Package: rave (via r-universe)

June 15, 2024

Type Package

Title R Analysis and Visualization of 'ECOG/iEEG' Data

Version 1.0.3.3

Description R Analysis and Visualization of 'ECOG/iEEG' Data RAVE provides cross-platform analysis and visualizations for neuroscience 'ECoG/iEEG' data by ``just clicking buttons''. This package include: (a) pre-processing of 'ECoG/iEEG' raw data into 'HDF5' format (a cross-platform data format that can be used for R, Matlab, Python, Java and C/C++); (b) 'ECoG/iEEG' epoch and visualizations in real time; (c) Users can write customized analysis as R packages and easily insert into RAVE for interactive visualizations. (d) 3D viewers driven by 'WebGL'.

License GPL-3

Encoding UTF-8

Language en-US

URL https://rave.wiki, https://beauchamplab.github.io/rave/

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

RoxygenNote 7.2.3

Imports utils, stats, graphics, grDevices, htmltools (>= 0.3.6), devtools, grid, signal (>= 0.7-6), dipsaus (>= 0.1.9), R6 (>= 2.2.2), future (>= 1.10.0), shiny (>= 1.0.5), stringr (>= 1.2.0), shinydashboard (>= 0.6.0), fftwtools (>= 0.9-7), digest (>= 0.6.13), shinyjs (>= 1.0), rlang (>= 0.1.6.9000), DT (>= 0.4), startup (>= 0.12.0), threeBrain (>= 0.2.1), shinyFiles (>= 0.7.3), shinyWidgets (>= 0.5.3), raveio (>= 0.0.5), ravetools (>= 0.0.3), ravedash

Suggests knitr, data.table (>= 1.11.8), servr (>= 0.13), reticulate (>= 1.6), rmarkdown, spelling, roxygen2, testthat, lobstr, rpymat, filearray, rstudioapi

Repository https://dipterix.r-universe.dev

RemoteUrl https://github.com/beauchamplab/rave

RemoteRef HEAD

RemoteSha 7aeb4d92369dfff6532a4fabb9a3416d22e89b30

Contents

`any_subject_loaded` *Function to check if data repository has data*

Description

Function to check if data repository has data

Usage

```
any_subject_loaded(rave_data = getDefaultDataRepository())
```

Arguments

`rave_data`	internally used

`archive_subject` *Archive Subject into Zipped file*

Description

Save subject data, including brain imaging files into a zipped file. Notice this function does not guarantee every file is in. Please always double check what's inside.

Usage

```
archive_subject(
  project_name,
  subject_code,
  include_cache = FALSE,
  include_fs = TRUE,
  include_raw = FALSE,
  save_to = tempdir()
)
```

Arguments

project_name	project name
subject_code	subject code
include_cache	whether to include cache for faster loading. Default is false
include_fs	whether to include 'FreeSurfer' and 'AFNI/SUMA' files
include_raw	whether to include raw data
save_to	directory to save file to

---

arrange_data_dir *Initialize data repository*

Description

Initialize data repository

Usage

```
arrange_data_dir(first_time = FALSE, reset = FALSE)
```

Arguments

first_time	will create data repositories for you
reset	reset to default data directory

---

arrange_modules *Update (optional), and check validity of modules*

Description

Update (optional), and check validity of modules

Usage

```
arrange_modules(refresh = FALSE, reset = FALSE, quiet = FALSE)
```

Arguments

refresh	check and updates file
reset	same as first_time, check module updates, ignored
quiet	no overwrite messages

as_subject *Make new subject object from character*

Description

Make new subject object from character

Usage

```
as_subject(subject, strict = TRUE, reference = "default")
```

Arguments

subject	characters in format "project/subject"
strict	logical indication whether preprocess folder is needed
reference	what reference file the subject is using

attachDefaultDataRepository *Attach subject data*

Description

Attach subject data

Usage

```
attachDefaultDataRepository(
  unload = TRUE,
  rave_data = getDefaultDataRepository()
)
```

Arguments

unload	TRUE if you want to detach
rave_data	internally used

`baseline` *Baseline signals*

Description

Baseline signals

Usage

```
baseline(
  el,
  from,
  to,
  method = "mean",
  unit = "%",
  data_only = FALSE,
  hybrid = TRUE,
  swap_file = tempfile(),
  mem_optimize = TRUE,
  same_dimension = unit %in% c("%", "dB"),
  preop = NULL,
  op,
  data_env = getDefaultDataRepository()
)
```

Arguments

`el`	`Tensor` or `ECoGTensor` object
`from`	baseline start time
`to`	baseline end time
`method`	mean or median, default is mean
`unit`	"%" percent signal change or "dB" decibel unit
`data_only`	return array or tensor object?
`hybrid`	if return tensor object, swap cache? useful for large dataset
`swap_file`	by default `tempfile()`, or you can specify path
`mem_optimize`	optimize for large dataset? default is TRUE
`same_dimension`	logical, true if op is element-wise operator
`preop`	function before baseline
`op`	function for baseline
`data_env`	internally used

cache_raw_voltage *Import Subject "Matlab" File and Create "HDF5" files*

Description

Import Subject "Matlab" File and Create "HDF5" files

Usage

```
cache_raw_voltage(project_name, subject_code, blocks, electrodes, ...)
```

Arguments

project_name	project name
subject_code	subject code
blocks	blocks to be imported
electrodes	to be imported (please check file existence before calling this function)
...	ignored

check_data_repo *Check if default data environment has object*

Description

Check if default data environment has object

Usage

```
check_data_repo(
  var = c("subject"),
  any = FALSE,
  data_repo = getDefaultDataRepository()
)
```

Arguments

var	variable name
any	whether all variables should be present of any variables should exist
data_repo	internally used

Value

Logical TRUE or FALSE indicating the existence of the variables

check_dependencies *Check and Install RAVE Dependencies*

Description

Check and Install RAVE Dependencies

Usage

```
check_dependencies(
  update_rave = TRUE,
  restart = TRUE,
  nightly = FALSE,
  demo_data = FALSE,
  ...
)
```

Arguments

update_rave	logical, whether to update RAVE
restart	logical, whether to restart 'RStudio' after installation
nightly	logical, whether to install develop version
demo_data	logical, whether to check demo data
...	for compatibility purpose, ignored

check_epoch *Check if epoch file is valid*

Description

Check if epoch file is valid

Usage

```
check_epoch(subject, epoch_name)
```

Arguments

subject	subject object or string
epoch_name	epoch name to check

check_subject *Complete validity check a RAVE subject*

Description

Complete validity check a RAVE subject

Usage

```
check_subject(subject, stop_on_error = FALSE)
```

Arguments

subject	character, must be "project/subject" format
stop_on_error	logical, whether stop when error occurs

