymat, htmltools, httpuv, knitr, plotly, RNifti, RNiftyReg, shiny, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Zhengjia Wang [aut, cre] (ORCID:
<https://orcid.org/0000-0001-5629-1116>),
Xiang Zhang [aut],
John Magnotti [aut],
Michael Beauchamp [aut],
Trustees of the University of Pennsylvania [cph] (All files in this
package unless explicitly stated in the file)

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

Repository CRAN

Date/Publication 2026-04-02 14:30:02 UTC

Contents

`archive_subject` *Archive and share a subject*

Description

Archive and share a subject

Usage

```
archive_subject(
  subject,
  path,
  includes = c("orignal_signals", "processed_data", "rave_imaging", "pipelines", "notes",
    "user_generated"),
  config = list(),
  work_path = NULL,
  zip_flags = NULL
)
```

Arguments

subject	'RAVE' subject to archive
path	path to a zip file to store; if missing or empty, then the path will be automatically created
includes	data to include in the archive; default includes all (original raw signals, processed signals, imaging files, stored pipelines, notes, and user-generated exports)
config	a list of configurations, including changing subject code, project name, or to exclude cache data; see examples
work_path	temporary working path where files are copied; default is temporary path. Set this variable explicitly when temporary path is on external drives (for example, users have limited storage on local drives and cannot hold the entire subject)
zip_flags	zip flags

Examples

```
## Not run:


# Basic usage
path <- archive_subject('demo/DemoSubject')

# clean up
unlink(path)

# Advanced usage: include all the original signals
# and processed data, no cache data, re-name to
# demo/DemoSubjectLite
path <- archive_subject(
  'demo/DemoSubject',
  includes = c("orignal_signals", "processed_data"),
  config = list(
    rename = list(
      project_name = "demo",
      subject_code = "DemoSubjectLite"
    ),
    orignal_signals = list(
```

```
      # include all raw signals
      include_all = TRUE
    ),
    processed_data = list(
      include_cache = FALSE
    )
  )
)

# Clean up temporary zip file
unlink(path)


## End(Not run)
```

---

as_rave_project *Convert character to* RAVEProject *instance*

Description

Convert character to RAVEProject instance

Usage

```
as_rave_project(x, ...)

## S3 method for class 'character'
as_rave_project(x, strict = TRUE, parent_path = NULL, ...)
```

Arguments

x	R object that can be converted to 'RAVE' project. When x is a character, see 'Details' on the rules.
...	passed to other methods, typically includes strict on whether to check existence of the project folder, and parent_path, specifying non-default project root
strict	whether to check project path; if set to true and the project path is missing, the program will raise warnings
parent_path	parent path in which the project is non-default, can be a path to the parent folder of the project, or a bids_project object. When the subject is from 'BIDS', the parent_path must be the root of 'BIDS' directory.

Details

A 'RAVE' project is an aggregation of subjects with the similar research targets. For example, 'RAVE' comes with a demo subject set, and the project 'demo' contains eight subjects undergoing same experiments. Project `'YAEL'` contains subject whose electrodes are localized by `'YAEL'` modules. The project can be "arbitrary": this is different to a 'BIDS' "project", often served as a data-set name or identifier. A 'BIDS' project may have multiple 'RAVE' projects. For example, an audio-visual 'BIDS' data may have a 'RAVE' project `'McGurk'` to study the `'McGurk'` effect and another `'synchrony'` to study the audio-visual synchronization.

A valid 'RAVE' project name must only contain letters and digits; underscores and dashes may be acceptable but might subject to future change. For example `'demo'` is a valid project name, but `'my demo'` is invalid.

RAVE supports storing the data in `'native'` or `'bids'`-compliant formats. The native format is compatible with the 'RAVE' 1.0 and 2.0, and requires no conversion to 'BIDS' format, while `'bids'` requires the data to be stored and processed in 'BIDS'-complaint format, which is better for data sharing and migration, but might be over-kill in some cases.

If the project string contains `'@'`, the characters after the 'at' sign will be interpreted as indication of the storage format. For instance `'demo@native'` or `'demo@bids:ds0001'` are interpreted differently. The previous one indicates that the project `'demo'` is stored with native format, usually located at `'rave_data/data_dir'` under the home directory (can be manually set to other locations). The latter one means the 'RAVE' project `'demo'` is stored under 'BIDS' folder with a 'BIDS' data-set name `'ds0001'`.

Value

A `RAVEProject` instance

See Also

`RAVEProject`

Examples

```
# ---- Native format (RAVE legacy) ------------------------
project <- as_rave_project("demo", strict = FALSE)

format(project)

project$path

project$subjects()

# Non-standard project locations (native format)
as_rave_project("demo", strict = FALSE,
                parent_path = "~/Downloads")


# ---- BIDS format ----------------------------------------
project <- as_rave_project("demo@bids:ds001", strict = FALSE)
```

```
format(project)

project$path

# BIDS format, given the parent folder; this example requires
# 'bidsr' sample data. Run `bidsr::download_bids_examples()` first.

examples <- bidsr::download_bids_examples(test = TRUE)

if(!isFALSE(examples)) {

  project <- as_rave_project(
    "audiovisual@bids", strict = FALSE,
    parent_path = file.path(examples, "ieeg_epilepsy_ecog"))

  # RAVE processed data is under BIDS dirivative folder
  project$path

  # "audiovisual@bids:ieeg_epilepsy_ecog"
  format(project)
}
```

`as_yael_process` *Create a 'YAEL' imaging processing instance*

Description

Image registration across different modals. Normalize brain 'T1'-weighted 'MRI' to template brain and generate subject-level atlas files. See `cmd_run_yael_preprocess` to see how to run a built-in workflow

Usage

```
as_yael_process(subject)
```

Arguments

`subject`	character (subject code, or project name with subject code), or `RAVESubject` instance.

Value

A processing instance, see `YAELProcess`

Examples

```
process <- as_yael_process("YAEL/test_subject")

## Not run:

# Import and set original T1w MRI and CT
process$set_input_image("/path/to/T1w_MRI.nii", type = "T1w")
process$set_input_image("/path/to/CT.nii.gz", type = "CT")

# Co-register CT to MRI
process$register_to_T1w(image_type = "CT")

# Morph T1w MRI to 0.5 mm^3 MNI152 template
process$map_to_template(
  template_name = "mni_icbm152_nlin_asym_09b",
  native_type = "T1w"
)


## End(Not run)
```

Auxiliary_electrode *Class definition for auxiliary channels*

Description

Class definition for auxiliary channels

Class definition for auxiliary channels

Value

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.

Super classes

`ravepipeline::RAVESerializable` -> `ravecore::RAVEAbstarctElectrode` -> `Auxiliary_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

`preprocess_info` preprocess information

`voltage_file` path to voltage 'HDF5' file

Methods

Public methods:

- Auxiliary_electrode$@marshal()
- Auxiliary_electrode$@unmarshal()
- Auxiliary_electrode$print()
- Auxiliary_electrode$set_reference()
- Auxiliary_electrode$new()
- Auxiliary_electrode$.load_noref_voltage()
- Auxiliary_electrode$.load_raw_voltage()
- Auxiliary_electrode$load_data_with_epochs()
- Auxiliary_electrode$load_dimnames_with_epochs()
- Auxiliary_electrode$load_data_with_blocks()
- Auxiliary_electrode$load_dim_with_blocks()
- Auxiliary_electrode$clear_cache()
- Auxiliary_electrode$clear_memory()
- Auxiliary_electrode$clone()

Method `@marshal()`**:** Internal method

Usage:

```
Auxiliary_electrode$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
Auxiliary_electrode$@unmarshal(object)
```

Arguments:

`object, ...` internal arguments

Method `print()`**:** print electrode summary

Usage:

```
Auxiliary_electrode$print()
```

Method `set_reference()`**:** set reference for current electrode

Usage:

```
Auxiliary_electrode$set_reference(reference)
```

Arguments:

`reference` either `NULL` or `LFP_electrode` instance

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

Usage:

```
Auxiliary_electrode$new(subject, number, quiet = FALSE)
```

Arguments:

`subject`, `number`, `quiet` see constructor in `RAVEAbstarctElectrode`

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

Usage:

```
Auxiliary_electrode$.load_noref_voltage(reload = FALSE)
```

Arguments:

`reload` whether to reload cache

`srate` voltage signal sample rate

Method `.load_raw_voltage()`: load raw voltage (no process)

Usage:

```
Auxiliary_electrode$.load_raw_voltage(reload = FALSE)
```

Arguments:

`reload` whether to reload cache

Method `load_data_with_epochs()`: method to load electrode data

Usage:

```
Auxiliary_electrode$load_data_with_epochs(type = c("raw-voltage", "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_dimnames_with_epochs()`: get expected dimension names

Usage:

```
Auxiliary_electrode$load_dimnames_with_epochs(
  type = c("raw-voltage", "voltage")
)
```

Arguments:

`type` see `load_data_with_epochs`

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

Usage:

```
Auxiliary_electrode$load_data_with_blocks(
  blocks,
  type = c("raw-voltage", "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

Method `load_dim_with_blocks()`: get expected dimension information for block-based loader

Usage:

```
Auxiliary_electrode$load_dim_with_blocks(
  blocks,
  type = c("raw-voltage", "voltage")
)
```

Arguments:

`blocks, type` see `load_data_with_blocks`

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

Usage:

```
Auxiliary_electrode$clear_cache(...)
```

Arguments:

`...` ignored

Method `clear_memory()`: method to clear memory

Usage:

```
Auxiliary_electrode$clear_memory(...)
```

Arguments:

`...` ignored

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

Usage:

```
Auxiliary_electrode$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`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

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

path2 <- backup_file(path, remove = TRUE)

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

---

cache_path *Manipulate cached data on the file systems*

Description

Manipulate cached data on the file systems

Usage

```
cache_root(check = FALSE)

clear_cached_files(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

```
cache_root()
```

---

cmd-external *External shell commands for 'RAVE'*

Description

These shell commands are only tested on 'MacOS' and 'Linux'. On 'Windows' machines, please use the 'WSL2' system.

Usage

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

cmd_run_r(
  expr,
  quoted = FALSE,
  verbose = TRUE,
  dry_run = FALSE,
  log_file = tempfile(),
  script_path = tempfile(),
  ...
)
```

Arguments

script	the shell script
script_path	path to run the script
command	which command to invoke; default is 'bash'
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

backup	whether to back up the script file immediately; default is true
args	further arguments in the shell command, especially the 'FreeSurfer' reconstruction command
...	passed to system2, or additional arguments
expr	expression to run as command
quoted	whether expr is quoted; default is false
verbose	whether to print out the command script; default is true under dry-run mode, and false otherwise
log_file	where should log file be stored

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

---

cmd_run_3dAllineate *Align images using 'AFNI'*

Description

This is a legacy script and possibly contain errors. Please use cmd_run_ants_coreg for faster and stable implementation instead.

Usage

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

Arguments

subject	subject ID
ct_path, mri_path	absolute paths to 'CT' and 'MR' image files
overwrite	whether to overwrite existing files
command_path	path to 'AFNI' home
dry_run	dry-run flag
verbose	whether to print out script

---

cmd_run_ants_coreg *Register a computerized tomography (CT) image to MRI via 'ANTs'*

Description

Please avoid calling ants_coreg directly; use cmd_run_ants_coreg for more robust behaviors

Usage

```
ants_coreg(
  ct_path,
  mri_path,
  coreg_path = NULL,
 reg_type = c("DenseRigid", "Rigid", "SyN", "Affine", "TRSAA", "SyNCC", "SyNOnly"),
  aff_metric = c("mattes", "meansquares", "GC"),
  syn_metric = c("mattes", "meansquares", "demons", "CC"),
  verbose = TRUE,
  ...
)

cmd_run_ants_coreg(
  subject,
  ct_path,
  mri_path,
 reg_type = c("DenseRigid", "Rigid", "SyN", "Affine", "TRSAA", "SyNCC", "SyNOnly"),
  aff_metric = c("mattes", "meansquares", "GC"),
  syn_metric = c("mattes", "meansquares", "demons", "CC"),
  verbose = TRUE,
  dry_run = FALSE
)
```

Arguments

ct_path, mri_path	absolute paths to 'CT' and 'MR' image files
coreg_path	registration path, where to save results; default is the parent folder of ct_path

reg_type	registration type, choices are 'DenseRigid', 'Rigid', 'Affine', 'SyN', 'TRSAA', 'SyNCC', 'SyNOnly', or other types; see ants_registration
aff_metric	cost function to use for linear or 'affine' transform
syn_metric	cost function to use for 'SyN' transform
verbose	whether to verbose command; default is true
...	passed to ants_registration
subject	'RAVE' subject
dry_run	whether to dry-run the script and to print out the command instead of executing the code; default is false

Value

Aligned 'CT' will be generated at the coreg_path path:

'ct_in_t1.nii.gz' aligned 'CT' image; the image is also re-sampled into 'MRI' space

'transform.yaml' transform settings and outputs

'CT_IJK_to_MR_RAS.txt' transform matrix from volume 'IJK' space in the original 'CT' to the 'RAS' anatomical coordinate in 'MR' scanner; 'affine' transforms only

'CT_RAS_to_MR_RAS.txt' transform matrix from scanner 'RAS' space in the original 'CT' to 'RAS' in 'MR' scanner space; 'affine' transforms only

---

cmd_run_dcm2niix *Convert DICOM to NIfTI via 'dcm2niix'*

Description

Check https://rave.wiki on how to set up 'conda' environment for 'RAVE' using 'ravemanager'.

Usage

```
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
)
```

Arguments

`subject`	'RAVE' subject or a subject ID
`src_path`	source directory
`type`	image type
`merge`, `float`, `crop`	`'dcm2niix'` parameters
`overwrite`	overwrite existing files
`command_path`	path to program `'dcm2niix'`
`dry_run`	whether to dry-run
`verbose`	whether to print out command

Value

A command set running the terminal command; a folder named with `type` will be created under the subject image input folder

Examples

```
## Not run:

cmd_run_dcm2niix(
  "YAEL/pt02",
  "/path/to/DICOMDIR",
  "MRI"
)


## End(Not run)
```

`cmd_run_freesurfer_recon_all`

Workflow: 'FreeSurfer' surface reconstruction

Description

Runs 'FreeSurfer' `recon-all` command underneath; must have 'FreeSurfer' installed.

Usage

```
cmd_run_freesurfer_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
)

cmd_run_freesurfer_recon_all_clinical(
  subject,
  mri_path,
  work_path = NULL,
  overwrite = FALSE,
  command_path = NULL,
  dry_run = FALSE,
  verbose = dry_run,
  ...
)
```

Arguments

subject	'RAVE' subject or subject ID
mri_path	path to 'T1'-weighted 'MRI', must be a 'NIfTI' file
args	type of workflow; see 'FreeSurfer' `recon-all` command documentation; default choice is `'-all'` to run all workflows
work_path	working directory; 'FreeSurfer' errors out when working directory contains white spaces. By default, 'RAVE' automatically creates a symbolic link to a path that contains no white space. Do not set this input manually unless you know what you are doing
overwrite	whether to overwrite existing work by deleting the folder; default is false. In case of errors, set this to true to restart the workflow; make sure you back up the files first.
command_path	'FreeSurfer' home directory. In some cases, 'RAVE' might not be able to find environment variable `'FREESURFER_HOME'`. Please manually set the path if the workflow fails. Alternatively, you can manually set FreeSurfer' home directory via 'RAVE' options `raveio_setopt("freesurfer_path", "/path/to/freesurfer/home")` prior to running the script
dry_run	avoid running the code, but print the process instead
verbose	print messages
...	ignored

Value

A list of shell command set.

Examples

```
# Requires `FreeSurfer` and only works on MacOS or Linux
# as `FreeSurfer` does not support Windows
```

```
## Not run:

# Create subject instance; strict=FALSE means it's OK if the subject
# is missing
subject <- as_rave_subject("YAEL/s01", strict = FALSE)

cmd_run_freesurfer_recon_all(
  subject = subject,
  mri_path = "/path/to/T1.nii.gz"
)


## End(Not run)
```

`cmd_run_fsl_flirt` *Run 'FSL' linear registration*

Description

Run 'FSL' linear registration

Usage

```
cmd_run_fsl_flirt(
  subject,
  mri_path,
  ct_path,
  dof = 6,
 cost = c("mutualinfo", "leastsq", "normcorr", "corratio", "normmi", "labeldiff", "bbr"),
  search = 90,
 searchcost = c("mutualinfo", "leastsq", "normcorr", "corratio", "normmi", "labeldiff",
    "bbr"),
  overwrite = FALSE,
  command_path = NULL,
  dry_run = FALSE,
  verbose = dry_run
)
```

Arguments

`subject`	'RAVE' subject or subject ID
`mri_path`	path to 'MRI' (fixed image)
`ct_path`	path to 'CT' (moving image)
`dof`	degrees of freedom; default is 6 (rigid-body); set to 12 ('affine')
`cost`, `searchcost`	cost function name

search	search degrees; default is 90 to save time, set to 180 for full search
overwrite	overwrite existing files
command_path	path to 'FSLDIR' environment variable
dry_run	whether to dry-run
verbose	whether to print out command

Value

A command set running the terminal command; a 'coregistration' folder will be created under the subject imaging directory

---

`cmd_run_yael_preprocess` *Run a built-in 'YAEL' imaging processing workflow*

Description

Image processing pipeline doi:10.1523/ENEURO.032823.2023, allowing cross-modality image registration, T1-weighted MRI normalization to template brain, creating subject-level brain atlas from inverse normalization.

Usage

```
yael_preprocess(
  subject,
  t1w_path = NULL,
  ct_path = NULL,
  t2w_path = NULL,
  fgatir_path = NULL,
  preopct_path = NULL,
  flair_path = NULL,
  t1w_contrast_path = NULL,
  register_policy = c("auto", "all"),
  register_reversed = FALSE,
  normalize_template = "mni_icbm152_nlin_asym_09b",
  normalize_policy = c("auto", "all"),
  normalize_images = c("T1w", "T2w", "T1wContrast", "fGATIR", "preopCT"),
 normalize_back = ifelse(length(normalize_template) >= 1, normalize_template[[1]], NA),
  atlases = list(),
  add_surfaces = FALSE,
  use_antspynet = TRUE,
  verbose = TRUE,
  ...
)

cmd_run_yael_preprocess(
```

```
  subject,
  t1w_path = NULL,
  ct_path = NULL,
  t2w_path = NULL,
  fgatir_path = NULL,
  preopct_path = NULL,
  flair_path = NULL,
  t1w_contrast_path = NULL,
  register_reversed = FALSE,
  normalize_template = "mni_icbm152_nlin_asym_09b",
  normalize_images = c("T1w", "T2w", "T1wContrast", "fGATIR", "preopCT"),
  run_recon_all = TRUE,
  dry_run = FALSE,
  use_antspynet = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`subject`	subject ID
`t1w_path`	path to 'T1'-weighted preoperative 'MRI', used as underlay and base image. If you want to have 'ACPC' aligned scanner coordinate system. Please align the image before feeding into this function. All images must contain skulls (do not strip skulls)
`ct_path`, `t2w_path`, `fgatir_path`, `preopct_path`, `flair_path`, `t1w_contrast_path`	additional optional images to be aligned to the underlay; the registration will be symmetric and the rigid-body transforms will be stored.
`register_policy`	whether to skip already registered images; default is true (`'auto'`); set to `'all'` to ignore existing registrations and force calculation
`register_reversed`	whether to swap the moving images and the fixing image; default is false
`normalize_template`	template to normalize to: default is `'mni_icbm152_nlin_asym_09b'` ('MNI152b', 0.5 mm resolution); when the computer memory is below 12 gigabytes, the template will automatically switch to `'mni_icbm152_nlin_asym_09a'` (known as 'MNI152a', 1 mm voxel resolution). Other choices are `'mni_icbm152_nlin_asym_09c'` and `'fsaverage'` (or known as 'MNI305')
`normalize_policy`	whether to skip existing normalization, if calculated; default is `'auto'` (yes); set to `'all'` to ignore
`normalize_images`	images used for normalization; default is to include common images before the implantation (if available)
`normalize_back`	length of one (select from `normalize_template`), which template is to be used to generate native brain mask and transform matrices

atlases	a named list: the names must be template names from normalize_template and the values must be directories of atlases of the corresponding templates (see 'Examples').
add_surfaces	whether to add surfaces for the subject; default is FALSE. The surfaces are created by reversing the normalization from template brain, hence the results will not be accurate. Enable this option only if cortical surface estimation is not critical (and 'FreeSurfer' reconstructions are inaccessible)
use_antspynet	whether to try 'antspynet' if available; default is true, which uses deep_atropos instead of the conventional atropos to speed up and possibly with more accurate results.
verbose	whether to print out the information; default is TRUE
...	reserved for legacy code and deprecated arguments
run_recon_all	whether to run 'FreeSurfer'; default is true
dry_run	whether to dry-run

Value

Nothing, a subject imaging folder will be created under 'RAVE' raw folder. It will take a while to run the workflow.

Examples

```
## Not run:

# For T1 normalization only; add ct_path to include coregistration
cmd_run_yael_preprocess(
  subject = "pt01",
  t1w_path = "/path/to/T1w.nii.gz",

  # normalize T1 to MNI152
  normalize_template = 'mni_icbm152_nlin_asym_09b'
)



## End(Not run)
```

---

collapse2 *Collapse high-dimensional tensor array*

Description

Collapse high-dimensional tensor array

Usage

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

## S3 method for class 'FileArray'
collapse2(x, keep, method = c("mean", "sum"), ...)

## S3 method for class 'RAVEFileArray'
collapse2(x, keep, method = c("mean", "sum"), ...)

## S3 method for class 'Tensor'
collapse2(x, keep, method = c("mean", "sum"), ...)

## S3 method for class 'array'
collapse2(x, keep, method = c("mean", "sum"), ...)
```

Arguments

x	R array, FileArray-class, or other objects
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

```
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)

# filearray
y <- filearray::as_filearray(x)

collapse2(y, c(3, 2))

collapse2(y, c(3, 2), "sum")

# clean up
y$delete(force = TRUE)
```

`collapse_power` *Collapse power array with given analysis cubes*

Description

Collapse power array with given analysis cubes

Usage

```
collapse_power(x, analysis_index_cubes)

## S3 method for class 'array'
collapse_power(x, analysis_index_cubes)

## S3 method for class 'FileArray'
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

```
# 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))

x$delete(force = TRUE)
```

`compose_channel` *Compose a phantom channel from existing electrodes*

Description

In some cases, for example, deep-brain stimulation ('DBS'), it is often needed to analyze averaged electrode channels from segmented 'DBS' leads, or create bipolar contrast between electrode channels, or to generate non-equally weighted channel averages for 'Laplacian' reference. `compose_channel` allows users to generate a phantom channel that does not physically exist, but is treated as a normal electrode channel in 'RAVE'.

Usage

```
compose_channel(
  subject,
  number,
  from,
  weights = rep(1/length(from), length(from)),
  normalize = FALSE,
  force = FALSE,
  label = sprintf("Composed-%s", number),
  signal_type = c("auto", "LFP", "Spike", "EKG", "Auxiliary", "Unknown")
)
```

Arguments

`subject`	'RAVE' subject
`number`	new channel number, must be positive integer, cannot be existing electrode channel numbers
`from`	a vector of electrode channels that is used to compose this new channel, must be non-empty; see `weights` if these channels are not equally weighted.
`weights`	numerical weights used on each `from` channels; the length of `weights` must equals to the length of `from`; default is equally weighted for each channel (mean of `from` channels).
`normalize`	whether to normalize the weights such that the composed channel has the same variance as `from` channels; default is false
`force`	whether to overwrite existing composed channel if it exists; default is false. By specifying `force=TRUE`, users are risking breaking the data integrity since any analysis based on the composed channel is no longer reproducible. Also users cannot overwrite original channels under any circumstances.
`label`	the label for the composed channel; will be stored at `'electrodes.csv'`
`signal_type`	signal type of the composed channel; default is `'auto'` (same as the first `from` channel); other choices see SIGNAL_TYPES

Value

Nothing

Examples

```
if(interactive() && has_rave_subject("demo/DemoSubject")) {

  # the actual example code:
  # new channel 100 = 2 x channel 14 - (channe 15 + 16)
  compose_channel(
    subject = "demo/DemoSubject",
    number = 100,
    from = c(14, 15, 16),
    weights = c(2, -1, -1),
```

```
    normalize = FALSE
  )

}
```

`convert_electrode_table_to_bids` *Convert electrode table*

Description

Convert electrode table

Usage

```
convert_electrode_table_to_bids(
  subject,
  space = c("ScanRAS", "MNI305", "fsnative")
)
```

Arguments

`subject`	'RAVE' subject
`space`	suggested coordinate space, notice this argument might not be supported when `'FreeSurfer'` reconstruction is missing.

Value

A list of table in data frame and a list of meta information

Examples

```
# Run `install_subject("DemoSubject")` first!
if( has_rave_subject("demo/DemoSubject") ) {

convert_electrode_table_to_bids(
  "demo/DemoSubject",
  space = "ScanRAS"
)

}
```

`export_table` *Export data frame to different common formats*

Description

Stores and load data in various of data format. See 'Details' for limitations.

Usage

```
export_table(
  x,
  file,
  format = c("auto", "csv", "csv.zip", "tsv", "h5", "fst", "json", "rds", "yaml"),
  ...
)

import_table(
  file,
  format = c("auto", "csv", "csv.zip", "tsv", "h5", "fst", "json", "rds", "yaml"),
  ...
)
```

Arguments

`x`	data table to be saved to `file`
`file`	file to store the data
`format`	data storage format, default is `'auto'` (infer from the file extension); other choices are `'csv'`, `'csv.zip'`, `'h5'`, `'fst'`, `'json'`, `'rds'`, `'yaml'`
`...`	parameters passed to other functions

Details

The format `'rds'`, `'h5'`, `'fst'`, `'json'`, and `'yaml'` try to preserve the first-level column attributes. Factors will be preserved in these formats. Such property does not exist in `'csv'`, `'csv.zip'` formats.

Open-data formats are `'h5'`, `'csv'`, `'csv.zip'`, `'json'`, `'yaml'`. These formats require the table elements to be native types (numeric, character, factor, etc.).

`'rds'`, `'h5'`, and `'fst'` can store large data sets. `'fst'` is the best choice is performance and file size are the major concerns. `'rds'` preserves all the properties of the table.

Value

The normalized path for `export_table`, and a `data.table` for `import_table`

Examples

```
x <- data.table::data.table(
  a = rnorm(10),
  b = letters[1:10],
  c = 1:10,
  d = factor(LETTERS[1:10])
)

f <- tempfile(fileext = ".csv.zip")

export_table(x = x, file = f)

y <- import_table(file = f)

str(x)
str(y)

# clean up
unlink(f)
```

`generate_atlases_from_template` *Create brain atlases from template*

Description

Reverse-transform from template atlases via volume mapping

Usage

```
generate_atlases_from_template(
  subject,
  atlas_folders,
  template_name = "mni_icbm152_nlin_asym_09b",
  any_mni152 = TRUE,
  surfaces = TRUE,
  as_job = FALSE,
  extra_transform = NULL,
  extra_transform_type = c("ants", "native")
)
```

Arguments

`subject`	'RAVE' subject instance or character; see `as_rave_subject`
`atlas_folders`	paths to the atlas folders

`template_name` template name where the atlases are created; see `get_available_morph_to_template` to get available templates and `cmd_run_yael_preprocess` to generate transforms to the templates

`any_mni152` if the template is 'MNI152', then whether to check other template folders; default is true

`surfaces` whether to generate surfaces; if true, then a 'GIfTI' file and a 'STL' file will be created for each atlas, if applicable. The 'GIfTI' file will be in right-anterior-superior 'RAS' coordinate system to comply with the 'NIfTI' standard, and 'STL' will be in left-posterior-superior 'LPS' system that can be imported to 'ITK'-based software such as 'BrainLab' or 'ANTs'.

`as_job` whether to run as a background job for cleaner environment; default is false.

`extra_transform`
additional 'affine' transform (a four-by-four matrix) that maps the subject native image (not points) to the target image. This argument is for pipelines that take in transformed/processed images as input; default is `NULL`, see 'Details'.

`extra_transform_type`
type of transform if `extra_transform` is provided.

Details

The workflow generates atlases from a template. To run this function, a separate normalization is required (see `cmd_run_yael_preprocess`). That function normalizes native subject's brain (`'T1w'` image) to a specified template (typically 'MNI152b' non-linear asymmetric version) via a deformation field. Once obtaining the normalization transform, `generate_atlases_from_template` inverse the process, creating subject-level atlases based on that template.

When `surfaces` is `TRUE`, each atlas or mask file will be binned with a threshold of `0.5`. The resulting binary mask will be used to generate a mask surface using `volume_to_surf` function via implicit Laplacian smoothing.

The resulting surfaces and atlases typically sit at native space aligned with input `'T1w'` image. For each volume, a 'GIfTI' surface will be created under right-anterior-superior ('RAS') coordinate system for further inferences. In addition, an 'STL' surface will be created under left-posterior-superior ('LPS') coordinate system. This file is mainly to be used as object file for visualizations in software such as 'BrainLab'.

Some users might have extra processing before 'YAEL' pipeline (such as 'ACPC' realignment). This function provides a set of extra arguments (`extra_transform` and `extra_transform_type`) allowing generating atlases to align with the raw 'DICOM' images. `extra_transform` needs to be an `'affine'` matrix that maps the image from the input of the 'YAEL' pipeline to the raw 'DICOM' image. Be aware that this transform is not the point transformation matrix. When `extra_transform` is non-empty, an extra folder `atlases_extra` will be created under the subject imaging folder, with transformed files aligned to 'DICOM' stored.

Value

The function returns nothing, but will create a folder named `'atlases'` under raw subject `'rave-imaging'` folder.

Examples

```
# Please check out https://rave.wiki to configure Python for RAVE
# or run ravemanager::configure_python()
## Not run:

generate_atlases_from_template(
  "YAEL/OCD07", "/path/to/OCD_atlases")

# If the image was ACPC realigned before being fed into the YAEL
# pipeline
generate_atlases_from_template(
  "YAEL/OCD07", "/path/to/OCD_atlases",
  extra_transform =
    "/path/to/sub-OCD7_from-T1wACPC_to-T1wNative_mode-image_xfm.mat"
)


## End(Not run)
```

---

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_available_morph_to_template *Get names of available non-linear transforms to the templates*

Description

This function obtains existing transforms rather than creating new transforms. Please check function cmd_run_yael_preprocess about mapping native images to the templates. This function requires additional 'Python' configuration.

Usage

```
get_available_morph_to_template(subject)
```

Arguments

subject	'RAVE' subject instance or character; see as_rave_subject

Value

Template names with valid non-linear transforms.

Examples

```
# Please check out https://rave.wiki to configure Python for RAVE
# or run ravemanager::configure_python()
## Not run:

get_available_morph_to_template("project_name/subject_code")


## End(Not run)
```

get_projects *Get all possible projects in 'RAVE' default directory*

Description

Get all possible projects in 'RAVE' default directory

Usage

```
get_projects(refresh = TRUE)
```

Arguments

refresh	whether to refresh the cache; default is true

Value

characters of project names

Examples

```
get_projects()
```

---

glimpse-repository *Visualizes repositories with interactive plots*

Description

Requires optional package 'plotly'; please install the package prior to launching the viewer.

Usage

```
glimpse_voltage_repository_with_blocks(
  repository,
  initial_block = NULL,
  channels = NULL,
  epoch = NULL,
  start_time = 0,
  duration = 5,
  channel_gap = 1000,
  highpass_freq = NA,
  lowpass_freq = NA
)

glimpse_voltage_filearray(
  filearray,
  sample_rate,
  channels = NULL,
  epoch = NULL,
  start_time = 0,
  duration = 5,
  channel_gap = 1000,
  highpass_freq = NA,
  lowpass_freq = NA
)
```

Arguments

`repository`	'RAVE' repository
`initial_block`	initial recording block to select
`channels`	channels to visualize; default is all
`epoch`	additional epoch to annotation
`start_time`, `duration`, `channel_gap`	initial start time, duration, and channel gap (can be changed later)
`highpass_freq`, `lowpass_freq`	filter to apply when visualizing the signals, useful when signals have 'DC' shift
`filearray`	a `as_filearray` object, must be two dimensional matrix for voltage (time by electrode), with `dimnames` being the time in seconds and electrode in label name
`sample_rate`	sample rate of the file-array

Value

An R-shiny application container environment; use `print` method to launch the application.

Examples

```
if(has_rave_subject("demo/DemoSubject")) {
  subject <- as_rave_subject("demo/DemoSubject", strict = FALSE)


  repository <- ravecore::prepare_subject_voltage_with_blocks(
    subject = subject)

  if (interactive()) {
    app <- glimpse_voltage_repository_with_blocks(
      repository = repository,
      initial_block = "008",
      epoch = "auditory_onset",
      highpass_freq = 0.5
    )

    print(app)
    close(app)

  }

}

# ---- Example 2 -------------------------------------------------

# Construct a filearray


sample_rate <- 100
filearray <- filearray::as_filearray(array(rnorm(50000),
                                           dim = c(10000, 5)))
```

```
dimnames(filearray) <- list(
  Time = seq_len(10000) / sample_rate,
  Electrode = 1:5
)

if(interactive()) {

  app <- glimpse_voltage_filearray(filearray = filearray,
                                   sample_rate = sample_rate,
                                   channel_gap = 6)

  print(app)
}
```

`import-signals` *Import signal data into 'RAVE'*

Description

Import signal data from different file formats; supports `'EDF'`, `'BrainVision'`, `'BlackRock'`, `'HDF5'`, and `'Matlab'` formats under either native or 'BIDS' standard. It is recommended to use 'RAVE' user interfaces to import data.