Value

None

check_subjects2 *check subject validity tools*

Description

check subject validity tools

Usage

```
check_subjects2(project_name, subject_code, quiet = FALSE)
```

Arguments

project_name	project_name
subject_code	subject_code
quiet	logical

check_subjects_old *check subject validity tools (use check_subjects2)*

Description

check subject validity tools (use check_subjects2)

Usage

```
check_subjects_old(
  project_name,
  subject_code,
  check = TRUE,
 folders = c("Subject Folder", "RAVE Folder", "Preprocessing Folder", "Meta Folder",
    "Channel Folder"),
  preprocess = c("Started Preprocess", "Notch Filter", "Wavelet"),
  Meta = c("Electrode File", "Time point File", "Frequency File", "Epoch File")
)
```

Arguments

project_name	project_name
subject_code	subject_code
check	check is internally used
folders	folders to check
preprocess	preprocess to check
Meta	Meta to check

create_local_cache *Create local cache to speed up loading speed*

Description

Create local cache to speed up loading speed

Usage

```
create_local_cache(project_name, subject_code, epoch, time_range)
```

Arguments

project_name	project name
subject_code	subject code
epoch	epoch name
time_range	time range to cache

customizedUI *Customized Shiny Elements*

Description

Customized Shiny Elements

Usage

```
customizedUI(inputId, width = 12L, ...)
```

Arguments

inputId	character, input id
width	integer from 1-12
...	passed to uiOutput

decimate_fir *Decimate or Down-sample a Signal using* FIR *Filters*

Description

Decimate or Down-sample a Signal using FIR Filters

Usage

```
decimate_fir(x, q, n = 30)
```

Arguments

x	signal to be decimated
q	integer factor to decimated by
n	filter order used for down-sample procedure, default is 30

Value

Down-sampled signal

define_initialization *Defines 'RAVE' Module Initialization Defines the global variables for the module. Called along with* define_input *to define UI initialization actions once a subject is loaded.*

Description

Defines 'RAVE' Module Initialization Defines the global variables for the module. Called along with define_input to define UI initialization actions once a subject is loaded.

Usage

```
define_initialization(expr)

## Default S3 method:
define_initialization(expr)

## S3 method for class 'rave_module_debug'
define_initialization(expr)

## S3 method for class 'rave_running'
define_initialization(expr)

## S3 method for class 'rave_running_local'
define_initialization(expr)
```

Arguments

expr	R expression to run after subject is loaded

Value

None

Examples

```
## Not run:
# Requires to install R package beauchamplab/ravebuiltins

# Enable debug mode
ravebuiltins::dev_ravebuiltins(reload = FALSE)
# Check data
define_initialization({
  rave_checks('power')
  power <- module_tools$get_power(referenced = TRUE)
})
```

```
# Initialize global variables for modules
ravebuiltins::dev_ravebuiltins(reload = FALSE)
define_initialization({
  print(subject$info())
  time_points = preload_info$time_points
})
define_input(
  shiny::sliderInput('time_range', 'Time-Range', min=0,
                     max=1, value = c(0,1)),
  init_args = c('min', 'max', 'value'),
  init_expr = {
    min = min(time_points)
    max = max(time_points)
    value = c(0, max)
  }
)

## End(Not run)
```

`define_input` *Defines 'RAVE' Module Inputs*

Description

Defines 'RAVE' Module Inputs

Usage

```
define_input(
  definition,
  init_args,
  init_expr,
  keyword = "inputId",
  update_level = 2,
  ...
)

## Default S3 method:
define_input(
  definition,
  init_args,
  init_expr,
  keyword = "inputId",
  update_level = 2,
  ...
)

## S3 method for class 'rave_module_debug'
```

```
define_input(definition, init_args, init_expr, ...)

## S3 method for class 'rave_running_local'
define_input(
  definition,
  init_args,
  init_expr,
  keyword = "inputId",
  update_level = 2,
  ...
)

## S3 method for class 'rave_running'
define_input(
  definition,
  init_args,
  init_expr,
  keyword = "inputId",
  update_level = 2,
  ...
)
```

Arguments

`definition`	R expression to define UI elements without `ns()`, for example, `textInput('varname', 'Label', ...)`
`init_args`	arguments to change once a subject is loaded
`init_expr`	expression to evaluate with subject loaded
`keyword`	what identifies the input element
`update_level`	update action code: see details.
`...`	ignored or passed to other methods.

Details

This function behaves differently in different contexts. By default, it returns the result of `definition`. When debugging modules (`"rave_module_debug"`), it assigns a variable to the global environment with the variable name defined as input ID. In other contexts it parse the definition and returns a list for 'RAVE' to use internally to compile the module.

If `update_level` is '0' then the input is defined as manual inputs, which will not trigger re-calculate if changed. If '1' is set, then the input is a render's input, and only update render functions. If '2' is used, then once user change an input, then the whole module is re-calculated.

`init_args` must be argument names of the definition. Once subject is loaded, `init_expr` will be evaluated in a local environment, then variables in `init_args` will be used to update the input widgets.

Value

See details

`define_output` *Define 'RAVE' Module Output*

Description

Define 'RAVE' Module Output

Usage

```
define_output(
  definition,
  title = "",
  width = 12L,
  order = Inf,
  keyword = "outputId",
  ...
)

## Default S3 method:
define_output(
  definition,
  title = "",
  width = 12L,
  order = Inf,
  keyword = "outputId",
  ...
)
```

Arguments

`definition`	R expression of output, such as `plotOutput('out')`
`title`	Title to show
`width`	integer from 1 to 12, similar to "width" in `column`
`order`	the order of output, smaller order will be displayed first
`keyword`	keyword for the output ID
`...`	ignored or passed to other methods

Value

In default or debug context, it returns HTML tags, but when 'RAVE' is running, the result will be parse list for internal use.

detect_modules *Check all packages to for new RAVE module packages*

Description

Check all packages to for new RAVE module packages

Usage

```
detect_modules(packages, as_module = TRUE, ...)
```

Arguments

packages	array of packages to search for, default is all packages
as_module	logical, try to return module instances or just a list of modules
...	ignored for compatibility purpose

diagnose_signal *Plot and Inspect Signals in Trace, Periodogram, and Histogram*

Description

Plot and Inspect Signals in Trace, Periodogram, and Histogram

Usage

```
diagnose_signal(
  s1,
  s2 = NULL,
  sc = NULL,
  srate,
  name = "",
  try_compress = TRUE,
  max_freq = 300,
  window = ceiling(srate * 2),
  noverlap = window/2,
  std = 3,
  cex = 1.5,
  lwd = 0.5,
  flim = NULL,
  nclass = 100,
  main = "Channel Inspection",
  col = c("black", "red"),
  which = NULL,
  start_time = 0,
```

```
  boundary = NULL,
  mar = c(5.2, 5.1, 4.1, 2.1),
  ...
)
```