Usage

```
import_from_brainvis(
  subject,
  blocks,
  electrodes,
  sample_rate,
  add = FALSE,
  data_type = "LFP",
  ...
)

import_from_edf(
  subject,
  blocks,
  electrodes,
  sample_rate,
  add = FALSE,
  data_type = "LFP",
  skip_validation = FALSE,
  ...
)
```

```
import_from_h5_mat_per_block(
  subject,
  blocks,
  electrodes,
  sample_rate,
  add = FALSE,
  data_type = "LFP",
  skip_validation = FALSE,
  ...
)

import_from_h5_mat_per_channel(
  subject,
  blocks,
  electrodes,
  sample_rate,
  add = FALSE,
  data_type = "LFP",
  skip_validation = FALSE,
  ...
)

import_from_nevnsx(
  subject,
  blocks,
  electrodes,
  sample_rate,
  add = FALSE,
  data_type = "LFP",
  skip_validation = FALSE,
  ...
)
```

Arguments

`subject`	a 'RAVE' subject or subject ID, consists of a project name, a forward slash, followed by a subject code; for example, `'demo/DemoSubject'` refers to a 'RAVE' native subject `'DemoSubject'` under the project `'demo'`; while `'demo@bids:ds001/01'` refers to subject `'sub-01'` from the 'BIDS' data set `'ds001'`, under its 'RAVE' project `'demo'` under the derivative folder.
`blocks`	recording block; see Section 'Recording Blocks' for details
`electrodes`	electrode (channels) to import, must be a vector of integers (channel numbers) or a character that can be interpreted as integers; for example, integer vector `1:10` stands for the first 10 channels, while `'1,3-10,15'` refers to channels 1, 3 to 10, then 15. Notice some formats might not have definition of the channel numbers, see Section 'Channel Numbers' for details
`sample_rate`	sampling frequency of the channel, must be positive. 'RAVE' only accepts uni-

	fied consistent sample rate across all channels with the same type. For example, if one 'LFP' channel is 2000 Hz, then all 'LFP' channels must be 2000 Hz. Channels with different sample rates will be either `decimate` (if possible, with a 'FIR' filter) or `resample` (fractional ratio) during import. Different channel types (such as 'Spike', 'Auxiliary', ...) can have different sample rates
`add`	whether the operation is to add new channels; default is false to protect data integrity
`data_type`	channel signal data type, can be `'LFP'`, `'Spike'`, or `'Auxiliary'`
`...`	passed to or reserved for other methods
`skip_validation`	whether to skip data validation, default is false (recommended)

Recording Blocks

The term "recording block" is defined as a continuous block of signals recorded during the experiment, typically during one run of the experiment, depending on the setups and format standards:

In the context of native standard, the raw 'RAVE' data is typically stored in the `'~/rave_data/raw_dir'` directory (`'~'` stands for your home directory, or documents directory under Windows). Each subject is stored under a folder named after the subject code. For example, subject `'DemoSubject'` has a raw folder path `'~/rave_data/raw_dir/DemoSubject'`. The block folders are stored under this subject folder (such as `'008'`, `'010'`, ...). Each block folder contains a 5-min recording from an experiment.

In the context of 'BIDS' standard, there is no official definition of a 'block', instead, 'BIDS' has an explicit definition of sessions, tasks, and runs. We typically consider that a combination of a session, a task, and a run consists of a recording block. For example, `'ses-01_task-01_run-01'` or `'ses-01_run-01'`, depending on the existence of the 'BIDS' entities.

Channel Numbers

(Electrode) channel numbers refer to vectors of integers, or characters that can be interpreted as integers. For integers, this is straightforward: `c(1:10, 21:30)` refers to channel 1 to 20, then 21 to 30. For characters, this is converted to integer internally via an unexported function `ravecore:::parse_svec`.

The channel numbers must be an integer. In some data formats (such as 'EDF') or some standards ('BIDS'), the channel number is not officially explicitly defined: they use the channel labels as the identifiers. To deal with this situation, 'RAVE' treats the channel order as their numbers. In some cases, this is less ideal because the channel labels might implicitly encode the channel numbers. 'RAVE' will ignore such information for consistent behavior.

`install_openneuro` *Install data-sets from OpenNeuro*

Description

Enjoy hundreds of open-access data sets from `https://openneuro.org` with a simple accession number.

Usage

```
install_openneuro(
  accession_number,
  subject_codes = NULL,
  tag = NULL,
  parent_folder = NULL
)
```

Arguments

accession_number	'OpenNeuro' accession number
subject_codes	subject codes, with or without the prefix 'sub-', default is NULL to download the entire data repository
tag	version number; default is NULL to download the latest version
parent_folder	parent directory where the data will be downloaded into the data folder name is always the accession number

Value

The data folder name on the local disk.

Examples

```
## Not run:

# Download Hermes D, Miller KJ, Wandell BA, Winawer J (2015) dataset
# from https://openneuro.org/datasets/ds005953

install_openneuro('ds005953')

# Download subject sub-HUP070 used by Bernabei & Li et al.
# from https://openneuro.org/datasets/ds004100

install_openneuro('ds004100', subject_codes = "HUP070")

# access the downloaded data
bids_parent_root <- ravepipeline::raveio_getopt("bids_data_dir")

# ---- Example of visualizing electrodes on the fsaverage ----
# Load BIDS project
proj_ds004100 <- bidsr::bids_project(
  file.path(bids_parent_root, "ds004100"))

# BIDS-R Subject instance
sub_HUP070 <- bidsr::bids_subject(proj_ds004100, "HUP070")

# Find BIDS entities with electrodes as suffix
electrode <- bidsr::query_bids(sub_HUP070, list(
  data_types = "ieeg",
```

```
  suffixes = "electrodes",
  sidecars = TRUE
))

# resolve electrode table path
electrode_path <- bidsr::resolve_bids_path(
  x = proj_ds004100,
  format(electrode$parsed[[1]]))

# laod electrode coordinate
tabular <- bidsr::as_bids_tabular(electrode_path)

# Build RAVE electrode table
electrode_coordinates <- data.frame(
  Electrode = 1:nrow(tabular$content),
  x = tabular$content$x,
  y = tabular$content$y,
  z = tabular$content$z,
  Label = tabular$content$name,
  Radius = 2,
  BIDSSubject = "sub-HUP070"
)

# Load RAVE brain - fsaverage
template <- threeBrain::merge_brain(template_subject = "fsaverage")
fsaverage <- template$template_object

# This dataset uses surface RAS; see coordsys JSON
# tkrRAS: surface RAS
# scannerRAS: MRI RAS
fsaverage$set_electrodes(electrode_coordinates, coord_sys = "tkrRAS")

fsaverage$plot()



## End(Not run)
```

`install_subject` *Install a subject from the internet, a zip file or a directory*

Description

Install a subject from the internet, a zip file or a directory

Usage

```
install_subject(
```

```
  path = ".",
  ask = interactive(),
  overwrite = FALSE,
  backup = TRUE,
  use_cache = TRUE,
  dry_run = FALSE,
  force_project = NA,
  force_subject = NA,
  ...
)
```

Arguments

path	path to subject archive, can be a path to directory, a zip file, or an internet address (must starts with 'http', or 'ftp')
ask	when overwrite is false, whether to ask the user if subject exists; default is true when running in interactive session; users will be prompt with choices; if ask=FALSE and overwrite=FALSE, then the process will end with a warning if the subject exists.
overwrite	whether to overwrite existing subject, see argument ask and backup
backup	whether to back-up the subject when overwriting the data; default is true, which will rename the old subject folders instead of removing; set to true to remove existing subject.
use_cache	whether to use cached extraction directory; default is true. Set it to FALSE if you want a clean installation.
dry_run	whether to dry-run the process instead of actually installing; this rehearsal can help you see the progress and prevent you from losing data
force_project, force_subject	force set the project or subject; will raise a warning as this might mess up some pipelines
...	passed to download.file

Examples

```
## Not run:

install_subject("DemoSubject")


## End(Not run)
```

LFP_electrode *Definitions of electrode with local field potential 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

Value

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

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.

Super classes

ravepipeline::RAVESerializable -> ravecore::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$@marshal()
- LFP_electrode$@unmarshal()
- 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$load_data_with_epochs()
- LFP_electrode$load_dimnames_with_epochs()
- LFP_electrode$load_data_with_blocks()
- LFP_electrode$load_dim_with_blocks()
- LFP_electrode$clear_cache()
- LFP_electrode$clear_memory()
- LFP_electrode$clone()

Method `@marshal()`**:** Internal method

Usage:

```
LFP_electrode$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
LFP_electrode$@unmarshal(object)
```

Arguments:

`object, ...` internal arguments

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 RAVEAbstarctElectrode

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

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_with_epochs()`: method to load electrode data

Usage:

```
LFP_electrode$load_data_with_epochs(
  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_dimnames_with_epochs()`: get expected dimension names

Usage:

```
LFP_electrode$load_dimnames_with_epochs(
  type = c("power", "phase", "voltage", "wavelet-coefficient", "raw-voltage")
)
```

Arguments:

`type` see `load_data_with_epochs`

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

Usage:

```
LFP_electrode$load_data_with_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

Method `load_dim_with_blocks()`: get expected dimension information for block-based loader

Usage:

```
LFP_electrode$load_dim_with_blocks(
  blocks,
  type = c("power", "phase", "voltage", "wavelet-coefficient", "raw-voltage")
)
```

Arguments:

`blocks, type` see `load_data_with_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

if(has_rave_subject("demo/DemoSubject")) {

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

# 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_with_epochs("power")
names(dimnames(power))

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


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

}
```

`LFP_reference` *Definitions of reference with local field potential 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`

Value

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

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.

Super classes

ravepipeline::RAVESerializable -> ravecore::RAVEAbstarctElectrode -> 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$@marshal()
- LFP_reference$@unmarshal()
- 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_with_epochs()
- LFP_reference$load_data_with_blocks()
- LFP_reference$load_dim_with_blocks()
- LFP_reference$clear_cache()
- LFP_reference$clear_memory()
- LFP_reference$clone()

Method @marshal(): Internal method

Usage:

```
LFP_reference$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
LFP_reference$@unmarshal(object)
```

Arguments:

`object, ...` internal arguments

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` must be `NULL`

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

Usage:

```
LFP_reference$new(subject, number, quiet = FALSE)
```

Arguments:

`subject, number, quiet` see constructor in `RAVEAbstarctElectrode`

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

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_with_epochs()`: method to load electrode data

Usage:

```
LFP_reference$load_data_with_epochs(
  type = c("power", "phase", "voltage", "wavelet-coefficient", "raw-voltage")
)
```

Arguments:

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

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

Usage:

```
LFP_reference$load_data_with_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

Method `load_dim_with_blocks()`: get expected dimension information for block-based loader

Usage:

```
LFP_reference$load_dim_with_blocks(
  blocks,
  type = c("power", "phase", "voltage", "wavelet-coefficient", "raw-voltage")
)
```

Arguments:

`blocks, type` see `load_data_with_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:

`deep` Whether to make a deep clone.

Examples

```
# Download subject demo/DemoSubject
if( has_rave_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_with_epochs("power")
```

```
names(dimnames(power))

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

# clear cache on hard disk
e$clear_cache()

}
```

---

meta-data *Load or save meta data to 'RAVE' subject*

Description

Load or save meta data to 'RAVE' subject

Usage

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

load_meta2(
 meta_type = c("electrodes", "frequencies", "time_points", "epoch", "references",
    "time_excluded", "info"),
  project_name,
  subject_code,
  subject_id,
  meta_name
)
```

Arguments

data	data table
meta_type	see load meta
project_name	project name
subject_code	subject code
subject_id	subject identified, alternative way to specify the project and subject in one string
meta_name	for epoch and reference only, the name the of the table

Value

The corresponding metadata

Examples

```
if(has_rave_subject("demo/DemoSubject")) {
  subject <- as_rave_subject("demo/DemoSubject", strict = FALSE)

  electrode_table <- subject$get_electrode_table()

  save_meta2(
    data = electrode_table,
    meta_type = "electrodes",
    project_name = subject$project_name,
    subject_code = subject$subject_code
  )

  load_meta2(meta_type = "electrodes", subject_id = subject)
}
```

---

new_electrode *Create new electrode channel instance or a reference signal instance*

Description

Create new electrode channel instance or a reference signal instance

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(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'
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. 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.

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 `RAVEAbstarctElectrode` class

Electrode or reference instances that inherit `RAVEAbstarctElectrode` class

Examples

```
# Download subject demo/DemoSubject
if( has_rave_subject("demo/DemoSubject") ) {


# 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_with_epochs("power")
names(dimnames(power))

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

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

}

# Download subject demo/DemoSubject
if( has_rave_subject("demo/DemoSubject") ) {


# 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_with_epochs("power")
names(dimnames(power))

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

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

}
```

new_rave_subject *Get* RAVESubject *instance from character*

Description

Get RAVESubject instance from character

Usage

```
new_rave_subject(project_name, subject_code, strict = TRUE)

as_rave_subject(subject_id, strict = TRUE, reload = TRUE)

has_rave_subject(subject_id)
```

Arguments

project_name	character of 'RAVE' project name
subject_code	character of 'RAVE' subject code
strict	whether to check if subject directories exist or not
subject_id	character in format "project/subject"
reload	whether to reload (update) subject information, default is true

Value

RAVESubject instance

See Also

RAVESubject

Examples

```
subject <- new_rave_subject(project_name = "demo@bids:ds04001",
                            subject_code = "DemoSubject",
                            strict = FALSE)

subject

subject$project$path
subject$imaging_path
```

`niftyreg_coreg` *Register a computerized tomography (CT) image to MRI via* `'NiftyReg'`

Description

Supports rigid, affine, or non-linear transformation

Usage

```
niftyreg_coreg(
  ct_path,
  mri_path,
  coreg_path = NULL,
  reg_type = c("rigid", "affine", "nonlinear"),
  interp = c("trilinear", "cubic", "nearest"),
  verbose = TRUE,
  ...
)

cmd_run_niftyreg_coreg(
  subject,
  ct_path,
  mri_path,
  reg_type = c("rigid", "affine", "nonlinear"),
  interp = c("trilinear", "cubic", "nearest"),
  verbose = TRUE,
  dry_run = FALSE,
  ...
)
```

Arguments

`ct_path`, `mri_path`	absolute paths to 'CT' and 'MR' image files
`coreg_path`	registration path, where to save results; default is the parent folder of `ct_path`
`reg_type`	registration type, choices are `'rigid'`, `'affine'`, or `'nonlinear'`
`interp`	how to interpolate when sampling volumes, choices are `'trilinear'`, `'cubic'`, or `'nearest'`
`verbose`	whether to verbose command; default is true
`...`	other arguments passed to `register_volume`
`subject`	'RAVE' subject
`dry_run`	whether to dry-run the script and to print out the command instead of executing the code; default is false

Value

Nothing is returned from the function. However, several files will be generated at the 'CT' path:

`'ct_in_t1.nii'` aligned 'CT' image; the image is also re-sampled into 'MRI' space

`'CT_IJK_to_MR_RAS.txt'` transform matrix from volume 'IJK' space in the original 'CT' to the 'RAS' anatomical coordinate in 'MR' scanner

`'CT_RAS_to_MR_RAS.txt'` transform matrix from scanner 'RAS' space in the original 'CT' to 'RAS' in 'MR' scanner space

---

`plot_volume_slices` *Plot volume slices into scalable vector graphics SVG images*

Description

Display slices, or interleave with image overlays. Require installing package `htmltools`.

Usage

```
plot_volume_slices(
  x,
  overlay = NULL,
  depths = seq(-100, 100, by = 18),
  which = c("coronal", "axial", "sagittal"),
  nc = NA,
  col = c("black", "white"),
  overlay_col = col,
  overlay_alpha = NA,
  interleave = is.na(overlay_alpha),
  interleave_period = 4,
  interleave_transition = c("ease-in-out", "linear"),
  pixel_width = 0.5,
  underlay_range = NULL,
  overlay_range = NULL,
  ...
)
```

Arguments

`x`	underlay, objects that can be converted to `as_ieegio_volume`; for example, 'NIfTI' or 'FreeSurfer' volume file path, array, loaded volume instances
`overlay`	same type as `x`, but optional; served as overlay
`depths`	depth position in millimeters, along the normal to the `which` plane
`which`	which plane to visualize; can be `"coronal"`, `"axial"`, `"sagittal"`
`nc`	number of columns; default is to be determined by total number of images

col, overlay_col
: underlay and overlay color keys, must have at least two colors to construct color palettes

overlay_alpha
: overlay transparency

interleave
: whether to interleave overlay; default is true when overlay_alpha is unspecified

interleave_period
: interleave animation duration per period, only used when overlay is specified; default is 4 seconds

interleave_transition
: interleave animation transition, only used when overlay is specified; choices are 'ease-in-out' (default) and 'linear'

pixel_width
: pixel width resolution; default is 0.5 millimeters

underlay_range, overlay_range
: numeric vectors of two, value ranges of underlay and overlay

...
: passed to internal method

Value

A 'SVG' tag object that can be embedded in shiny applications or plotted directly.

Examples

```
# toy-example:

shape <- c(10, 10, 10)
vox2ras <- matrix(
  c(10, 17.32, 0, -136,
    -17.32, 10, 20, -63,
    0, -20, 0, 100,
    0, 0, 0, 1),
  nrow = 4, byrow = TRUE
)

# continuous
x <- abs(array(sin(seq_len(100) / 10), shape))

underlay <- ieegio::as_ieegio_volume(x, vox2ras = vox2ras)
overlay <- ieegio::as_ieegio_volume(x > 0.2, vox2ras = vox2ras)

if(interactive()) {

  plot_volume_slices(
    underlay, overlay = overlay,
    depths = seq(0, 150, length.out = 4), pixel_width = 5,
    overlay_col = c("#00000000", "#FF000044", "#FF0000FF")
  )
```

```
}

# Require `install_subject("yael_demo_001")`
if(has_rave_subject("YAEL/yael_demo_001")) {

subject <- ravecore::as_rave_subject("YAEL/yael_demo_001",
                                     strict = FALSE)


t1 <- file.path(subject$imaging_path, "coregistration",
                "MRI_reference.nii.gz")

ct <- file.path(subject$imaging_path, "coregistration",
                "CT_RAW.nii.gz")
transform <- read.table(
  file.path(subject$imaging_path, "coregistration",
            "CT_IJK_to_MR_RAS.txt")
)

ct_image_original <- ieegio::read_volume(ct)
ct_image_aligned <- ieegio::as_ieegio_volume(
  ct_image_original[], vox2ras = as.matrix(transform)
)

if(interactive()) {
  plot_volume_slices(
    t1, overlay = ct_image_aligned,
    overlay_col = c("#00000000", "#FF000044", "#FF0000FF"),
    nc = 6
  )
}