Arguments

s1	Signal for inspection
s2	Signal to compare, default NULL
sc	compressed signal to speedup the trace plot, if not provided, then either the original s1 is used, or a compressed version will be used. See parameter try_compress.
srate	Sample rate of s1, note that s2 and s1 must have the same sample rate
name	Analysis name, for e.g. "CAR", "Notch", etc.
try_compress	If length of s1 is too large, it might take long to draw trace plot, my solution is to down-sample s1 first (like what Matlab does), and then plot the compressed signal. Some information will be lost during this process, however, the trade-off is the speed. try_compress=FALSE indicates that you don't want to compress signals under any situation (this might be slow).
max_freq	Max frequency to plot, should be no larger than half of the sampling rate.
window	Window length to draw the Periodogram
noverlap	Number of data points that each adjacent windows overlap
std	Error bar (red line) be drawn at standard deviations, by default is 3, meaning the error bars represent 3 standard deviations.
cex, lwd, mar, ...	passed to plot.default
flim	log10 of frequency range to plot
nclass	Number of classes for histogram
main	Plot title
col	Color for two signals, length of 2.
which	Which sub-plot to plot
start_time	When does signal starts
boundary	Boundary for signal plot, default is 1 standard deviation

Examples

```
library(stats)
time <- seq(0, 100, by = 1/200)
s2 <- sin(2 * pi * 60 * time) + rnorm(length(time))
diagnose_signal(s2, srate = 200)

# Apply notch filter
s1 = notch_filter(s2, 200, 58,62)
diagnose_signal(s1, s2, srate = 200)
```

download_sample_data *Function to download demo data to data repository*

Description

Function to download demo data to data repository

Usage

```
download_sample_data(subject, version = "v0.1.8-beta", ...)
```

Arguments

subject	demo subject
version	rave release version
...	other parameters passed to download_subject_data

Value

Nothing

download_subject_data *Function to download subjects from internet/local*

Description

Function to download subjects from internet/local

Usage

```
download_subject_data(
  con,
  replace_if_exists = FALSE,
  override_project = NULL,
  override_subject = NULL,
  temp_dir = tempdir(),
  remove_zipfile = TRUE,
  subject_settings = NULL,
  mode = "wb",
  ...
)
```

Arguments

con	an url or local file path
replace_if_exists	Automatically replace current subject if subject files exist (default FALSE)
override_project	if not null, project will be renamed to this value
override_subject	if not null, subject will be renamed to this value
temp_dir	temp directory to store downloaded zip files and extracted files
remove_zipfile	clear downloaded zip files? if con is local file, this will be forced to FALSE
subject_settings	override "subject.yaml" see details
mode, ...	passed to download.file

Details

Each downloaded zip file should have a "subject.yaml" file indicating default project name, subject code, data directory and raw data directory.

If you want to override subject settings, you need to implement your own subject_settings. See examples.

Examples

```
## Not run:
# Normal example
download.file(
  'https://s3-us-west-2.amazonaws.com/rave-demo-subject/sfn-demo/data-large.zip',
  destfile = "~/rave_data/data-small.zip", mode = "wb")
download_subject_data(con = "~/rave_data/data-small.zip")

# or the following
# download_subject_data(
# 'https://s3-us-west-2.amazonaws.com/rave-demo-subject/sfn-demo/data-large.zip'
# )

# rename project to demo_junk
download_subject_data(con = "~/rave_data/data-small.zip",
           override_project = 'demo_junk')

# override settings
download_subject_data(
  con = "~/rave_data/data-small.zip",
  subject_settings = list(
    # subject conf
    'demo_project/demo_subject' = list(
      data_dir = 'data 2/data_dir/demo/sub1',
      raw_dir = 'data 2/raw_dir/sub1'
    )
```

```
  )
)

## End(Not run)
```

ECoGRepository *R6 class for* iEEG/ECoG *data Repository*

Description

A repository to keep subject information, including electrode instances, reference information, epoch data, and offers method to epoch data.

Public fields

`subject` Subject instance

`raw` dictionary to store Electrode instances

`reference` dictionary to store references for electrodes

`epochs` dictionary to store epoch data

`raw_volt` environment, stores pre-referenced analog traces

`raw_power` environment, stores pre-referenced power spectrum

`raw_phase` environment, stores pre-referenced phase data

`volt` environment, stores referenced analog traces

`power` environment, stores referenced power spectrum

`phase` environment, stores referenced phase data

Methods

Public methods:

- ECoGRepository$info()
- ECoGRepository$print()
- ECoGRepository$new()
- ECoGRepository$get_electrode()
- ECoGRepository$load_electrodes()
- ECoGRepository$epoch()
- ECoGRepository$load_reference()
- ECoGRepository$baseline()

Method `info()`**:** obtain the information

Usage:

```
ECoGRepository$info(print = TRUE)
```

Arguments:

`print` logical, whether to print the information, default is true

Returns: character of the information

Method `print()`**:** print memory address

Usage:

```
ECoGRepository$print(...)
```

Arguments:

`...` ignored

Returns: none

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

Usage:

```
ECoGRepository$new(subject, reference = "default", autoload = TRUE)
```

Arguments:

`subject` character such as `"project/subject"` or `Subject` instance

`reference` character, reference name, default is `"default"`, which refers to `"reference_default.csv"` in subject meta folder

`autoload` logical, whether to auto-load reference for all electrodes, default is yes.

Returns: An `ECoGRepository` instance

Method `get_electrode()`**:** get `Electrode` instances

Usage:

```
ECoGRepository$get_electrode(electrode, name = "raw")
```

Arguments:

`electrode` integers, referring to electrode numbers

`name` character, `"raw"`, `"power"`, `"raw_phase"`, etc.

Returns: list of environments containing electrode instances

Method `load_electrodes()`**:** load electrodes; usually don't need to directly call this method if `autoload` is true when initializing the repository

Usage:

```
ECoGRepository$load_electrodes(electrodes, reference = "default")
```

Arguments:

`electrodes` electrode number (integer)

`reference` name of reference

Returns: none

Method `epoch()`**:** slice the data according to epoch table

Usage:

```
ECoGRepository$epoch(
  epoch_name,
  pre,
  post,
  electrodes = NULL,
  frequency_range = NULL,
  data_type = "power",
  referenced = TRUE,
  func = NULL,
  quiet = FALSE
)
```

Arguments:

`epoch_name` the name of epoch; for example, `"YABa"` refers to `"epoch_YABa.csv"` in subject meta folder.

`pre` positive number in seconds, how long should the time be kept before the onset

`post` positive number in seconds, how long should the time be kept after onset

`electrodes` integers, electrode numbers

`frequency_range` experimental, frequency range to include

`data_type` data types to epoch; default is `"power"`, which is power spectrum, or amplitude. Other choices are `"phase"` for phase data and `"volt"` for voltage or analog signal traces.

`referenced` whether to load data referenced or without reference

`func` experimental, function to apply to each electrodes

`quiet` whether to suppress output messages, default is no

Returns: none. However the results are stored in public fields.

Method `load_reference()`**:** load references

Usage:

```
ECoGRepository$load_reference(ref_name, electrodes = NULL)
```

Arguments:

`ref_name` reference name

`electrodes` electrode numbers

Returns: none

Method `baseline()`**:** baseline signals (deprecated)

Usage:

```
ECoGRepository$baseline(from, to, electrodes = NULL, print.time = FALSE)
```

Arguments:

`from`, `to`, `electrodes`, `print.time` internally used

Returns: data after baseline. Please use `baseline` instead

Author(s)

Zhengjia Wang

Examples

```
## Not run:

# Two ways to create instances
repo <- ECoGRepository$new('demo/YAB')

subject <- Subject$new(project_name = 'demo', subject_code = 'YAB')
repo <- ECoGRepository$new(subject)

# Create an instance without auto collecting references, only load
# interesting electrodes
repo <- ECoGRepository$new('demo/YAB', autoload = FALSE)
repo$load_electrodes(c(14,15))

# Create an instance with non-default reference
repo <- ECoGRepository$new('demo/YAB', reference = 'bipolar')

# Epoch data according to epoch file "epoch_YABaOutlier.csv" in meta folder
# epoch_name should be "epoch_(name).csv"
repo$epoch(epoch_name = 'YABaOutlier', pre = 1, post = 2,
           electrodes = 14, referenced = TRUE, data_type = "power")
repo$power
#> Dimension:  287 x 16 x 301 x 1
#> - Trial: 1, 2, 3, 4, 5, 6,...
#> - Frequency: 2, 12, 22, 32, 42...
#> - Time: -1, -0.99, -0.98,...
#> - Electrode: 14


## End(Not run)
```

`Electrode` *R6 Class for Electrode*

Description

Stores single electrode or reference signals

Public fields

`electrode` electrode number in integer

`raw_power` stores pre-epoch power spectrum with no reference

`raw_phase` stores pre-epoch phase with no reference

`raw_volt` stores pre-epoch analog traces with no reference

`phase` stores pre-epoch phase after reference

`power` stores pre-epoch power spectrum after reference

`volt` stores pre-epoch analog traces after reference

`preload` which of the three data are pre-loaded

`reference` character or `Electrode` instance indicating the reference for current electrode

`has_power`, `has_phase`, `has_volt` whether power, phase, or voltage data exist in file system before and after reference

Active bindings

`has_power`, `has_phase`, `has_volt` whether power, phase, or voltage data exist in file system before and after reference

`blocks` character vector of block names (read-only)

`subject_id` character of subject ID (read-only)

`reference_electrode` whether this is a reference (read-only)

Methods

Public methods:

- `Electrode$info()`
- `Electrode$print()`
- `Electrode$switch_reference()`
- `Electrode$referenced()`
- `Electrode$clean()`
- `Electrode$new()`
- `Electrode$epoch()`

Method `info()`**:** print electrode information

Usage:

```
Electrode$info()
```

Returns: none

Method `print()`**:** overrides default print method

Usage:

```
Electrode$print(...)
```

Arguments:

`...` ignored

Returns: none

Method `switch_reference()`**:** switch reference (experimental)

Usage:

```
Electrode$switch_reference(new_reference)
```

Arguments:

`new_reference` An electrode instance

Returns: none

Method `referenced()`: get referenced data

Usage:

```
Electrode$referenced(type = "power", ram = TRUE)
```

Arguments:

`type` which data to reference, default is power

`ram` whether to load data to memory

Returns: If `ram` is true, then returns a list of matrices. The length of the list equals the number of blocks, and each matrix is frequency by time points. If `ram` is false, then returns an environment with each element a `LazyH5` or `LazyFST` instance.

Method `clean()`: remove data from memory

Usage:

```
Electrode$clean(types = c("power", "phase", "volt"), force = FALSE)
```

Arguments:

`types` data types to clean

`force` whether to remove pre-loaded data types

Returns: none

Method `new()`: constructor

Usage:

```
Electrode$new(
  subject,
  electrode,
  reference_by = "noref",
  preload = NULL,
  is_reference = FALSE
)
```

Arguments:

`subject` `Subject` instance or characters like `"proj/sub"`

`electrode` number, integer

`reference_by` reference signals, choices are character, or `Electrode` instance; default is `"noref"`, meaning no reference to the electrode

`preload` data to load along with constructor

`is_reference` is current instance a reference?

Returns: An `Electrode` instance

Method `epoch()`: epoch electrode

Usage:

```
Electrode$epoch(
  epoch_name,
  pre,
  post,
  types = c("volt", "power", "phase"),
  raw = FALSE,
  hybrid = TRUE
)
```

Arguments:

`epoch_name` epoch name, for example, `epoch_name="default"` refers to epoch file `"epoch_default.csv"` in subject meta folder

`pre` seconds before trial onset to load

`post` seconds after trial onset to load

`types` characters, data types to load; choices are `"volt"`, `"power"`, and `"phase"`

`raw` whether epoch pre-referenced data?