}
```

---

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"),
  ...
)
```

Arguments

x	R array, filearray, or 'rave_prepare_power' object created by prepare_subject_power_with_epoc
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_with_epochs only; default is all available electrodes
filebase	where to store the output; default is NULL and is automatically determined

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, frequency, 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 and filearray, 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

```
if( has_rave_subject("demo/DemoSubject") ) {

# The following code need to download additional demo data
# Please see https://rave.wiki/ for more details

repo <- prepare_subject_power_with_epochs(
  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


}
```

`prepare_subject_bare` *'RAVE' repository: basic*

Description

'RAVE' repository: basic

Usage

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

prepare_subject_bare0(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  ...,
  auto_exclude = FALSE,
  quiet = TRUE,
  repository_id = NULL
)
```

Arguments

`subject`	'RAVE' subject
`electrodes`	string or integers indicating electrodes to load
`reference_name`	name of the reference table
`...`	passed to `RAVESubjectBaseRepository` constructor
`auto_exclude`	whether to automatically discard bad channels
`quiet`	see field `quiet`
`repository_id`	see field `repository_id`

Value

A RAVESubjectBaseRepository instance

Examples

```
if ( has_rave_subject("demo/DemoSubject") ) {


repository <- prepare_subject_bare0("demo/DemoSubject",
                                    electrodes = 14:16,
                                    reference_name = "default")

print(repository)

repository$subject
repository$subject$raw_sample_rates

repository$electrode_table

repository$reference_table

electrodes <- repository$electrode_instances

# Channel 14
e <- electrodes$e_14

# referenced voltage
voltage <- e$load_data_with_blocks("008", "voltage")

ravetools::diagnose_channel(voltage, srate = 2000)


}
```

---

prepare_subject_with_blocks

'RAVE' repository: with entire recording blocks

Description

Loads recording blocks - continuous recording chunks, typically a run of minutes.

Usage

```
prepare_subject_with_blocks(
  subject,
  electrodes = NULL,
  blocks = NULL,
```

```
  reference_name = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE
)

prepare_subject_raw_voltage_with_blocks(
  subject,
  electrodes = NULL,
  blocks = NULL,
  reference_name = "noref",
  downsample = NA,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE
)

prepare_subject_voltage_with_blocks(
  subject,
  electrodes = NULL,
  blocks = NULL,
  reference_name = NULL,
  downsample = NA,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE
)

prepare_subject_time_frequency_coefficients_with_blocks(
  subject,
  electrodes = NULL,
  blocks = NULL,
  reference_name = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE
)

prepare_subject_phase_with_blocks(
  subject,
  electrodes = NULL,
  blocks = NULL,
  reference_name = NULL,
  ...,
```

```
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE
)

prepare_subject_power_with_blocks(
  subject,
  electrodes = NULL,
  blocks = NULL,
  reference_name = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE
)
```

Arguments

`subject`	'RAVE' subject
`electrodes`	string or integers indicating electrodes to load
`blocks`	names of the recording blocks to load, can be queried via `subject$blocks`
`reference_name`	name of the reference table
`...`	passed to `RAVESubjectBaseRepository` constructor
`quiet`	see field `quiet`
`repository_id`	see field `repository_id`
`strict`	whether to check existence of subject before loading data; default is true
`downsample`	positive integer or `NA`, indicating whether the signals should be down-sampled during loading, for voltage traces only; default is `NA`, meaning no down-sampling

Details

`prepare_subject_with_blocks` does not actually load any signal data. Its existence is simply for backward compatibility. It instantiates a super-class of the rest of methods. Therefore, please refer to the rest of the methods for loading specific data types.

If you do not need to analyze super high-frequency signals, it is recommended to set a proper `downsample` value to down-sample the signals while loading voltage traces. This helps optimizing the data storage and speed up computation (significantly). For example, suppose you have 200 channels sampled at 30,000 Hz, a 30-minute recording will cost around 80+ gigabyte memory only to store, let along the storage needed to compute analyses and time needed to perform those analyses. Down-sampling the channels helps a lot. If you are mostly interested in signals below 100 Hz, then down-sampling voltage traces to 400 Hz will preserve the frequency components needed, and it takes 1.2 gigabytes to hold the same recording in memory.

Due to the large-data nature of blocks of signals, the repository will prepare cache files for all the channels, allowing users to load the cached data later without needing to reload

Value

A RAVESubjectRecordingBlockRepository instance

Examples

```
if( has_rave_subject("demo/DemoSubject") ) {


# ---- An use-case example ------------------------------------------------
# Install subject via install_subject("DemoSubject")
subject <- as_rave_subject("demo/DemoSubject")

# list all blocks
subject$blocks

repository <- prepare_subject_voltage_with_blocks(
  subject,
  electrodes = 13:16,
  blocks = "008",
  reference = "default"
)

print(repository)

repository$blocks

# get data
container <- repository$get_container()

# block data
container$`008`
lfp_list <- container$`008`$LFP
channel_sample_rate <- lfp_list$sample_rate

# Even we only load channels 14-16, all the channels are here for
# in case we want to use the cache for future purposes
lfp_list$dimnames$Electrode

# Plot all loaded channels
channel_sel <- lfp_list$dimnames$Electrode %in% c(14, 15, 16)
channel_signals <- lfp_list$data[, channel_sel,
                                 drop = FALSE,
                                 dimnames = FALSE]

ravetools::plot_signals(t(channel_signals),
                        sample_rate = channel_sample_rate,
                        channel_names = 14:16)

# Load channel 14 and plot pwelch
channel_sel <- lfp_list$dimnames$Electrode == 14

channel_signals <- lfp_list$data[, channel_sel,
```

```
                                drop = TRUE,
                                dimnames = FALSE]

ravetools::diagnose_channel(channel_signals,
                            srate = channel_sample_rate,
                            name = "Channel 14",
                            nclass = 30)

# ---- Use cache ------------------------------------------------

subject <- as_rave_subject("demo/DemoSubject")

# Lazy-load block 008
repository <- prepare_subject_voltage_with_blocks(
  subject,
  electrodes = 13:16,
  blocks = "008",
  reference = "default",
  lazy_load = TRUE  # <-- trick
)

# Immediately load data with force=FALSE to use cache if exists
repository$mount_data(force = FALSE)

# ---- More examples --------------------------------------------


subject <- as_rave_subject("demo/DemoSubject")
repository <- prepare_subject_power_with_blocks(
  subject,
  electrodes = 14,
  blocks = "008",
  reference_name = "default"
)

block_008 <- repository$power$`008`$LFP

channel_sel <- block_008$dimnames$Electrode == 14

# Drop electrode margin
power <- block_008$data[, , channel_sel,
                        drop = TRUE, dimnames = FALSE]

# global baseline
power_baselined_t <- 10 * log10(t(power))
power_baselined_t <- power_baselined_t - rowMeans(power_baselined_t)

ravetools::plot_signals(
  power_baselined_t,
  sample_rate = block_008$sample_rate,
  channel_names = block_008$dimnames$Frequency,
  space = 1,
  start_time = 20,
```

```
  duration = 30, ylab = "Frequency",
  main = "Channel 14 - Power with Global Baseline (20-50 sec)"
)


}
```

`prepare_subject_with_epochs` *'RAVE' repository: with epochs*

Description

'RAVE' repository: with epochs

Usage

```
prepare_subject_with_epochs(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  epoch_name = NULL,
  time_windows = NULL,
  stitch_events = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE
)

prepare_subject_raw_voltage_with_epochs(
  subject,
  electrodes = NULL,
  epoch_name = NULL,
  time_windows = NULL,
  stitch_events = NULL,
  ...,
  quiet = TRUE,
  repository_id = NULL,
  strict = TRUE
)

prepare_subject_voltage_with_epochs(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  epoch_name = NULL,
```

```
  time_windows = NULL,
  stitch_events = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE
)

prepare_subject_time_frequency_coefficients_with_epochs(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  epoch_name = NULL,
  time_windows = NULL,
  stitch_events = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE
)

prepare_subject_power_with_epochs(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  epoch_name = NULL,
  time_windows = NULL,
  stitch_events = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE
)

prepare_subject_power(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  epoch_name = NULL,
  time_windows = NULL,
  stitch_events = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE
)

prepare_subject_phase_with_epochs(
```

```
  subject,
  electrodes = NULL,
  reference_name = NULL,
  epoch_name = NULL,
  time_windows = NULL,
  stitch_events = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE
)
```

Arguments

subject	'RAVE' subject
electrodes	string or integers indicating electrodes to load
reference_name	name of the reference table
epoch_name	name of the epoch trial table
time_windows	numeric vector with even lengths, the time start and end of the trials, for example, c(-1, 2) means load 1 second before the trial onset and 2 seconds after trial onset
stitch_events	events where the time_windows is based; default is trial onset (NULL)
...	passed to RAVESubjectBaseRepository constructor
quiet	see field quiet
repository_id	see field repository_id
strict	whether to check existence of subject before loading data; default is true

Value

A RAVESubjectEpochRepository instance

Examples

```
if ( has_rave_subject("demo/DemoSubject") ) {


repository <- prepare_subject_with_epochs(
  "demo/DemoSubject", electrodes = 14:16,
  reference_name = "default", epoch_name = "auditory_onset",
  time_windows = c(-1, 2))

print(repository)

head(repository$epoch$table)

electrodes <- repository$electrode_instances

# Channel 14
```

```
e <- electrodes$e_14

# referenced voltage
voltage <- e$load_data_with_epochs("voltage")

# 6001 time points (2000 sample rate)
# 287 trials
# 1 channel
dim(voltage)

ravetools::plot_signals(t(voltage[, 1:10, 1]),
                        sample_rate = 2000,
                        ylab = "Trial",
                        main = "First 10 trials")

}
```

`py_nipy_coreg` *Register a computerized tomography (CT) image to MRI via 'nipy'*

Description

Align `'CT'` using `nipy.algorithms.registration.histogram_registration`.

Usage

```
py_nipy_coreg(
  ct_path,
  mri_path,
  clean_source = TRUE,
  inverse_target = TRUE,
  precenter_source = TRUE,
  smooth = 0,
  reg_type = c("rigid", "affine"),
  interp = c("pv", "tri"),
  similarity = c("crl1", "cc", "cr", "mi", "nmi", "slr"),
  optimizer = c("powell", "steepest", "cg", "bfgs", "simplex"),
  tol = 1e-04,
  dry_run = FALSE
)

cmd_run_nipy_coreg(
  subject,
  ct_path,
  mri_path,
  clean_source = TRUE,
  inverse_target = TRUE,
  precenter_source = TRUE,
```

```
  reg_type = c("rigid", "affine"),
  interp = c("pv", "tri"),
  similarity = c("crl1", "cc", "cr", "mi", "nmi", "slr"),
  optimizer = c("powell", "steepest", "cg", "bfgs", "simplex"),
  dry_run = FALSE,
  verbose = FALSE
)
```

Arguments

`ct_path`, `mri_path`
: absolute paths to 'CT' and 'MR' image files

`clean_source`
: whether to replace negative 'CT' values with zeros; default is true

`inverse_target`
: whether to inverse 'MRI' color intensity; default is true

`precenter_source`
: whether to adjust the 'CT' transform matrix before alignment, such that the origin of 'CT' is at the center of the volume; default is true. This option may avoid the case that 'CT' is too far-away from the 'MR' volume at the beginning of the optimization

`smooth`, `interp`, `optimizer`, `tol`
: optimization parameters, see `'nipy'` documentation for details.

`reg_type`
: registration type, choices are `'rigid'` or `'affine'`

`similarity`
: the cost function of the alignment; choices are `'crl1'` ('L1' regularized correlation), `'cc'` (correlation coefficient), `'cr'` (correlation), `'mi'` (mutual information), `'nmi'` (normalized mutual information), `'slr'` (likelihood ratio). In reality I personally find `'crl1'` works best in most cases, though many tutorials suggest `'nmi'`.

`dry_run`
: whether to dry-run the script and to print out the command instead of executing the code; default is false

`subject`
: 'RAVE' subject

`verbose`
: whether to verbose command; default is false

Value

Nothing is returned from the function. However, several files will be generated at the 'CT' path:

`'ct_in_t1.nii'` aligned 'CT' image; the image is also re-sampled into 'MRI' space

`'CT_IJK_to_MR_RAS.txt'` transform matrix from volume 'IJK' space in the original 'CT' to the 'RAS' anatomical coordinate in 'MR' scanner

`'CT_RAS_to_MR_RAS.txt'` transform matrix from scanner 'RAS' space in the original 'CT' to 'RAS' in 'MR' scanner space

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.

Value

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.

Super class

`ravepipeline::RAVESerializable` -> `RAVEAbstarctElectrode`

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

`stitch_events` events to stitch, when loading window is not default to trial onset; must be `NULL` or a character vector of length 2

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', 'Multitaper', etc.

`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

`cache_root` run-time cache path; `NA` if epoch or trial intervals are missing

`trial_intervals` trial intervals relative to epoch onset

Methods

Public methods:

- RAVEAbstarctElectrode$new()
- RAVEAbstarctElectrode$set_reference()
- RAVEAbstarctElectrode$set_epoch()
- RAVEAbstarctElectrode$clear_cache()
- RAVEAbstarctElectrode$clear_memory()
- RAVEAbstarctElectrode$load_data_with_epochs()
- RAVEAbstarctElectrode$load_data()
- RAVEAbstarctElectrode$load_dimnames_with_epochs()
- RAVEAbstarctElectrode$load_data_with_blocks()
- RAVEAbstarctElectrode$load_blocks()
- RAVEAbstarctElectrode$load_dim_with_blocks()
- RAVEAbstarctElectrode$clone()

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

Usage:

```
RAVEAbstarctElectrode$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:

```
RAVEAbstarctElectrode$set_reference(reference)
```

Arguments:

`reference` NULL or RAVEAbstarctElectrode instance

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

Usage:

```
RAVEAbstarctElectrode$set_epoch(epoch, stitch_events = NULL)
```

Arguments:

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

`stitch_events` events to stitch, default is NULL, meaning when loading data, the time is relative to the trial onset (column "Time" in the epoch file); set to a character of length 2, representing the events if time is not relative to trial onset. Please remove the prefix. For example, for a column named "Event_name", the event name is "name".

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

Usage:

```
RAVEAbstarctElectrode$clear_cache(...)
```

Arguments:

... implemented by child instances

Method `clear_memory()`: method to clear memory

Usage:

```
RAVEAbstarctElectrode$clear_memory(...)
```

Arguments:

... implemented by child instances

Method `load_data_with_epochs()`: method to load electrode data

Usage:

```
RAVEAbstarctElectrode$load_data_with_epochs(type)
```

Arguments:

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

Method `load_data()`: alias of `load_data_with_epochs` for legacy code

Usage:

```
RAVEAbstarctElectrode$load_data(type)
```

Arguments:

`type` see `load_data_with_epochs`

Method `load_dimnames_with_epochs()`: get expected dimension names

Usage:

```
RAVEAbstarctElectrode$load_dimnames_with_epochs(type)
```

Arguments:

`type` see `load_data_with_epochs`

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

Usage:

```
RAVEAbstarctElectrode$load_data_with_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

Method `load_blocks()`: alias of `load_data_with_blocks` for legacy code

Usage:

```
RAVEAbstarctElectrode$load_blocks(blocks, type, simplify = TRUE)
```

Arguments:

`blocks`, `type`, `simplify` see `load_data_with_blocks`

Method `load_dim_with_blocks()`: get expected dimension information for block-based loader

Usage:

```
RAVEAbstarctElectrode$load_dim_with_blocks(blocks, type)
```

Arguments:

`blocks, type` see `load_data_with_blocks`

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

Usage:

```
RAVEAbstarctElectrode$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
if( has_rave_subject("demo/DemoSubject") ) {

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

generator <- RAVEAbstarctElectrode

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

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

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

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

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

# load & set reference
ref <- generator$new(e$subject, ref_name)
e$set_reference(ref)

}
```

ravecore-constants *'RAVE' constants*

Description

Constant variables

Usage

```
LOCATION_TYPES

SIGNAL_TYPES

IMPORT_FORMATS

YAEL_IMAGE_TYPES

MNI305_to_MNI152
```

Format

An object of class `character` of length 5.

An object of class `character` of length 5.

An object of class `list` of length 7.

An object of class `character` of length 10.

An object of class `matrix` (inherits from `array`) with 4 rows and 4 columns.

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").

Value

`self$table`

If `event` is one of `"trial onset"`, `"default"`, `""`, or `NULL`, then the result will be `"Time"` column; if the event is found, then return will be the corresponding event column. When the event is not found and `missing` is `"error"`, error will be raised; default is to return `"Time"` column, as it's trial onset and is mandatory.

If `condition_type` is one of `"default"`, `""`, or `NULL`, then the result will be `"Condition"` column; if the condition type is found, then return will be the corresponding condition type column. When the condition type is not found and `missing` is `"error"`, error will be raised; default is to return `"Condition"` column, as it's the default and is mandatory.

Super class

`ravepipeline::RAVESerializable` -> `RAVEEpoch`

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

`available_events` available events other than trial onset

`available_condition_type` available condition type other than the default

Methods

Public methods:

- `RAVEEpoch$@marshal()`
- `RAVEEpoch$@unmarshal()`
- `RAVEEpoch$new()`
- `RAVEEpoch$trial_at()`
- `RAVEEpoch$update_table()`
- `RAVEEpoch$set_trial()`
- `RAVEEpoch$get_event_colname()`
- `RAVEEpoch$get_condition_colname()`
- `RAVEEpoch$clone()`

Method `@marshal()`**:** Internal method

Usage:

```
RAVEEpoch$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
RAVEEpoch$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

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()
```

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 `get_event_colname()`**:** Get epoch column name that represents the desired event

Usage:

```
RAVEEpoch$get_event_colname(
  event = "",
  missing = c("warning", "error", "none")
)
```

Arguments:

`event` a character string of the event, see `$available_events` for all available events; set to `"trial onset"`, `"default"`, or blank to use the default

`missing` what to do if event is missing; default is to warn

Method `get_condition_colname()`**:** Get condition column name that represents the desired condition type

Usage:

```
RAVEEpoch$get_condition_colname(
  condition_type = "default",
  missing = c("error", "warning", "none")
)
```

Arguments:

`condition_type` a character string of the condition type, see `$available_condition_type` for all available condition types; set to `"default"` or blank to use the default

`missing` what to do if condition type is missing; default is to warn if the condition column is not found.

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

if(has_rave_subject("demo/DemoSubject")) {

# 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
list(old_trial1, new_trial1)

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

}
```

RAVEPreprocessSettings *Defines preprocess configurations*

Description

R6 class definition

Value

list of electrode type, number, etc.