`hybrid` whether to fast-cache the data on hard-drive? See also `Tensor`

Returns: list of data after epoch

Author(s)

Zhengjia Wang

Examples

```
## Not run:
# Electrode with no reference
e1 <- Electrode$new('demo/YAB', electrode = 14, reference_by = 'noref')
e1$reference
#> Subject: demo/YAB
#> Electrode: noref (Reference)

# Add Common Average Reference in rave/data/reference/ref_13-63,65-84.h5
e2 <- Electrode$new('demo/YAB', electrode = 14,
                    reference_by = 'ref_13-63,65-84')

# Electrode with bipolar reference by another electrode
e3 <- Electrode$new('demo/YAB', electrode = 14, reference_by = 'ref_15')

# Alternative way
reference <- Electrode$new('demo/YAB', electrode = 15, is_reference = TRUE)
e4 <- Electrode$new('demo/YAB', electrode = 14, reference_by = reference)

# e3, e4 are the same in scientific meaning. To test it, epoch them
power3 <- e3$epoch('YABaOutlier', 1, 2, 'power',
                   raw = FALSE, hybrid = FALSE)$power
power4 <- e4$epoch('YABaOutlier', 1, 2, 'power',
                   raw = FALSE, hybrid = TRUE)$power

# Compare e3 and e4, result difference should be 0
range(power3$get_data() - power4$get_data())
#> 0

# With or without hybrid, the size will be different
# No hybrid, totally in memory
lobstr::obj_size(power3)
#> 12 MB
# Hybrid, data is swapped to hard-drive
lobstr::obj_size(power4)
#> 908 kB
```

```
## End(Not run)
```

`electrode_localization` *Electrode localization*

Description

Electrode localization

Usage

```
electrode_localization(subject_code, freesurfer_path, ct_path, ...)
```

Arguments

`subject_code`	'RAVE' subject code
`freesurfer_path`	'FreeSurfer' folder path that points to reconstructed subject brain
`ct_path`	'CT' path (in 'Nifti' format). The 'CT' has to be aligned to 'T1-MRI'. Please check this tutorial.
`...`	other parameters passing to `localization_module`

Value

This function will launch a shiny application.

`eval_when_ready` *Add Function to run once Module is Ready*

Description

Usually contains reactive functions that requires shiny reactive context

Usage

```
eval_when_ready(FUN)
```

Arguments

`FUN`	function that takes an environment (runtime environment) as arguments.

ExecEnvir *Session-based Module Runtime Environment Class*

Description

where all the module functions are executed. It's rarely created manually, use `get_module` to create module, run with `start_app(m, test.mode=TRUE)`, and then inspect modules.

Public fields

`.__rave_context__.` context string for current instance, indicating whether the module is running locally (public, but internally used)

`.__rave_package__.` current package name to run (public, but internally used)

`.__rave_module__.` module ID (public, but internally used)

`.__rave_module_instance__.` self instance (public, but internally used)

`module_env` `ModuleEnvir` instance

`cache_env` cache environment to store key-value pairs locally

`parent_env` the parent/top environment of the module, usually global environment or some name-space if the module is implemented as an R package

`wrapper_env` stores all the utility functions. Some functions are overridden there such as `observe`, `rave_checks`, or `eval_when_ready`. These functions behave differently inside or outside of shiny context, and with or without data loaded. The environment will be locked once the module is initialized. The parent environment is `parent_env`

`static_env` stores module static functions. These functions are evaluated under `parse_env` and then moved here. The environment is locked after initialization. Its parent environment is `wrapper_env`

`param_env` stores parameters and most of the user inputs. It can also serve as a repository for global variables. Unlike the previous environments, `param_env` is unlocked, but module creators do not have access to this environment directly. The parent environment is `static_env`

`runtime_env` where the main part of module is running. All shiny `observe` and `observeEvent` are redirected to this environment by default (unless using `shiny::observe`). All functions in `static_env` have access to this environment. The parent environment is `param_env`

`async_env` where asynchronous codes run

`parse_env` environment where modules are parsed. The parent environment is `runtime_env`. Once all functions are evaluated, this environment is not used. However, module creators don't directly access this environment once the module is initialized.

`ns` shiny name-space functions, is equivalent to `shiny::NS(module_id)`. The goal is to add prefixes to module inputs so that two modules with the same input ID are named differently

`auto_execute` (Deprecated) whether to auto-calculate results

`manual_inputIds` character vector; name list of manually input IDs. Used when the algorithm takes long to run

`rendering_inputIds` character vector; name list of input IDs that when one of the corresponding inputs is changed, then `rave_execute` will not get evaluated. Only the outputs are changed.

`input_update` expressions to update inputs

`register_output_events` expressions to register outputs

`register_input_events` expressions to register inputs

`execute` module main function. The function is dynamically generated. Don't call directly.

`async_module` (experimental) whether the module contains any asynchronous part

`global_reactives` shiny global `reactives`, internal use only

`local_reactives` shiny local `reactives`, internal use only

`internal_reactives` internal reactive values to control some elements, internal use only

`ready_functions` functions to run when the module is ready. The functions are called at the last step of `shinirize`. Usually it's used along with `eval_when_ready`, to make sure `global_reactives` and `local_reactives` getting registered before functions calls

Active bindings

`input_ids` vector of input IDs (read-only)

`input_labels` vector of input labels (read-only)

`output_labels` vector of output labels (read-only)

`output_ids` vector of output IDs (read-only)

Methods

Public methods:

- `ExecEnvir$reload()`
- `ExecEnvir$finalize()`
- `ExecEnvir$info()`
- `ExecEnvir$print()`
- `ExecEnvir$clean()`
- `ExecEnvir$new()`
- `ExecEnvir$reset()`
- `ExecEnvir$copy()`
- `ExecEnvir$execute_with()`
- `ExecEnvir$names()`
- `ExecEnvir$register_module()`
- `ExecEnvir$register_context()`
- `ExecEnvir$rave_inputs()`
- `ExecEnvir$rave_outputs()`
- `ExecEnvir$rave_updates()`
- `ExecEnvir$rave_execute()`
- `ExecEnvir$set_browser()`
- `ExecEnvir$generate_input_ui()`

- ExecEnvir$generate_output_ui()
- ExecEnvir$is_global()
- ExecEnvir$clone()

Method `reload()`: (experimental) signal the modules to reload

Usage:

```
ExecEnvir$reload()
```

Returns: none

Method `finalize()`: garbage collection

Usage:

```
ExecEnvir$finalize()
```

Returns: none

Method `info()`: print variables in different layers (environment)

Usage:

```
ExecEnvir$info()
```

Returns: none

Method `print()`: print the memory address

Usage:

```
ExecEnvir$print(...)
```

Arguments:

`...` ignored

Returns: memory address

Method `clean()`: clean the environments to release the resource

Usage:

```
ExecEnvir$clean()
```

Returns: none

Method `new()`: constructor

Usage:

```
ExecEnvir$new(session = getDefaultReactiveDomain(), parent_env = NULL)
```

Arguments:

`session` shiny session instance

`parent_env` parent environment of this instance: package name space or global environment

Method `reset()`: reset the runtime environment, rarely used

Usage:

```
ExecEnvir$reset(inputs)
```

Arguments:

`inputs` reactive value list

Returns: none

Method `copy()`: (deprecated) copy the instance locally

Usage:

```
ExecEnvir$copy(
  session_id = "__fake_runtime_env__",
  data_env = getDefaultDataRepository()
)
```

Arguments:

`session_id` character

`data_env` where the data is stored, default is the environment returned by `getDefaultDataRepository`

Returns: a copied instance

Method `execute_with()`: (deprecated) execute module with given parameter

Usage:

```
ExecEnvir$execute_with(param, async = FALSE, plan = NULL)
```

Arguments:

`param` named list

`async` whether to run the whole module

`plan` future plan

Returns: runtime environment

Method `names()`: returns names of a list, if names are null, returns blank characters

Usage:

```
ExecEnvir$names(x)
```

Arguments:

`x` a list

Returns: the names of the list

Method `register_module()`: register `ModuleEnvir` instance

Usage:

```
ExecEnvir$register_module(module_env)
```

Arguments:

`module_env` `ModuleEnvir` instance. The modules are shared across different sessions, but to run the module, we need to create runtime environment, which is `ExecEnvir`

Returns: none

Method `register_context()`: Register 'RAVE' context for current environment (internally used)

Usage:

```
ExecEnvir$register_context(context = c("rave_running", "rave_running_local"))
```

Arguments:

`context` context string to indicate whether the module is running locally

Returns: None

Method `rave_inputs()`**:** parse input components

Usage:

```
ExecEnvir$rave_inputs(
  ...,
  .input_panels = list(),
  .tabsets = list(),
  .env = NULL,
  .manual_inputs = NULL,
  .render_inputs = NULL
)
```

Arguments:

`...` shiny input calls, such as `textInput('id', 'Name', ...)`

`.input_panels, .tabsets` together define the input layouts

`.env` ignored, debug only

`.manual_inputs` input IDs that won't cause module re-calculate when inputs are updated

`.render_inputs` input IDs that only trigger render functions when updated

Returns: none

Method `rave_outputs()`**:** parse output components

Usage:

```
ExecEnvir$rave_outputs(
  ...,
  .output_tabsets = list(),
  .tabsets = list(),
  .env = NULL
)
```

Arguments:

`...` shiny output calls, such as `plotOutput('id', 'Title')`

`.output_tabsets, .tabsets` together define the output layouts

`.env` debug use

Returns: none

Method `rave_updates()`**:** input initialization when `iEEG/ECoG` data are imported

Usage:

```
ExecEnvir$rave_updates(..., .env = NULL)
```

Arguments:

`...` R expressions

`.env` for debug use

Method `rave_execute()`**:** parse, and compile to main function

Usage:

```
ExecEnvir$rave_execute(..., auto = TRUE, .env = NULL, async_vars = NULL)
```

Arguments:

`...` R expressions

`auto` whether the module should run automatically

`.env` debug use

`async_vars` variables further passed to `async` module

Returns: none, but `ExecEnvir$execute` will be generated.

Method `set_browser()`**:** (experimental) cache R expression in browser `localStorage`

Usage:

```
ExecEnvir$set_browser(expr, session = getDefaultReactiveDomain())
```

Arguments:

`expr` R expression

`session` shiny session instance

Method `generate_input_ui()`**:** generate input panels according to parsed `rave_inputs`

Usage:

```
ExecEnvir$generate_input_ui(sidebar_width = 3L)
```

Arguments:

`sidebar_width` integer from 1 to 11, the width of the input panels

Returns: HTML tags

Method `generate_output_ui()`**:** generate outputs labels according to parsed `rave_outputs`

Usage:

```
ExecEnvir$generate_output_ui(sidebar_width = 3L)
```

Arguments:

`sidebar_width` integer from 1 to 11, the width of the input panels, the output panel width is calculated as `12-sidebar_width`

Returns: HTML tags

Method `is_global()`**:** (deprecated) check if variable is shared across modules. Please use `cache_input` instead to get variable values.

Usage:

```
ExecEnvir$is_global(inputId)
```

Arguments:

`inputId` input ID

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Zhengjia Wang

Examples

```
## Not run:

# Load module
module <- get_module('ravebuiltins', 'power_explorer')

# Create execute environmen
execenv <- module$get_or_new_exec_env()
execenv$info()


## End(Not run)
```

---

export_diagnose_voltage

Export voltage (analog trace) diagnostic plots for each electrode

---

Description

You must import subject through `rave_preprocess()` and then run this function

Usage

```
export_diagnose_voltage(
  subject,
  electrodes,
  blocks,
  save_dir = "./export",
  width = 12,
  height = 7,
  useDingbats = FALSE,
  onefile = TRUE,
  winlen,
  freq_lim,
  nclass = 50,
  fore_col = "black",
  back_col = "grey80",
  ...
)
```

Arguments

`subject`	character with the following format: `"project_name/subject_code"`
`electrodes`	a integer vector. For example: `c(1,3,4,10:15)`
`blocks`	the blocks to include. Default is all blocks of this subject
`save_dir`	the directory you want to save the files
`width, height, useDingbats`	passed to `pdf`
`onefile`	collect images within one file?
`winlen`	window length, default is twice of the subject sampling rate
`freq_lim`	default is half of voltage sampling rate
`nclass`	number of classes in histogram plot, default is 50
`fore_col`	Periodogram color for Notch filtered signals
`back_col`	Periodogram color for raw signals
`...`	All other parameters passed to `pdf`

See Also

`pdf`, `diagnose_signal`, `pwelch`

`fake_session` *Fake 'shiny' Session for Debug Purpose*

Description

Fake 'shiny' Session for Debug Purpose

Usage

```
fake_session(rave_id = "__fake_session__", id = NULL)
```

Arguments

`rave_id`	internally used
`id`	module ID, used to create scope, will passed to `NS`

Value

Fake shiny session for debug purpose

finalize_installation *Finalize installation*

Description

download demo data

Usage

```
finalize_installation(
  packages,
  upgrade = c("ask", "config-only", "always", "never", "data-only"),
  async = FALSE
)
```

Arguments

packages	package name to finalize. 'rave' to only update base demo data, or `c('threeBrain', 'ravebuiltins')` to upgrade built-in data, or leave it blank to upgrade all.
upgrade	whether to ask. Default is `'always'` to receive default settings. Other choices are `'ask'`, `'never'`, `'config-only'`, and `'data-only'`
async	whether to run scripts in parallel; default is true.