NULL when no channel is composed. When flat is TRUE, a data frame of weights with the columns composing electrode channel numbers, composed channel number, and corresponding weights; if flat is FALSE, then a weight matrix;

Super class

ravepipeline::RAVESerializable -> RAVEPreprocessSettings

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

electrode_composed composed electrode channels, not actual physically contacts, but is generated from those physically ones

`wavelet_params` wavelet parameters

`notch_params` Notch filter parameters

`electrode_types` electrode signal types

`@freeze_blocks` whether to free block, internally used

`@freeze_lfp_ecog` whether to freeze electrodes that record 'LFP' 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_path2` raw data path, based on the format standard; for native, this is equivalent to `raw_path`; for 'BIDS', this is subject raw directory in `'BIDS'` project

`raw_path2_type` raw data path type, 'native' or 'bids'

`raw_path` legacy raw data path for 'RAVE', regardless of `raw_path2_type`. This field exists for compatibility support the legacy scripts. Please use `raw_path2` combined with `raw_path2_type` for supporting 'BIDS' format

`raw_path_type` legacy type for `raw_path`, always returns `'native'`

Methods

Public methods:

- `RAVEPreprocessSettings$@marshal()`
- `RAVEPreprocessSettings$@unmarshal()`
- `RAVEPreprocessSettings$new()`
- `RAVEPreprocessSettings$valid()`
- `RAVEPreprocessSettings$has_raw()`
- `RAVEPreprocessSettings$set_blocks()`
- `RAVEPreprocessSettings$get_block_paths()`
- `RAVEPreprocessSettings$set_electrodes()`
- `RAVEPreprocessSettings$set_sample_rates()`
- `RAVEPreprocessSettings$migrate()`
- `RAVEPreprocessSettings$electrode_info()`
- `RAVEPreprocessSettings$save()`
- `RAVEPreprocessSettings$get_compose_weights()`
- `RAVEPreprocessSettings$clone()`

Method `@marshal()`**:** Internal method

Usage:

```
RAVEPreprocessSettings$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
RAVEPreprocessSettings$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

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 `get_block_paths()`**:** get block-related files

Usage:

```
RAVEPreprocessSettings$get_block_paths(
  block,
  force_native = FALSE,
  check = TRUE
)
```

Arguments:

`block` block names (for all available blocks, see `all_blocks`)

`force_native` whether to ignore the format standard, such as 'BIDS' and force return the native paths; default is false

`check` whether to check the file paths to make sure the returned paths are valid; default is true

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

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

Usage:

```
RAVEPreprocessSettings$save()
```

Method `get_compose_weights()`**:** get weights of each composed channels

Usage:

```
RAVEPreprocessSettings$get_compose_weights(flat = TRUE)
```

Arguments:

`flat` whether to flatten the data frame; default is true

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

Usage:

```
RAVEPreprocessSettings$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

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

if( has_rave_subject("demo/DemoSubject") ) {

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

}
```

RAVEProject *Definition for 'RAVE' project class*

Description

See as_rave_project for creating 'RAVE' project class

Value

character vector

true or false whether subject is in the project

A data table of pipeline time-stamps and directories

Super class

ravepipeline::RAVESerializable -> RAVEProject

Active bindings

`path` project folder, absolute path

`name` project name, character

`pipeline_path` path to pipeline scripts under project's folder

`format_standard` storage format, can be either `'native'` or `'bids'`-compliant

`@impl` the internal object

Methods

Public methods:

- RAVEProject$@marshal()
- RAVEProject$@unmarshal()
- RAVEProject$print()
- RAVEProject$format()
- RAVEProject$new()
- RAVEProject$subjects()
- RAVEProject$has_subject()
- RAVEProject$group_path()
- RAVEProject$subject_pipelines()
- RAVEProject$clone()

Method `@marshal()`**:** Internal method

Usage:

```
RAVEProject$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
RAVEProject$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

Method `print()`**:** override print method

Usage:

```
RAVEProject$print(...)
```

Arguments:

`...` ignored

Method `format()`**:** override format method

Usage:

```
RAVEProject$format(...)
```

Arguments:

`...` ignored

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

Usage:

```
RAVEProject$new(project_name, strict = TRUE, parent_path = NULL)
```

Arguments:

`project_name` character

`strict` whether to check project path

`parent_path` `NULL`, a path to the project parent folder for native projects, or the path to 'BIDS' root directory.

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

Usage:

```
RAVEProject$subjects()
```

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

Usage:

```
RAVEProject$has_subject(subject_code)
```

Arguments:

`subject_code` character, subject name

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 `subject_pipelines()`**:** list saved pipelines

Usage:

```
RAVEProject$subject_pipelines(
  pipeline_name,
  cache = FALSE,
  check = TRUE,
  all = FALSE
)
```

Arguments:

`pipeline_name` name of the pipeline

`cache` whether to use cached registry

`check` whether to check if the pipelines exist as directories

`all` whether to list all pipelines; default is false; pipelines with the same label but older time-stamps will be hidden

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

Value

data frame

integer vector of valid electrodes

The same as value

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

A data frame with four columns: 'namespace' for the group name of the entry (entries within the same namespace usually share same module), 'timestamp' for when the entry was registered. 'entry_name' is the name of the entry. If include_history is true, then multiple entries with the same 'entry_name' might appear since the obsolete entries are included. 'entry_value' is the value of the corresponding entry.

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.

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.

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.

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

A table of pipeline registry

A PipelineTools instance

Super class

ravepipeline::RAVESerializable -> RAVESubject

Active bindings

@impl the internal object

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

`imaging_path` root path to imaging processing folder

`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

`report_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

`electrode_composed` composed electrode channels, not actual physically contacts, but is generated from those physically ones

Methods

Public methods:

- `RAVESubject$@marshal()`
- `RAVESubject$@unmarshal()`
- `RAVESubject$print()`
- `RAVESubject$new()`
- `RAVESubject$meta_data()`
- `RAVESubject$valid_electrodes()`
- `RAVESubject$initialize_paths()`
- `RAVESubject$set_default()`
- `RAVESubject$get_default()`
- `RAVESubject$get_note_summary()`
- `RAVESubject$get_epoch()`
- `RAVESubject$get_reference()`
- `RAVESubject$get_electrode_table()`

- RAVESubject$get_frequency()
- RAVESubject$list_pipelines()
- RAVESubject$load_pipeline()
- RAVESubject$clone()

Method `@marshal()`: Internal method

Usage:

```
RAVESubject$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`: Internal method

Usage:

```
RAVESubject$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

Method `print()`: override print method

Usage:

```
RAVESubject$print(...)
```

Arguments:

`...` ignored

Method `new()`: constructor

Usage:

```
RAVESubject$new(
  project_name,
  subject_code = NULL,
  strict = TRUE,
  parent_path = NULL
)
```

Arguments:

`project_name` character project name

`subject_code` character subject code

`strict` whether to check if subject folders exist

`parent_path` parent path if no default path is used, this is for the root directory if subject is in 'BIDS' format

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",
  strict = TRUE
)
```

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'`.

`strict` whether to raise errors if the files are missing; default is true; alternative is to return `NULL` on missing

Method `valid_electrodes()`**:** get valid electrode numbers

Usage:

```
RAVESubject$valid_electrodes(reference_name = NULL, 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

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)

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)

Method `get_note_summary()`: get summary table of all the key-value pairs used by 'RAVE' modules for the subject

Usage:

```
RAVESubject$get_note_summary(namespaces, include_history = FALSE)
```

Arguments:

`namespaces` namespaces for the entries; see method `get_default` or `set_default`. Default is all possible namespaces

`include_history` whether to include history entries; default is false

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

Usage:

```
RAVESubject$get_epoch(
  epoch_name = "default",
  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

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

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,
  warn = TRUE
)
```

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

`warn` whether to warn about missing electrodes; default is true

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

Method `list_pipelines()`**:** list saved pipelines

Usage:

```
RAVESubject$list_pipelines(
  pipeline_name,
  cache = FALSE,
  check = TRUE,
  all = FALSE
)
```

Arguments:

`pipeline_name` pipeline ID

`cache` whether to use cache registry to speed up

`check` whether to check if the pipelines exist

`all` whether to list all pipelines; default is false; pipelines with the same label but older timestamps will be hidden

Method `load_pipeline()`**:** load saved pipeline

Usage:

```
RAVESubject$load_pipeline(directory)
```

Arguments:

`directory` pipeline directory name

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`

RAVESubjectBaseRepository *'RAVE' class for base repository*

Description

The class is for creating child classes, to instantiate the class, please use `prepare_subject_bare0` to create base repository.

Value

The root directory where the files are stored.

Super class

`ravepipeline::RAVESerializable` -> `RAVESubjectRepository`

Public fields

`@restored` internal flag indicating whether the repository is restored from serialization. Repositories restored from serialization will behave differently (slightly) for performance considerations

`repository_id` repository identifier, typically generated with random string

`quiet` whether to suppress update warning messages, when requested electrodes are not fully processed or excluded

Active bindings

`auto_exclude` whether to automatically discard channels that are marked as "excluded" (such as bad channels or channels that should not be analyzed); default is often true

`meta_info` list of meta information

`needs_update` write-only attribute when subject needs to be reloaded from the disk and reference table needs to be updated, use `repo$needs_update <- TRUE`

`project` project instance, see `RAVEProject`

`subject` subject instance, see `RAVESubject`

`electrode_list` integer vector of electrodes included

`electrode_table` the entire electrode table

`electrode_signal_types` more accurate name should be "channel" signal types: currently returns `'LFP'`, `'Auxiliary'`, or `'Spike'`, for each channel

`electrode_instances` electrode channel instance helpers for loading electrode data

`reference_name` name of reference table

`reference_table` reference table

`references_list` a vector of reference channel names, used together with `reference_instances`

`reference_instances` instances of reference channels, for referencing on the fly, used for `electrode_instances`

`digest_key` a list of repository data used to generate repository signature

`signature` signature of the repository, two repositories might share the same signature if their contents are the same (even with different identifiers); generated from `digest_key`

Methods

Public methods:

- `RAVESubjectBaseRepository$@get_container()`
- `RAVESubjectBaseRepository$@marshal()`
- `RAVESubjectBaseRepository$@unmarshal()`
- `RAVESubjectBaseRepository$print()`
- `RAVESubjectBaseRepository$new()`
- `RAVESubjectBaseRepository$export_matlab()`
- `RAVESubjectBaseRepository$clone()`

Method `@get_container()`**:** Internal method, do not use it directly

Usage:

```
RAVESubjectBaseRepository$@get_container()
```

Method `@marshal()`**:** Internal method

Usage:

```
RAVESubjectBaseRepository$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
RAVESubjectBaseRepository$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

Method `print()`**:** User-friendly print method

Usage:

```
RAVESubjectBaseRepository$print()
```

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

Usage:

```
RAVESubjectBaseRepository$new(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  ...,
  auto_exclude = TRUE,
```

```
  quiet = TRUE,
  repository_id = NULL,
  strict = TRUE,
  .class = NULL
)
```

Arguments:

`subject` 'RAVE' subject

`electrodes` string or integers indicating electrodes to load

`reference_name` name of the reference table

`...` reserved, currently ignored

`auto_exclude` whether to automatically discard bad channels

`quiet` see field `quiet`

`repository_id` see field `repository_id`

`strict` whether the mode should be strict; default is true and errors out when subject is missing

`.class` internally used, do not set, even if you know what this is

Method `export_matlab()`**:** Export the repository to 'Matlab' for future analysis

Usage:

```
RAVESubjectBaseRepository$export_matlab(..., verbose = TRUE)
```

Arguments:

`...` reserved for child classes

`verbose` print progresses

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

Usage:

```
RAVESubjectBaseRepository$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`prepare_subject_bare0`

`RAVESubjectEpochPhaseRepository` *'RAVE' class for epoch repository - time-frequency phase*

Description

The repository inherits `link{RAVESubjectEpochTimeFreqBaseRepository}`, with epoch trials, and is intended for loading processed and referenced time-frequency coefficients. Use `prepare_subject_phase_with_epoc` to create an instance.

Super classes

ravepipeline::RAVESerializable -> ravecore::RAVESubjectRepository -> ravecore::RAVESubjectEpochReposit
-> ravecore::RAVESubjectEpochTimeFreqBaseRepository -> RAVESubjectEpochPhaseRepository

Active bindings

`phase` a named map of time-frequency coefficient phase, mounted by `mount_data`

Methods

Public methods:

- `RAVESubjectEpochPhaseRepository$@marshal()`
- `RAVESubjectEpochPhaseRepository$@unmarshal()`
- `RAVESubjectEpochPhaseRepository$new()`
- `RAVESubjectEpochPhaseRepository$clone()`

Method `@marshal()`**:** Internal method

Usage:

```
RAVESubjectEpochPhaseRepository$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
RAVESubjectEpochPhaseRepository$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

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

Usage:

```
RAVESubjectEpochPhaseRepository$new(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  epoch_name = NULL,
  time_windows = NULL,
  stitch_events = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE,
  lazy_load = FALSE,
  .class = NULL
)
```

Arguments:

`subject` 'RAVE' subject

`electrodes` string or integers indicating electrodes to load

`reference_name` name of the reference table

`epoch_name` name of the epoch trial table

`time_windows` numeric vector with even lengths, the time start and end of the trials, for example, `c(-1, 2)` means load 1 second before the trial onset and 2 seconds after trial onset

`stitch_events` events where the `time_windows` is based; default is trial onset (`NULL`)

`...` passed to `RAVESubjectEpochTimeFreqBaseRepository` constructor

`quiet` see field `quiet`

`repository_id` see field `repository_id`

`strict` whether the mode should be strict; default is true and errors out when subject is missing

`lazy_load` whether to delay `mount_data`; default is false

`.class` internally used, do not set, even if you know what this is

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

Usage:

```
RAVESubjectEpochPhaseRepository$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`RAVESubjectEpochPowerRepository`

'RAVE' class for epoch repository - time-frequency power

Description

The repository inherits `link{RAVESubjectEpochTimeFreqBaseRepository}`, with epoch trials, and is intended for loading processed and referenced time-frequency coefficients. Use `prepare_subject_power_with_epoc` to create an instance.

Super classes

`ravepipeline::RAVESerializable` -> `ravecore::RAVESubjectRepository` -> `ravecore::RAVESubjectEpochReposit` -> `ravecore::RAVESubjectEpochTimeFreqBaseRepository` -> `RAVESubjectEpochPowerRepository`

Active bindings

`power` a named map of time-frequency power spectrogram, mounted by `mount_data`

Methods

Public methods:

- RAVESubjectEpochPowerRepository$@marshal()
- RAVESubjectEpochPowerRepository$@unmarshal()
- RAVESubjectEpochPowerRepository$new()
- RAVESubjectEpochPowerRepository$clone()

Method `@marshal()`**:** Internal method

Usage:

```
RAVESubjectEpochPowerRepository$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
RAVESubjectEpochPowerRepository$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

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

Usage:

```
RAVESubjectEpochPowerRepository$new(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  epoch_name = NULL,
  time_windows = NULL,
  stitch_events = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE,
  lazy_load = FALSE,
  .class = NULL
)
```

Arguments:

`subject` 'RAVE' subject

`electrodes` string or integers indicating electrodes to load

`reference_name` name of the reference table

`epoch_name` name of the epoch trial table

`time_windows` numeric vector with even lengths, the time start and end of the trials, for example, `c(-1, 2)` means load 1 second before the trial onset and 2 seconds after trial onset

`stitch_events` events where the `time_windows` is based; default is trial onset (`NULL`)

`...` passed to RAVESubjectEpochTimeFreqBaseRepository constructor

`quiet` see field `quiet`

`repository_id` see field `repository_id`

`strict` whether the mode should be strict; default is true and errors out when subject is missing

`lazy_load` whether to delay `mount_data`; default is false

`.class` internally used, do not set, even if you know what this is

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

Usage:

```
RAVESubjectEpochPowerRepository$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`RAVESubjectEpochRawVoltageRepository` *'RAVE' class for epoch repository - raw voltage*

Description

The repository inherits `link{RAVESubjectEpochRepository}`, with epoch trials, and is intended for loading raw (without any processing or reference) voltage signals. Use `prepare_subject_raw_voltage_with_epochs` to create an instance.