getDefaultDataRepository *Get environment where subject data is loaded*

Description

Get environment where subject data is loaded

Usage

```
getDefaultDataRepository(
  session = getDefaultReactiveDomain(),
  session_id,
  session_based = NULL
)
```

Arguments

session	shiny session, default is NULL
session_id	internal use
session_based	internal use

`getExecEnvirFromContext`
Get Module Runtime Environment from Current Context

Description

Get Module Runtime Environment from Current Context

Usage

```
getExecEnvirFromContext()
```

Value

An `ExecEnvir` instance

`getModuleEnvirFromContext`
Get Module Instance from Current Context

Description

Get Module Instance from Current Context

Usage

```
getModuleEnvirFromContext()
```

Value

An `ModuleEnvir` instance

get_content *Parse 'RAVE' Module Contents*

Description

Parse 'RAVE' Module Contents

Usage

```
get_content(content, env, evaluate = TRUE, chunks = FALSE)
```

Arguments

content	characters, R code to parse into expressions
env	environment to parse the code into, please specify the context
evaluate	whether to evaluate parse expression into env
chunks	whether to respect code notations and chunk the code into separate parts

Details

If "evaluate=TRUE", then the parse code will be evaluated within env and returns a logical value: TRUE means the content has something, otherwise returns FALSE

If "evaluate=FALSE", returns the parsed expression and add attributes about the names of each chunk and whether they are asynchronous

Value

See details.

get_dir *Get Directories in 'RAVE'*

Description

Get Directories in 'RAVE'

Usage

```
get_dir(
  subject_code,
  project_name,
  block_num,
  mkdirs = NULL,
  subject_id,
  relative = FALSE
)
```

Arguments

subject_code	subject code; can be ignored when subject_id is provided
project_name	project name; can be ignored when subject_id is provided
block_num	block name (optional)
mkdirs	internally used
subject_id	subject ID; can be omitted if subject_code and project_name are provided
relative	whether to return relative path or absolute to root directory

---

get_fake_updated_message *internally used for debugging functions*

Description

internally used for debugging functions

Usage

```
get_fake_updated_message(..., .args = list(), .func = NULL)
```

Arguments

...	see with_fake_session
.args	same as ...
.func	function to pass to with_fake_session

---

get_mem_usage *Get RAM usage*

Description

Get RAM usage

Usage

```
get_mem_usage(
  modules = list(),
  data_envir = getDefaultDataRepository(),
  session = getDefaultReactiveDomain()
)
```

Arguments

modules	which module(s)
data_envir	default uses getDefaultDataRepository
session	shiny session instance

get_module *Function to find modules in packages*

Description

Function to find modules in packages

Usage

```
get_module(package, module_id, local = FALSE, ...)
```

Arguments

package	package name to search for modules
module_id	(optional) module ID if the package contains multiple modules
local	run module locally?
...	ignored for compatibility purpose

get_path *Safe Way to Access Module Package Files Using Relative Path*

Description

Safe Way to Access Module Package Files Using Relative Path

Usage

```
get_path(..., mustWork = FALSE, is_directory = FALSE)
```

Arguments

...	relative path to the file
mustWork	whether the file must exists
is_directory	whether required file is a directory

Value

If you are developing the package, `get_path` returns the absolute file path, otherwise it uses `system.file` to get the file from package library.

get_rave_theme *Get RAVE Theme from Package Settings*

Description

Get RAVE Theme from Package Settings

Arguments

packages	packages to check
type	characters, "continuous", "discrete", or both
theme	"light" or "dark"; default is current theme saved in rave_options('current_theme')
session	shiny session

Value

A list contains all palettes found in the packages.

Examples

```
pal = get_rave_theme('rave', type = c('continuous', 'discrete'), theme='light')
print(pal, plot=TRUE)

pal = get_rave_theme('rave', type = c('continuous', 'discrete'), theme='dark')
print(pal, plot=TRUE)
```

get_subjects *Get all subjects within project*

Description

Get all subjects within project

Usage

```
get_subjects(project_name, check_subfolders = TRUE, check_rawdata = FALSE)
```

Arguments

project_name	project
check_subfolders	logical, check whether folder ‘rave‘ exists in subject folder, default true
check_rawdata	logical, whether raw subject folder exists, default false

get_val *Get Value or Default*

Description

Get Value or Default

Usage

```
get_val(x, key = NULL, ..., .invalids = c("null", "na"))
```

Arguments

x	a list, or environment, or just any R object
key	the name to obtain from x. Default is NULL
...	if the value is invalid, the default value to return
.invalids	what counts as invalid? Default is NULL and NA, represented by "null" and "na"

import_electrodes *Import* .csv *files that contain electrode information*

Description

The table to import must contains a column 'Electrode' that is consistent with the corresponding subject.

Usage

```
import_electrodes(path, subject, use_fs = NA, ...)
```

Arguments

path	path to the electrode file to import
subject	'RAVE' project-subject combination
use_fs	whether to use 'FreeSurfer', default is to auto-detect
...	passed to read.csv

init_app *Initialize main application for debugging purpose*

Description

Initialize main application for debugging purpose

Usage

```
init_app(
  modules = NULL,
  active_module = NULL,
  launch.browser = TRUE,
  theme = "red",
  disable_sidebar = FALSE,
  simplify_header = FALSE,
  ...,
  data_repo = getDefaultDataRepository()
)
```

Arguments

modules	which modules to show. See load_modules
active_module	which module to focus at start up (use module ID)
launch.browser	launch browsers, default is on
theme	color theme for the website
disable_sidebar	hide sidebar at startup?
simplify_header	hide header at startup?
...	other parameters like test.mode for module debugging
data_repo	internally used

init_module *Initialize 'RAVE' module for debug purpose*

Description

Initialize 'RAVE' module for debug purpose

Usage

```
init_module(
  module_id,
  debug = FALSE,
  parse_context = c("rave_running", "rave_running_local")
)
```

Arguments

module_id	module ID
debug	whether to expose all functions to the global environment
parse_context	parsing context, for internal use

See Also

load_rave_module_package

---

lapply_async *lapply using future package in asynchronous way*

Description

lapply using future package in asynchronous way

Usage

```
lapply_async(
  x,
  fun,
  ...,
  .ncores = 0,
  .call_back = NULL,
  .packages = NULL,
  .envir = environment(),
  .globals = TRUE,
  .gc = TRUE,
  .as_datatable = FALSE,
  .nrows = 0
)

lapply_async3(
  x,
  fun,
  ...,
  .globals = TRUE,
  .gc = TRUE,
```

```
  .callback = NULL,
  .ncores = 0
)
```

Arguments

x, fun, ...	(See lapply)
.ncores	Number of cores to use. If the value is 0, the number of cores will be determined by rave_options('max_worker').
.call_back	A function takes current iteration number as argument, can be NULL.
.packages	NULL be default, then the function will detect attached packages automatically. Otherwise you have to specify the packages that you want to load.
.envir	internally used
.globals	Automatically detect variables. See ?future::future
.gc	Clean up environment after each iterations? Recommended for large datasets.
.as_datatable	logical, return result as data.frame. Experimental.
.nrows	integer, if .as_datatable=TRUE, number of rows expected.
.callback	function or NULL, callback function to monitor updates.