Super classes

`ravepipeline::RAVESerializable` -> `ravecore::RAVESubjectRepository` -> `ravecore::RAVESubjectEpochReposit` -> `RAVESubjectEpochRawVoltageRepository`

Active bindings

`digest_key` a list of repository data used to generate repository signature

`raw_voltage` a named map of raw voltage data, mounted by `mount_data`, alias of `get_container`

`reference_table` reference table, all channels will be marked as no reference

Methods

Public methods:

- `RAVESubjectEpochRawVoltageRepository$@marshal()`
- `RAVESubjectEpochRawVoltageRepository$@unmarshal()`
- `RAVESubjectEpochRawVoltageRepository$new()`
- `RAVESubjectEpochRawVoltageRepository$mount_data()`
- `RAVESubjectEpochRawVoltageRepository$clone()`

Method `@marshal()`**:** Internal method

Usage:

```
RAVESubjectEpochRawVoltageRepository$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
RAVESubjectEpochRawVoltageRepository$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

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

Usage:

```
RAVESubjectEpochRawVoltageRepository$new(
  subject,
  electrodes = NULL,
  epoch_name = NULL,
  time_windows = NULL,
  stitch_events = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE,
  lazy_load = FALSE,
  reference_name = "noref",
  .class = NULL
)
```

Arguments:

`subject` 'RAVE' subject

`electrodes` string or integers indicating electrodes to load

`epoch_name` name of the epoch trial table

`time_windows` numeric vector with even lengths, the time start and end of the trials, for example, `c(-1, 2)` means load 1 second before the trial onset and 2 seconds after trial onset

`stitch_events` events where the `time_windows` is based; default is trial onset (`NULL`)

`...` passed to `RAVESubjectEpochRepository` constructor

`quiet` see field `quiet`

`repository_id` see field `repository_id`

`strict` whether the mode should be strict; default is true and errors out when subject is missing

`lazy_load` whether to delay calling `mount_data`; default is false

`reference_name` ignored, always `'noref'` for raw voltage data

`.class` internally used, do not set, even if you know what this is

Method `mount_data()`**:** function to mount raw voltage signals

Usage:

```
RAVESubjectEpochRawVoltageRepository$mount_data(
  ...,
  force = TRUE,
  electrodes = NULL
)
```

Arguments:

`...` reserved

`force` force update data; default is true

`electrodes` electrodes to update for expert-use use; default is `NULL` (all electrode channels will be mounted)

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

Usage:

```
RAVESubjectEpochRawVoltageRepository$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`RAVESubjectEpochRepository` *'RAVE' class for epoch repository*

Description

Compared to `RAVESubjectBaseRepository`, this repository requires epoch information. please use `prepare_subject_with_epochs` to instantiate this repository.

Value

The root directory where the files are stored.

A named map, typically with data arrays, shape/dimension information

Super classes

`ravepipeline::RAVESerializable` -> `ravecore::RAVESubjectRepository` -> `RAVESubjectEpochRepository`

Active bindings

`needs_update` write-only attribute when subject needs to be reloaded from the disk and reference table needs to be updated, use `repo$needs_update <- TRUE`

`meta_info` list of meta information

`sample_rates` a named list of sampling frequencies; the names are signal types (`'LFP'`, `'Auxiliary'`, or `'Spike'`) and the values are the sampling frequencies

`sample_rate` a single number of the sample rate; if the electrode channels contain local-field potential `'LFP'` signal type, then the sample rate is the `'LFP'` sample rate; otherwise the sample rate is `'Spike'` channel sample rate, if exists, or whatever comes first. This field is for backward compatibility support, use `sample_rates` for more accurate number

`epoch_name` name of the epoch table

`epoch` `RAVEEpoch` instance

`epoch_table` epoch table, equivalent to `repository$epoch$table`

`stitch_events` events where `time_windows` are based on

`time_windows` list of time ranges to load; the time is relative to `stitch_events`; default is trial onset

`electrode_table` the entire electrode table with reference information

`electrode_instances` electrode channel instance helpers for loading electrode data

`reference_instances` instances of reference channels, for referencing on the fly, used for `electrode_instances`

`digest_key` a list of repository data used to generate repository signature

Methods

Public methods:

- `RAVESubjectEpochRepository$@marshal()`
- `RAVESubjectEpochRepository$@unmarshal()`
- `RAVESubjectEpochRepository$new()`
- `RAVESubjectEpochRepository$export_matlab()`
- `RAVESubjectEpochRepository$set_epoch()`
- `RAVESubjectEpochRepository$mount_data()`
- `RAVESubjectEpochRepository$get_container()`
- `RAVESubjectEpochRepository$clone()`

Method `@marshal()`**:** Internal method

Usage:

```
RAVESubjectEpochRepository$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
RAVESubjectEpochRepository$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

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

Usage:

```
RAVESubjectEpochRepository$new(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  epoch_name = NULL,
  time_windows = NULL,
  stitch_events = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE,
  lazy_load = FALSE,
  .class = NULL
)
```

Arguments:

`subject` 'RAVE' subject

`electrodes` string or integers indicating electrodes to load

`reference_name` name of the reference table

`epoch_name` name of the epoch trial table

`time_windows` numeric vector with even lengths, the time start and end of the trials, for example, `c(-1, 2)` means load 1 second before the trial onset and 2 seconds after trial onset

`stitch_events` events where the `time_windows` is based; default is trial onset (`NULL`)

`...` passed to `RAVESubjectBaseRepository` constructor

`quiet` see field `quiet`

`repository_id` see field `repository_id`

`strict` whether the mode should be strict; default is true and errors out when subject is missing

`lazy_load` whether to delay (lazy) the evaluation `mount_data`

`.class` internally used, do not set, even if you know what this is

Method `export_matlab()`**:** Export the repository to 'Matlab' for future analysis

Usage:

```
RAVESubjectEpochRepository$export_matlab(..., verbose = TRUE)
```

Arguments:

`...` reserved for child classes

`verbose` print progresses

Method `set_epoch()`**:** change trial epoch profiles

Usage:

```
RAVESubjectEpochRepository$set_epoch(epoch_name, stitch_events = NULL)
```

Arguments:

`epoch_name` name of epoch table

`stitch_events` events to stitch

Method `mount_data()`**:** function to mount data, not doing anything in this class, but may be used by child classes

Usage:

```
RAVESubjectEpochRepository$mount_data(..., force = TRUE, electrodes = NULL)
```

Arguments:

`...` reserved

`force` force update data; default is true

`electrodes` electrodes to update; default is `NULL` (all electrode channels)

Method `get_container()`**:** get container where loaded data are stored

Usage:

```
RAVESubjectEpochRepository$get_container(force = FALSE, ...)
```

Arguments:

`force, ...` passed to `mount_data`

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

Usage:

```
RAVESubjectEpochRepository$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`prepare_subject_with_epochs`

`RAVESubjectEpochTimeFreqBaseRepository` *'RAVE' class for epoch repository - time-frequency (internal)*

Description

The repository inherits `link{RAVESubjectEpochRepository}`, with epoch trials, and is intended for loading processed and referenced time-frequency coefficients.

Super classes

`ravepipeline::RAVESerializable` -> `ravecore::RAVESubjectRepository` -> `ravecore::RAVESubjectEpochReposit` -> `RAVESubjectEpochTimeFreqBaseRepository`

Active bindings

`sample_rate` time-frequency coefficient sample rate

`frequency` frequencies where the time-frequency coefficients are evaluated

`time` time in seconds for each trial

`time_points` see `time` field, existed for backward compatibility

`signal_type` do not use

`digest_key` a list of repository data used to generate repository signature

Methods

Public methods:

- RAVESubjectEpochTimeFreqBaseRepository$@marshal()
- RAVESubjectEpochTimeFreqBaseRepository$@unmarshal()
- RAVESubjectEpochTimeFreqBaseRepository$new()
- RAVESubjectEpochTimeFreqBaseRepository$mount_data()
- RAVESubjectEpochTimeFreqBaseRepository$clone()

Method `@marshal()`**:** Internal method

Usage:

```
RAVESubjectEpochTimeFreqBaseRepository$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
RAVESubjectEpochTimeFreqBaseRepository$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

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

Usage:

```
RAVESubjectEpochTimeFreqBaseRepository$new(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  epoch_name = NULL,
  time_windows = NULL,
  stitch_events = NULL,
  data_type = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE,
  lazy_load = FALSE,
  .class = NULL
)
```

Arguments:

`subject` 'RAVE' subject

`electrodes` string or integers indicating electrodes to load

`reference_name` name of the reference table

`epoch_name` name of the epoch trial table

`time_windows` numeric vector with even lengths, the time start and end of the trials, for example, `c(-1, 2)` means load 1 second before the trial onset and 2 seconds after trial onset

`stitch_events` events where the `time_windows` is based; default is trial onset (`NULL`)

`data_type` for child classes to fill; data type (power, phase, or complex time-frequency coefficients)

`...` passed to `RAVESubjectEpochRepository` constructor

`quiet` see field `quiet`

`repository_id` see field `repository_id`

`strict` whether the mode should be strict; default is true and errors out when subject is missing

`lazy_load` whether to delay `mount_data`; default is false

`.class` internally used, do not set, even if you know what this is

Method `mount_data()`**:** function to mount processed and referenced spectrogram

Usage:

```
RAVESubjectEpochTimeFreqBaseRepository$mount_data(
  ...,
  force = TRUE,
  electrodes = NULL
)
```

Arguments:

`...` reserved

`force` force update data; default is true

`electrodes` electrodes to update for expert-use use; default is `NULL` (all electrode channels will be mounted)

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

Usage:

```
RAVESubjectEpochTimeFreqBaseRepository$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`RAVESubjectEpochTimeFreqCoefRepository`

'RAVE' class for epoch repository - time-frequency

Description

The repository inherits `link{RAVESubjectEpochTimeFreqBaseRepository}`, with epoch trials, and is intended for loading processed and referenced time-frequency coefficients. Use `prepare_subject_time_frequency_` to create an instance.

Super classes

`ravepipeline::RAVESerializable` -> `ravecore::RAVESubjectRepository` -> `ravecore::RAVESubjectEpochReposit` -> `ravecore::RAVESubjectEpochTimeFreqBaseRepository` -> `RAVESubjectEpochTimeFreqCoefRepository`

Active bindings

`coefficients` a named map of time-frequency coefficient data, mounted by `mount_data`

`wavelet` not used anymore, see `coefficients`

Methods

Public methods:

- `RAVESubjectEpochTimeFreqCoefRepository$@marshal()`
- `RAVESubjectEpochTimeFreqCoefRepository$@unmarshal()`
- `RAVESubjectEpochTimeFreqCoefRepository$new()`
- `RAVESubjectEpochTimeFreqCoefRepository$clone()`

Method `@marshal()`**:** Internal method

Usage:

```
RAVESubjectEpochTimeFreqCoefRepository$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
RAVESubjectEpochTimeFreqCoefRepository$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

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

Usage:

```
RAVESubjectEpochTimeFreqCoefRepository$new(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  epoch_name = NULL,
  time_windows = NULL,
  stitch_events = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE,
  lazy_load = FALSE,
  .class = NULL
)
```

Arguments:

`subject` 'RAVE' subject

`electrodes` string or integers indicating electrodes to load

`reference_name` name of the reference table

`epoch_name` name of the epoch trial table

`time_windows` numeric vector with even lengths, the time start and end of the trials, for example, `c(-1, 2)` means load 1 second before the trial onset and 2 seconds after trial onset

`stitch_events` events where the `time_windows` is based; default is trial onset (`NULL`)

`...` passed to `RAVESubjectEpochTimeFreqBaseRepository` constructor

`quiet` see field `quiet`

`repository_id` see field `repository_id`

`strict` whether the mode should be strict; default is true and errors out when subject is missing

`lazy_load` whether to delay `mount_data`; default is false

`.class` internally used, do not set, even if you know what this is

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

Usage:

```
RAVESubjectEpochTimeFreqCoefRepository$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`RAVESubjectEpochVoltageRepository`

'RAVE' class for epoch repository - voltage

Description

The repository inherits `link{RAVESubjectEpochRepository}`, with epoch trials, and is intended for loading processed and referenced voltage signals. Use `prepare_subject_voltage_with_epochs` to create an instance.

Super classes

`ravepipeline::RAVESerializable` -> `ravecore::RAVESubjectRepository` -> `ravecore::RAVESubjectEpochReposit` -> `RAVESubjectEpochVoltageRepository`

Active bindings

`digest_key` a list of repository data used to generate repository signature

`voltage` a named map of voltage data, mounted by `mount_data`

Methods

Public methods:

- `RAVESubjectEpochVoltageRepository$@marshal()`
- `RAVESubjectEpochVoltageRepository$@unmarshal()`
- `RAVESubjectEpochVoltageRepository$new()`
- `RAVESubjectEpochVoltageRepository$mount_data()`

- RAVESubjectEpochVoltageRepository$clone()

Method `@marshal()`: Internal method

Usage:

```
RAVESubjectEpochVoltageRepository$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`: Internal method

Usage:

```
RAVESubjectEpochVoltageRepository$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

Method `new()`: constructor

Usage:

```
RAVESubjectEpochVoltageRepository$new(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  epoch_name = NULL,
  time_windows = NULL,
  stitch_events = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE,
  lazy_load = FALSE,
  .class = NULL
)
```

Arguments:

`subject` 'RAVE' subject

`electrodes` string or integers indicating electrodes to load

`reference_name` name of the reference table

`epoch_name` name of the epoch trial table

`time_windows` numeric vector with even lengths, the time start and end of the trials, for example, `c(-1, 2)` means load 1 second before the trial onset and 2 seconds after trial onset

`stitch_events` events where the `time_windows` is based; default is trial onset (`NULL`)

`...` passed to RAVESubjectEpochRepository constructor

`quiet` see field `quiet`

`repository_id` see field `repository_id`

`strict` whether the mode should be strict; default is true and errors out when subject is missing

`lazy_load` whether to delay `mount_data`; default is false

`.class` internally used, do not set, even if you know what this is

Method `mount_data()`: function to mount referenced voltage signals

Usage:

```
RAVESubjectEpochVoltageRepository$mount_data(
  ...,
  force = TRUE,
  electrodes = NULL
)
```

Arguments:

`...` reserved

`force` force update data; default is true

`electrodes` electrodes to update for expert-use use; default is `NULL` (all electrode channels will be mounted)

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

Usage:

```
RAVESubjectEpochVoltageRepository$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`RAVESubjectRecordingBlockPhaseRepository` *'RAVE' class for loading time-frequency phase components*

Description

Loads time-frequency phase

Super classes

`ravepipeline::RAVESerializable` -> `ravecore::RAVESubjectRepository` -> `ravecore::RAVESubjectRecordingBlo`
-> `ravecore::RAVESubjectRecordingBlockTimeFreqBaseRepository` -> `RAVESubjectRecordingBlockPhaseReposit`

Active bindings

`phase` data container, alias of `get_container`

Methods

Public methods:

- `RAVESubjectRecordingBlockPhaseRepository$@marshal()`
- `RAVESubjectRecordingBlockPhaseRepository$@unmarshal()`
- `RAVESubjectRecordingBlockPhaseRepository$new()`
- `RAVESubjectRecordingBlockPhaseRepository$clone()`

Method `@marshal()`: Internal method

Usage:

```
RAVESubjectRecordingBlockPhaseRepository$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
RAVESubjectRecordingBlockPhaseRepository$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

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

Usage:

```
RAVESubjectRecordingBlockPhaseRepository$new(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  blocks = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE,
  lazy_load = FALSE,
  .class = NULL
)
```

Arguments:

`subject` 'RAVE' subject

`electrodes` string or integers indicating electrodes to load

`reference_name` name of the reference table

`blocks` name of the recording blocks to load

`...` passed to `RAVESubjectBaseRepository` constructor

`quiet` see field `quiet`

`repository_id` see field `repository_id`

`strict` whether the mode should be strict; default is true and errors out when subject is missing

`lazy_load` whether to delay (lazy) the evaluation `mount_data`

`.class` internally used, do not set, even if you know what this is

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

Usage:

```
RAVESubjectRecordingBlockPhaseRepository$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`prepare_subject_phase_with_blocks`

RAVESubjectRecordingBlockPowerRepository
'RAVE' class for loading time-frequency power components

Description

Loads time-frequency power

Super classes

ravepipeline::RAVESerializable -> ravecore::RAVESubjectRepository -> ravecore::RAVESubjectRecordingBlo
-> ravecore::RAVESubjectRecordingBlockTimeFreqBaseRepository -> RAVESubjectRecordingBlockPowerReposit

Active bindings

power data container, alias of get_container

Methods

Public methods:

- RAVESubjectRecordingBlockPowerRepository$@marshal()
- RAVESubjectRecordingBlockPowerRepository$@unmarshal()
- RAVESubjectRecordingBlockPowerRepository$new()
- RAVESubjectRecordingBlockPowerRepository$clone()

Method @marshal(): Internal method

Usage:

```
RAVESubjectRecordingBlockPowerRepository$@marshal(...)
```

Arguments:

... internal arguments

Method @unmarshal(): Internal method

Usage:

```
RAVESubjectRecordingBlockPowerRepository$@unmarshal(object, ...)
```

Arguments:

object, ... internal arguments

Method new(): constructor

Usage:

```
RAVESubjectRecordingBlockPowerRepository$new(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  blocks = NULL,
```

```
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE,
  lazy_load = FALSE,
  .class = NULL
)
```

Arguments:

`subject` 'RAVE' subject

`electrodes` string or integers indicating electrodes to load

`reference_name` name of the reference table

`blocks` name of the recording blocks to load

`...` passed to `RAVESubjectBaseRepository` constructor

`quiet` see field `quiet`

`repository_id` see field `repository_id`

`strict` whether the mode should be strict; default is true and errors out when subject is missing

`lazy_load` whether to delay (lazy) the evaluation `mount_data`

`.class` internally used, do not set, even if you know what this is

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

Usage:

```
RAVESubjectRecordingBlockPowerRepository$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`prepare_subject_power_with_blocks`

`RAVESubjectRecordingBlockRawVoltageRepository`

'RAVE' class for blocks of voltage repository

Description

Compared to `RAVESubjectBaseRepository`, this repository loads the entire voltage traces for selected blocks; use `prepare_subject_raw_voltage_with_blocks` to instantiate this repository.

Super classes

`ravepipeline::RAVESerializable` -> `ravecore::RAVESubjectRepository` -> `ravecore::RAVESubjectRecordingBlo`
-> `RAVESubjectRecordingBlockRawVoltageRepository`

Active bindings

`reference_name` name of reference table; always `'noref'`

`reference_table` reference table; a reference table with `'noref'` on all channels

`references_list` a vector of reference channel names; always `'noref'`

`reference_instances` instances of reference channels, empty in this type of repositories

`sample_rates` a named list of sampling frequencies; the names are signal types (`'LFP'`, `'Auxiliary'`, or `'Spike'`) and the values are the sampling frequencies

`raw_voltage` data container, alias of `get_container`

Methods

Public methods:

- `RAVESubjectRecordingBlockRawVoltageRepository$@marshal()`
- `RAVESubjectRecordingBlockRawVoltageRepository$@unmarshal()`
- `RAVESubjectRecordingBlockRawVoltageRepository$new()`
- `RAVESubjectRecordingBlockRawVoltageRepository$mount_data()`
- `RAVESubjectRecordingBlockRawVoltageRepository$clone()`

Method `@marshal()`**:** Internal method

Usage:

```
RAVESubjectRecordingBlockRawVoltageRepository$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
RAVESubjectRecordingBlockRawVoltageRepository$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

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

Usage:

```
RAVESubjectRecordingBlockRawVoltageRepository$new(
  subject,
  electrodes = NULL,
  reference_name = "noref",
  blocks = NULL,
  downsample = NA,
  ...,
  quiet = TRUE,
  repository_id = NULL,
  strict = TRUE,
  lazy_load = FALSE,
  .class = NULL
)
```

Arguments:

`subject` 'RAVE' subject

`electrodes` string or integers indicating electrodes to load

`reference_name` always `'noref'` (no reference); trying to set to other values will result in a warning

`blocks` name of the recording blocks to load

`downsample` down-sample rate by this integer number to save space and speed up computation; typically 'ERP' signals do not need super high sampling frequencies to load; default is `NA` and no down-sample is performed.

`...` passed to `RAVESubjectBaseRepository` constructor

`quiet` see field `quiet`

`repository_id` see field `repository_id`

`strict` whether the mode should be strict; default is true and errors out when subject is missing

`lazy_load` whether to delay (lazy) the evaluation `mount_data`

`.class` internally used, do not set, even if you know what this is

Method `mount_data()`**:** function to mount data

Usage:

```
RAVESubjectRecordingBlockRawVoltageRepository$mount_data(
  ...,
  force = TRUE,
  electrodes = NULL
)
```

Arguments:

`...` reserved

`force` force update data; default is true; set to false to use cache

`electrodes` electrodes to update; default is `NULL` (all electrode channels)

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

Usage:

```
RAVESubjectRecordingBlockRawVoltageRepository$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`prepare_subject_raw_voltage_with_blocks`

RAVESubjectRecordingBlockRepository *'RAVE' class for loading entire recording block repository*

Description

Compared to RAVESubjectBaseRepository, this repository requires specifying block information. please use prepare_subject_with_blocks to instantiate this repository.

Value

The root directory where the files are stored.

A named map, typically with data arrays, shape/dimension information

Super classes

ravepipeline::RAVESerializable -> ravecore::RAVESubjectRepository -> RAVESubjectRecordingBlockReposito

Active bindings

`meta_info` list of meta information

`needs_update` write-only attribute when subject needs to be reloaded from the disk and reference table needs to be updated, use `repo$needs_update <- TRUE`

`blocks` names of recording blocks

`electrode_table` the entire electrode table with reference information

`digest_key` a list of repository data used to generate repository signature

Methods

Public methods:

- RAVESubjectRecordingBlockRepository$@marshal()
- RAVESubjectRecordingBlockRepository$@unmarshal()
- RAVESubjectRecordingBlockRepository$new()
- RAVESubjectRecordingBlockRepository$export_matlab()
- RAVESubjectRecordingBlockRepository$get_container()
- RAVESubjectRecordingBlockRepository$clone()

Method `@marshal()`**:** Internal method

Usage:

```
RAVESubjectRecordingBlockRepository$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
RAVESubjectRecordingBlockRepository$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

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

Usage:

```
RAVESubjectRecordingBlockRepository$new(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  blocks = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE,
  lazy_load = FALSE,
  .class = NULL
)
```

Arguments:

`subject` 'RAVE' subject

`electrodes` string or integers indicating electrodes to load

`reference_name` name of the reference table

`blocks` name of the recording blocks to load

`...` passed to `RAVESubjectBaseRepository` constructor

`quiet` see field `quiet`

`repository_id` see field `repository_id`

`strict` whether the mode should be strict; default is true and errors out when subject is missing

`lazy_load` whether to delay (lazy) the evaluation `mount_data`

`.class` internally used, do not set, even if you know what this is

Method `export_matlab()`**:** Export the repository to 'Matlab' for future analysis

Usage:

```
RAVESubjectRecordingBlockRepository$export_matlab(..., verbose = TRUE)
```

Arguments:

`...` reserved for child classes

`verbose` print progresses

Method `get_container()`**:** get container where loaded data are stored

Usage:

```
RAVESubjectRecordingBlockRepository$get_container(force = FALSE, ...)
```

Arguments:

`force, ...` passed to `mount_data`

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

Usage:

```
RAVESubjectRecordingBlockRepository$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`prepare_subject_with_blocks`

`RAVESubjectRecordingBlockTimeFreqBaseRepository` *'RAVE' class for loading entire block of time-frequency coefficients*

Description

'RAVE' class for loading entire block of time-frequency coefficients

'RAVE' class for loading entire block of time-frequency coefficients

Super classes

`ravepipeline::RAVESerializable` -> `ravecore::RAVESubjectRepository` -> `ravecore::RAVESubjectRecordingBlo` -> `RAVESubjectRecordingBlockTimeFreqBaseRepository`

Active bindings

`sample_rates` a named list of sampling frequencies; the names are signal types (`'LFP'`, `'Auxiliary'`, or `'Spike'`) and the values are the sampling frequencies

`sample_rate` numeric sample rate for 'spectrogram'

Methods

Public methods:

- `RAVESubjectRecordingBlockTimeFreqBaseRepository$@marshal()`
- `RAVESubjectRecordingBlockTimeFreqBaseRepository$@unmarshal()`
- `RAVESubjectRecordingBlockTimeFreqBaseRepository$new()`
- `RAVESubjectRecordingBlockTimeFreqBaseRepository$mount_data()`
- `RAVESubjectRecordingBlockTimeFreqBaseRepository$clone()`

Method `@marshal()`**:** Internal method

Usage:

```
RAVESubjectRecordingBlockTimeFreqBaseRepository$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`: Internal method

Usage:

```
RAVESubjectRecordingBlockTimeFreqBaseRepository$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

Method `new()`: constructor

Usage:

```
RAVESubjectRecordingBlockTimeFreqBaseRepository$new(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  blocks = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE,
  lazy_load = FALSE,
  .class = NULL
)
```

Arguments:

`subject` 'RAVE' subject

`electrodes` string or integers indicating electrodes to load

`reference_name` name of the reference table

`blocks` name of the recording blocks to load

`...` passed to `RAVESubjectBaseRepository` constructor

`quiet` see field `quiet`

`repository_id` see field `repository_id`

`strict` whether the mode should be strict; default is true and errors out when subject is missing

`lazy_load` whether to delay (lazy) the evaluation `mount_data`

`.class` internally used, do not set, even if you know what this is

Method `mount_data()`: function to mount data

Usage:

```
RAVESubjectRecordingBlockTimeFreqBaseRepository$mount_data(
  ...,
  force = TRUE,
  electrodes = NULL
)
```

Arguments:

`...` reserved

`force` force update data; default is true; set to false to use cache

`electrodes` electrodes to update; default is `NULL` (all electrode channels)

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

Usage:

```
RAVESubjectRecordingBlockTimeFreqBaseRepository$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`prepare_subject_with_blocks`

`RAVESubjectRecordingBlockTimeFreqCoefRepository`
'RAVE' class for loading time-frequency coefficients

Description

Loads time-frequency coefficients (complex numbers)

Super classes

`ravepipeline::RAVESerializable` -> `ravecore::RAVESubjectRepository` -> `ravecore::RAVESubjectRecordingBlo`
-> `ravecore::RAVESubjectRecordingBlockTimeFreqBaseRepository` -> `RAVESubjectRecordingBlockTimeFreqCoefR`

Active bindings

`coefficients` data container, alias of `get_container`

Methods

Public methods:

- `RAVESubjectRecordingBlockTimeFreqCoefRepository$@marshal()`
- `RAVESubjectRecordingBlockTimeFreqCoefRepository$@unmarshal()`
- `RAVESubjectRecordingBlockTimeFreqCoefRepository$new()`
- `RAVESubjectRecordingBlockTimeFreqCoefRepository$clone()`

Method `@marshal()`**:** Internal method

Usage:

```
RAVESubjectRecordingBlockTimeFreqCoefRepository$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
RAVESubjectRecordingBlockTimeFreqCoefRepository$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

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

Usage:

```
RAVESubjectRecordingBlockTimeFreqCoefRepository$new(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  blocks = NULL,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE,
  lazy_load = FALSE,
  .class = NULL
)
```

Arguments:

`subject` 'RAVE' subject

`electrodes` string or integers indicating electrodes to load

`reference_name` name of the reference table

`blocks` name of the recording blocks to load

`...` passed to `RAVESubjectBaseRepository` constructor

`quiet` see field `quiet`

`repository_id` see field `repository_id`

`strict` whether the mode should be strict; default is true and errors out when subject is missing

`lazy_load` whether to delay (lazy) the evaluation `mount_data`

`.class` internally used, do not set, even if you know what this is

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

Usage:

```
RAVESubjectRecordingBlockTimeFreqCoefRepository$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`prepare_subject_time_frequency_coefficients_with_blocks`

RAVESubjectRecordingBlockVoltageRepository
'RAVE' class for blocks of voltage repository

Description

Compared to RAVESubjectBaseRepository, this repository loads the entire voltage traces for selected blocks; use prepare_subject_voltage_with_blocks to instantiate this repository.

Super classes

ravepipeline::RAVESerializable -> ravecore::RAVESubjectRepository -> ravecore::RAVESubjectRecordingBlo
-> RAVESubjectRecordingBlockVoltageRepository

Active bindings

`sample_rates` a named list of sampling frequencies; the names are signal types (`'LFP'`, `'Auxiliary'`, or `'Spike'`) and the values are the sampling frequencies

`voltage` data container, alias of `get_container`

Methods

Public methods:

- RAVESubjectRecordingBlockVoltageRepository$@marshal()
- RAVESubjectRecordingBlockVoltageRepository$@unmarshal()
- RAVESubjectRecordingBlockVoltageRepository$new()
- RAVESubjectRecordingBlockVoltageRepository$mount_data()
- RAVESubjectRecordingBlockVoltageRepository$clone()

Method `@marshal()`**:** Internal method

Usage:

```
RAVESubjectRecordingBlockVoltageRepository$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
RAVESubjectRecordingBlockVoltageRepository$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

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

Usage:

```
RAVESubjectRecordingBlockVoltageRepository$new(
  subject,
  electrodes = NULL,
  reference_name = NULL,
  blocks = NULL,
  downsample = NA,
  ...,
  quiet = FALSE,
  repository_id = NULL,
  strict = TRUE,
  lazy_load = FALSE,
  .class = NULL
)
```

Arguments:

`subject` 'RAVE' subject

`electrodes` string or integers indicating electrodes to load

`reference_name` name of the reference table

`blocks` name of the recording blocks to load

`downsample` down-sample rate by this integer number to save space and speed up computation; typically 'ERP' signals do not need super high sampling frequencies to load; default is `NA` and no down-sample is performed.

`...` passed to `RAVESubjectBaseRepository` constructor

`quiet` see field `quiet`

`repository_id` see field `repository_id`

`strict` whether the mode should be strict; default is true and errors out when subject is missing

`lazy_load` whether to delay (lazy) the evaluation `mount_data`

`.class` internally used, do not set, even if you know what this is

Method `mount_data()`**:** function to mount data

Usage:

```
RAVESubjectRecordingBlockVoltageRepository$mount_data(
  ...,
  force = TRUE,
  electrodes = NULL
)
```

Arguments:

`...` reserved

`force` force update data; default is true; set to false to use cache

`electrodes` electrodes to update; default is `NULL` (all electrode channels)

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

Usage:

```
RAVESubjectRecordingBlockVoltageRepository$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

prepare_subject_voltage_with_blocks

---

rave_brain *Load 'FreeSurfer' brain from 'RAVE'*

Description

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

Usage

```
rave_brain(
  subject,
  surfaces = "pial",
  overlays = "aparc.a2009s+aseg",
  annotations = "label/aparc.a2009s",
  ...,
  usetemplateifmissing = FALSE,
  include_electrodes = TRUE
)
```

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 threeBrain
overlays	volumes to overlay; default is 'aparc.a2009s+aseg'
annotations	surface annotation or curvature data to load; default is 'label/aparc.a2009s', referring to the '*h.aparc.a2009s.annot' under the label folder.
...	ignored, reserved for legacy code
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.
include_electrodes	whether to include electrode in the model; default is true

Value

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

Examples

```
if(has_rave_subject("demo/DemoSubject")) {

  brain <- rave_brain("demo/DemoSubject")

  if (interactive()) {

    brain$plot()

  }

}
```

---

rave_cmd_tools *Find external command-line tools*

Description

Find external command-line tools

Usage

```
normalize_commandline_path(
  path,
  type = c("dcm2niix", "freesurfer", "fsl", "afni", "others"),
  unset = NA
)

cmd_dcm2niix(error_on_missing = TRUE, unset = NA)

cmd_freesurfer_home(error_on_missing = TRUE, unset = NA)

cmd_fsl_home(error_on_missing = TRUE, unset = NA)

cmd_afni_home(error_on_missing = TRUE, unset = NA)

cmd_homebrew(error_on_missing = TRUE, unset = NA)

cmd_dry_run()

rscript_path(...)
```

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
...	ignored

Value

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

---

rave_legacy_subject_format_conversion

Legacy 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_legacy_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

rave_path *Find file paths based on storage*

Description

A generic function that will be dispatched to using different method based on input `x`

Usage

```
rave_path(x, storage = NULL, ...)
```

Arguments

`x`	R object
`storage`	storage type, different options based on different R objects
`...`	additional arguments passed to dispatched method

run_wavelet *Apply Morlet-Wavelet to subject*

Description

Calculates time-frequency decomposition; not intended for direct use. Please use 'RAVE' pipelines (see 'Examples').

Usage

```
run_wavelet(
  subject,
  electrodes,
  freqs,
  cycles,
  target_sample_rate = 100,
  kernels_precision = "float",
  pre_downsample = 1,
  verbose = TRUE
)
```

Arguments

`subject`	'RAVE' subject or subject ID
`electrodes`	electrode channels to apply, must be imported and `'LFP'` type
`freqs`	numeric vector of frequencies to apply
`cycles`	number of wavelet cycles at each `freqs`, integers

`target_sample_rate`
: the resulting 'spectrogram' sampling frequency

`kernels_precision`
: double or single (default) floating precision

`pre_downsample`
: down-sample (integer) priory to the decomposition; set to 1 (default) to avoid

`verbose`
: whether to verbose the progress

Details

The channel signals are first down-sampled (optional) by a ratio of `pre_downsample` via a 'FIR' filter. After the down-sample, 'Morlet' wavelet kernels are applied to the signals to calculate the wavelet coefficients (complex number) at each frequency in `freqs`. The number of `cycles` at each frequency controls the number of sine and cosine waves, allowing users to balance the time and power accuracy. After the decomposition, the 'spectrogram' is further down-sampled to `target_sample_rate`, assuming the brain power is a smooth function over time. This down-sample is done via time-point sampling to preserve the phase information (so the linear functions such as common-average or bi-polar reference can be carried over to the complex coefficients).

Value

The decomposition results are stored in 'RAVE' subject data path; the function only returns the wavelet parameters.

Examples

```
# Check https://rave.wiki for additional pipeline installation

## Not run:

# ---- Recommended usage ------------------------------------------


pipeline <- ravepipeline::pipeline("wavelet_module")
pipeline$set_settings(
  project_name = "demo",
  subject_code = "DemoSubject",
  precision = "float",
  pre_downsample = 4,
  kernel_table = ravetools::wavelet_cycles_suggest(
    freqs = seq(1, 200, by = 1)),
  target_sample_rate = 100
)

# Internally, the above pipeline includes this function call below

# ---- For demonstration use, do not call this function directly ----

# Original sample rate: 2000 Hz
# Downsample by 4 to 500 Hz first - 250 Hz Nyquist
# Wavelet at each 1, 2, ..., 200 Hz
#   The number of cycles log-linear from 2 to 20
```

```
#   The wavelet coefficient sample rate is 500 Hz
# Further down-sample to 100 Hz to save storage space

run_wavelet(
  subject = "demo/DemoSubject",
  electrodes = c(13:16, 2),
  pre_downsample = 4,
  freqs = seq(1, 200, by = 1),
  cycles = c(2, 20),
  target_sample_rate = 100
)


## End(Not run)
```

`snapshot_project` *Create overview report for given project or subject*

Description

Create overview report for given project or subject

Usage

```
snapshot_subject(x, target_path = NULL, quick = FALSE)

snapshot_project(
  x,
  target_path = NULL,
  template_subjects = NULL,
  quick = FALSE
)
```

Arguments

`x`	characters or a 'RAVE' project or subject instance
`target_path`	directory where the snapshot data will be stored; default is under `'project_overview'` group data folder, see method `group_data` from `RAVEProject` class.
`quick`	whether to skip certain validations and subjects if snapshot reports have already existed
`template_subjects`	a vector of characters of template brain to be used for generating the group brain; see `available_templates` for available templates.

Value

`snapshot_subject` returns a list of subject summary and the calculated target path; `snapshot_project` returns the path to the compiled project report in 'HTML'.

Examples

```
## Not run:

# Run in parallel
ravepipeline::with_rave_parallel({

  snapshot_project("demo")

})


## End(Not run)
```

Spike_electrode *Class definition for micro-wire spike channels*

Description

Class definition for micro-wire spike channels

Class definition for micro-wire spike channels

Value

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.

Super classes

`ravepipeline::RAVESerializable` -> `ravecore::RAVEAbstarctElectrode` -> `Spike_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

`voltage_file` path to voltage 'HDF5' file

Methods

Public methods:

- Spike_electrode$@marshal()
- Spike_electrode$@unmarshal()
- Spike_electrode$print()
- Spike_electrode$set_reference()
- Spike_electrode$new()
- Spike_electrode$.load_noref_voltage()
- Spike_electrode$.load_raw_voltage()
- Spike_electrode$load_data_with_epochs()
- Spike_electrode$load_dimnames_with_epochs()
- Spike_electrode$load_data_with_blocks()
- Spike_electrode$load_dim_with_blocks()
- Spike_electrode$clear_cache()
- Spike_electrode$clear_memory()
- Spike_electrode$clone()

Method `@marshal()`**:** Internal method

Usage:

```
Spike_electrode$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
Spike_electrode$@unmarshal(object)
```

Arguments:

`object`, `...` internal arguments

Method `print()`**:** print electrode summary

Usage:

```
Spike_electrode$print()
```

Method `set_reference()`**:** set reference for current electrode

Usage:

```
Spike_electrode$set_reference(reference)
```

Arguments:

`reference` either `NULL` or `LFP_electrode` instance

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

Usage:

```
Spike_electrode$new(subject, number, quiet = FALSE)
```

Arguments:

`subject`, `number`, `quiet` see constructor in `RAVEAbstarctElectrode`

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

Usage:

```
Spike_electrode$.load_noref_voltage(reload = FALSE)
```

Arguments:

`reload` whether to reload cache

`srate` voltage signal sample rate

Method `.load_raw_voltage()`: load raw voltage (no process)

Usage:

```
Spike_electrode$.load_raw_voltage(reload = FALSE)
```

Arguments:

`reload` whether to reload cache

Method `load_data_with_epochs()`: method to load electrode data

Usage:

```
Spike_electrode$load_data_with_epochs(type = c("raw-voltage", "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_dimnames_with_epochs()`: get expected dimension names

Usage:

```
Spike_electrode$load_dimnames_with_epochs(type = c("raw-voltage", "voltage"))
```

Arguments:

`type` see `load_data_with_epochs`

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

Usage:

```
Spike_electrode$load_data_with_blocks(
  blocks,
  type = c("raw-voltage", "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

Method `load_dim_with_blocks()`: get expected dimension information for block-based loader

Usage:

```
Spike_electrode$load_dim_with_blocks(
  blocks,
  type = c("raw-voltage", "voltage")
)
```

Arguments:

`blocks, type` see `load_data_with_blocks`

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

Usage:

```
Spike_electrode$clear_cache(...)
```

Arguments:

`...` ignored

Method `clear_memory()`: method to clear memory

Usage:

```
Spike_electrode$clear_memory(...)
```

Arguments:

`...` ignored

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

Usage:

```
Spike_electrode$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`transform_point_to_template` *Calculate template 'MNI' coordinates for points on native brain*

Description

Calculate template 'MNI' coordinates for points on native brain

Usage

```
transform_point_to_template(
  subject,
  positions,
  space = c("scannerRAS", "tkrRAS"),
  mapping_method = c("volumetric", "surface"),
  flip_hemisphere = FALSE,
  verbose = TRUE,
  project_surface = "pial",
  volumetric_transform = c("auto", "affine", "nonlinear"),
  ...
)

transform_thinfilm_to_mni152(
  subject,
  flip_hemisphere = FALSE,
  interpolator = 0.3,
  n_segments = c(16, 16),
  group_labels = NULL,
  project_surface = "pial",
  volumetric_transform = c("auto", "affine", "nonlinear"),
  template_subject = c("cvs_avg35_inMNI152", "fsaverage", "bert", "MNI152")
)
```

Arguments

`subject`	'RAVE' subject
`positions`	optional matrix of 3 columns, either in scanner or surface space (specified by `space`); default is missing and will use the electrode localization results (`electrodes.csv`)
`space`	if `positions` is given, which native coordinate system should be used; default is native 'T1' (or `'scannerRAS'`); alternative is 'FreeSurfer' surface coordinate (or `'tkrRAS'`)
`mapping_method`	whether the mapping is `'volumetric'` or `'surface'`; default is the former.
`flip_hemisphere`	whether to flip the hemisphere; default is `FALSE`
`verbose`	whether to verbose the mapping progress; default is true
`project_surface`	for surface mapping only, which surface to project electrodes onto; default is `'pial'` surface, other common choices are `'white'` for white-matter, or `'smoothwm'` for smoothed white matter
`volumetric_transform`	for volume mapping only, which type of transform to use; default is `'auto'` detecting and use non-linear deformation if exists, and fall back to 'affine' transform; other choices are `'affine'` or `'nonlinear'`
`...`	ignored

interpolator	whether the transform lean towards volume mapping (interpolator=0) or surface mapping (interpolator=1)
n_segments	positive integers with length of two: resolution of the mapping; default segments the thin-film array into 16 by 16 segments
group_labels	NULL (default) or a character vector indicating the group labels of thin-film electrodes; default assumes that all contacts are from thin-film electrodes.
template_subject	template subject to be mapped to; default is 'cvs_avg35_inMNI152', which is a 'MNI152' template generated by 'FreeSurfer'; other choices are 'fsaverage' and 'bert'

Value

A table of electrode 'MNI' coordinates.

Examples

```
if (has_rave_subject("demo/DemoSubject")) {

  transform_point_to_template(
    subject = 'demo/DemoSubject',
    mapping_method = "volumetric"
  )

}
```

---

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

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 legacy raw data folder

`raw_path` the subject's raw data folder based on format standard

`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

`power_data` same as `power_data` but for the phase data

Epoch table (`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)
```

YAELProcess *Class definition of 'YAEL' image pipeline*

Description

Rigid-registration across multiple types of images, non-linear normalization from native brain to common templates, and map template atlas or regions of interest back to native brain. See examples at as_yael_process

Value

whether the image has been set (or replaced)

Absolute path if the image

'RAVE' subject instance

Nothing

A list of moving and fixing images, with rigid transformations from different formats.

See method get_template_mapping

A list of input, output images, with forward and inverse transform files (usually two 'Affine' with one displacement field)

transformed image in 'ANTs' format

transformed image in 'ANTs' format

Paths to the atlas (volume) files

A matrix of 3 columns, each row is a transformed points (invalid rows will be filled with NA)

A matrix of 3 columns, each row is a transformed points (invalid rows will be filled with NA)

Super class

ravepipeline::RAVESerializable -> YAELProcess

Active bindings

`subject_code` 'RAVE' subject code

`image_types` allowed image types

`work_path` Working directory ('RAVE' imaging path)

Methods

Public methods:

Method `@marshal()`**:** Internal method

Usage:

```
YAELProcess$@marshal(...)
```

Arguments:

`...` internal arguments

Method `@unmarshal()`**:** Internal method

Usage:

```
YAELProcess$@unmarshal(object, ...)
```

Arguments:

`object, ...` internal arguments

Method `new()`**:** Constructor to instantiate the class

Usage:

```
YAELProcess$new(subject, image_types, imaging_path = NULL, ...)
```

Arguments:

`subject` 'RAVE' subject or subject ID; for native standard, this can be character code without project names, but for 'BIDS' subjects, this must be a full subject ID with project information

`image_types` vector of image types, such as `'T1w'`, `'CT'`, `'fGATIR'`. All images except `'CT'` will be considered `'preop'` (before electrode implantation). Please use `'postop'` to indicate if an image is taken after the implantation (for example, `'postopT1w'`)

`imaging_path` imaging path (path to `'rave-imaging'` if not default); internally used to set the work path during serialization. Please do not set it manually unless you know what you are doing

`...` reserved for legacy code

Method `set_input_image()`**:** Set the raw input for different image types

Usage:

```
YAELProcess$set_input_image(
  path,
  type = YAEL_IMAGE_TYPES,
  overwrite = FALSE,
  on_error = c("warning", "error", "ignore")
)
```

Arguments:

`path` path to the image files in `'NIfTI'` format

`type` type of the image

`overwrite` whether to overwrite existing images if the same type has been imported before; default is false

`on_error` when the file exists and `overwrite` is false, how should this error be reported; choices are `'warning'` (default), `'error'` (throw error and abort), or `'ignore'`.

Method `get_input_image()`**:** Get image path

Usage:

```
YAELProcess$get_input_image(type = YAEL_IMAGE_TYPES)
```

Arguments:

`type` type of the image

Method `get_subject()`**:** Get 'RAVE' subject instance

Usage:

```
YAELProcess$get_subject(..., strict = FALSE)
```

Arguments:

`...` ignored

`strict` passed to `as_rave_subject`

Method `register_to_T1w()`**:** Register other images to `'T1'` weighted `'MRI'`

Usage:

```
YAELProcess$register_to_T1w(image_type = "CT", reverse = FALSE, verbose = TRUE)
```

Arguments:

`image_type` type of the image to register, must be set via `process$set_input_image` first.

`reverse` whether to reverse the registration; default is false, meaning the fixed (reference) image is the `'T1'`. When setting to true, then the `'T1'` 'MRI' will become the moving image

`verbose` whether to print out the process; default is true

Method `get_native_mapping()`**:** Get the mapping configurations used by `register_to_T1w`

Usage:

```
YAELProcess$get_native_mapping(image_type = YAEL_IMAGE_TYPES, relative = FALSE)
```

Arguments:

`image_type` type of the image registered to 'T1' weighted 'MRI'

`relative` whether to use relative path (to the `work_path` field)

Method `map_to_template()`**:** Normalize native brain to `'MNI152'` template

Usage:

```
YAELProcess$map_to_template(
  template_name = rpyants_builtin_templates(),
  use_images = c("T1w", "T2w", "T1wContrast", "fGATIR", "preopCT"),
  native_type = "T1w",
  use_antspynet = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments:

`template_name` which template to use, choices are `'mni_icbm152_nlin_asym_09a'`, `'mni_icbm152_nlin_asym_09b'` `'mni_icbm152_nlin_asym_09c'`, and `'fsaverage'`.

`use_images` a vector of image types to use for normalization; default types are `'T1w'`, `'T2w'`, `'T1wContrast'`, `'fGATIR'`, and `'preopCT'`. To use all available images for normalization, use wildcard `"all"`

`native_type` which type of image should be used to map to template; default is `'T1w'`

`use_antspynet` whether to try `'antspynet'` if available; default is true, which uses `deep_atropos` instead of the conventional `atropos` to speed up and possibly with more accurate results.

`verbose` whether to print out the process; default is true

`...` additional tuning parameters passed to internal 'Python' code.

Method `get_template_mapping()`**:** Get configurations used for normalization

Usage:

```
YAELProcess$get_template_mapping(
  template_name = rpyants_builtin_templates(),
  native_type = "T1w",
  relative = FALSE
)
```

Arguments:

`template_name` which template is used

`native_type` which native image is mapped to template

`relative` whether the paths should be relative or absolute; default is false (absolute paths)

Method `transform_image_from_template()`**:** Apply transform from images (usually an atlas or 'ROI') on template to native space

Usage:

```
YAELProcess$transform_image_from_template(
  template_roi_path,
  template_name = rpyants_builtin_templates(),
  native_type = "T1w",
  interpolator = c("auto", "nearestNeighbor", "linear", "gaussian", "bSpline",
    "cosineWindowedSinc", "welchWindowedSinc", "hammingWindowedSinc",
    "lanczosWindowedSinc", "genericLabel"),
  verbose = TRUE
)
```

Arguments:

`template_roi_path` path to the template image file which will be transformed into individuals' image

`template_name` templates to use

`native_type` which type of native image to use for calculating the coordinates (default `'T1w'`)

`interpolator` how to interpolate the `'voxels'`; default is `"auto"`: `'linear'` for probabilistic map and `'nearestNeighbor'` otherwise.

`verbose` whether the print out the progress

Method `transform_image_to_template()`**:** Apply transform to images (usually an atlas or 'ROI') from native space to template

Usage:

```
YAELProcess$transform_image_to_template(
  native_roi_path,
  template_name = rpyants_builtin_templates(),
  native_type = "T1w",
  interpolator = c("auto", "nearestNeighbor", "linear", "gaussian", "bSpline",
    "cosineWindowedSinc", "welchWindowedSinc", "hammingWindowedSinc",
    "lanczosWindowedSinc", "genericLabel"),
  verbose = TRUE
)
```

Arguments:

`native_roi_path` path to the native image file that will be transformed into template

`template_name` templates to use

`native_type` which type of native image to use for calculating the coordinates (default `'T1w'`)

`interpolator` how to interpolate the `'voxels'`; default is `"auto"`: `'linear'` for probabilistic map and `'nearestNeighbor'` otherwise.

`verbose` whether the print out the progress

Method `generate_atlas_from_template()`**:** Generate atlas maps from template and morph to native brain

Usage:

```
YAELProcess$generate_atlas_from_template(
  template_name = rpyants_builtin_templates(),
  atlas_folder = NULL,
  surfaces = NA,
  verbose = TRUE,
  lambda = 0.2,
  degree = 2,
  threshold_lb = 0.5,
  threshold_ub = NA
)
```

Arguments:

`template_name` which template to use

`atlas_folder` path to the atlas folder (that contains the atlas files)

`surfaces` whether to generate surfaces (triangle mesh); default is `NA` (generate if not existed). Other choices are `TRUE` for always generating and overwriting surface files, or `FALSE` to disable this function. The generated surfaces will stay in native `'T1'` space.

`verbose` whether the print out the progress

`lambda`, `degree`, `threshold_lb`, `threshold_ub` passed to `volume_to_surf`

Method `transform_points_to_template()`**:** Transform points from native images to template

Usage:

```
YAELProcess$transform_points_to_template(
  native_ras,
  template_name = rpyants_builtin_templates(),
  native_type = "T1w",
  verbose = TRUE
)
```

Arguments:

`native_ras` matrix or data frame with 3 columns indicating points sitting on native images in right-anterior-superior (`'RAS'`) coordinate system.

`template_name` template to use for mapping

`native_type` native image type where the points sit on

`verbose` whether the print out the progress

Method `transform_points_from_template()`**:** Transform points from template images to native

Usage:

```
YAELProcess$transform_points_from_template(
  template_ras,
  template_name = rpyants_builtin_templates(),
  native_type = "T1w",
  verbose = TRUE
)
```

Arguments:

`template_ras` matrix or data frame with 3 columns indicating points sitting on template images in right-anterior-superior (`'RAS'`) coordinate system.

`template_name` template to use for mapping

`native_type` native image type where the points sit on

`verbose` whether the print out the progress

Method `construct_ants_folder_from_template()`**:** Create a reconstruction folder (as an alternative option) that is generated from template brain to facilitate the three-dimensional viewer. Please make sure method `map_to_template` is called before using this method (or the program will fail)

Usage:

```
YAELProcess$construct_ants_folder_from_template(
  template_name = rpyants_builtin_templates(),
  add_surfaces = TRUE
)
```

Arguments:

`template_name` template to use for mapping

`add_surfaces` whether to create surfaces that is morphed from template to local; default is `TRUE`. Please enable this option only if the cortical surfaces are not critical (for example, you are studying the deep brain structures). Always use `'FreeSurfer'` if cortical information is used.

Method `get_brain()`**:** Get three-dimensional brain model

Usage:

```
YAELProcess$get_brain(
  electrodes = TRUE,
  coord_sys = c("scannerRAS", "tkrRAS", "MNI152", "MNI305"),
  ...
)
```

Arguments:

`electrodes` whether to add electrodes to the viewers; can be logical, data frame, or a character (path to electrode table). When the value is `TRUE`, the electrode file under `project_name` will be loaded; when `electrodes` is a `data.frame`, or path to a `'csv'` file, then please specify `coord_sys` on what is the coordinate system used for columns `"x"`, `"y"`, and `"z"`.

`coord_sys` coordinate system if `electrodes` is a data frame with columns `"x"`, `"y"`, and `"z"`, available choices are `'scannerRAS'` (defined by 'T1' weighted native 'MRI' image), `'tkrRAS'` (`'FreeSurfer'` defined native 'TK-registered'), `'MNI152'` (template 'MNI' coordinate system averaged over 152 subjects; this is the common "'MNI' coordinate space" we often refer to), and `'MNI305'` (template 'MNI' coordinate system averaged over 305 subjects; this coordinate system used by templates such as `'fsaverage'`)

`...` passed to `threeBrain`

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

Usage:

```
YAELProcess$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Index