Examples

```
## Not run:
lapply_async(1:10, function(x){
  Sys.sleep(2) # Run for 1 secs
  Sys.getpid()
}, .ncores = 3, .call_back = function(i){
  cat('Running iteration -', i, '\n')
})

## End(Not run)
```

---

load_local_cache *Load local cache for fast importing voltage, power, and phase*

Description

Load local cache for fast importing voltage, power, and phase

Usage

```
load_local_cache(
  project_name,
  subject_code,
  epoch,
  time_range,
```

```
  frequency_range = NULL,
  electrodes,
  referenced = FALSE,
  data_type = "voltage"
)
```

Arguments

project_name	project name
subject_code	subject code
epoch	epoch name
time_range	time range to cache
frequency_range	frequency range to cache
electrodes	electrodes to cache
referenced	which reference to be used
data_type	which type(s) of data to cache

---

load_meta *Load subject meta data*

Description

Load subject meta data

Usage

```
load_meta(meta_type, project_name, subject_code, subject_id, meta_name)
```

Arguments

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

load_modules *Load RAVE Modules*

Description

Load RAVE Modules

Usage

```
load_modules(legacy = FALSE)
```

Arguments

legacy	for internal debug use

load_rave_module_package *Function to load RAVE module package with UI tools*

Description

called internally by `init_module` or other module packages

Usage

```
load_rave_module_package(
  env,
  parse_context = c("rave_module_debug", "rave_running", "rave_running_local")
)
```

Arguments

env	environment to load tools
parse_context	parsing context

load_scripts *Load scripts that cannot put into package R folder*

Description

Use in `comp.R` to load scripts that cannot be put into package `"R/"` folder. Usually the scripts contains shiny reactive values that changes dynamically.

Usage

```
load_scripts(..., asis = FALSE)
```

Arguments

`...`	script files that are wrapped by `get_path`, or R quasi-quotations wrapped by `quo`.
`asis`	if the scripts to be loaded is a file, whether to copy to a temporary directory when launching 'RAVE'. Usually we set this to be true to save loading time. However, if your scripts also source other scripts in relative path, we recommend setting `asis=FALSE` and also load additional scripts using this function.

Details

This function raises error when running in default contexts, and requires debug mode, or run inside of 'RAVE' instance.

Value

None, but will source, or run whatever code provided.

See Also

`rave_context`, `get_path`, `quo`

ModuleEnvir *R6 'RAVE' Module Class*

Description

contains module data, functions, etc.

Public fields

`cache_env` cache environment for module

`module_id` module ID, unique

`label_name` corresponding module name

`script_path` compiled module scripts

`script` if `script_path` not exists, alternative script

`author` who wrote the module not often used

`version` module version

`packages` the packages to be loaded for the module

`rmd_path` deprecated

`parent_env` parent environment of the module, usually global environment or package environment

`from_package` whether the module is compiled from another R package. This value is required to be true since `"rave-0.1.9"`.

`package_name` which package does the module belong to?

`sidebar_width` input panel width, from 1 to 11

Methods

Public methods:

- `ModuleEnvir$info()`
- `ModuleEnvir$print()`
- `ModuleEnvir$new()`
- `ModuleEnvir$get_or_new_exec_env()`
- `ModuleEnvir$load_script()`
- `ModuleEnvir$render_ui()`
- `ModuleEnvir$clean()`

Method `info()`**:** print module information

Usage:

```
ModuleEnvir$info()
```

Returns: none

Method `print()`**:** print module information and returns memory address

Usage:

```
ModuleEnvir$print(...)
```

Arguments:

`...` ignored

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

Usage:

```
ModuleEnvir$new(
  module_id,
  label_name,
  script_path,
  author = NULL,
  version = "0",
  packages = NULL,
  .script_content = NULL,
  rmd_path = NULL,
  parent_env = globalenv()
)
```

Arguments:

`module_id`, `label_name`, `script_path`, `author`, `version` see fields

`packages`, `parent_env`, `rmd_path` see fields

`.script_content` internal use

Method `get_or_new_exec_env()`**:** get the corresponding `ExecEnvir` with shiny session

Usage:

```
ModuleEnvir$get_or_new_exec_env(
  session = getDefaultReactiveDomain(),
  ...,
  new = FALSE
)
```

Arguments:

`session` shiny session; see shiny `domains`

`...` ignored

`new` whether to force creating a new runtime environment if previous one already exists

Returns: an `ExecEnvir` instance associated with current module and given session

Method `load_script()`**:** load and compile script into registered `ExecEnvir`

Usage:

```
ModuleEnvir$load_script(session = getDefaultReactiveDomain())
```

Arguments:

`session` shiny session; see shiny `domains`

Returns: none

Method `render_ui()`**:** generate 'HTML' tags

Usage:

```
ModuleEnvir$render_ui(session = getDefaultReactiveDomain())
```

Arguments:

`session` shiny session; see shiny `domains`

Returns: 'HTML' tags

Method `clean()`**:** clean the module environment

Usage:

```
ModuleEnvir$clean(session = getDefaultReactiveDomain(), session_id)
```

Arguments:

`session`	shiny session; see shiny `domains`
`session_id`	shiny 'RAVE' ID, default is auto-generated

Examples

```
## Not run:

module <- get_module('ravebuiltins', 'power_explorer')
module
#> Module Name: Power Explorer
#> Version: 0
#> Script Path: ...
#> Author(s):


## End(Not run)
```

`module_analysis_names` *Find module analysis names*

Description

Find module analysis names

Usage

```
module_analysis_names(
  module_id,
  project_name,
  data_env = getDefaultDataRepository()
)
```

Arguments

`module_id`	module id
`project_name`	project name
`data_env`	internally used

`mount_demo_subject` *Load Demo Subject According to Package Configuration File*

Description

Load Demo Subject According to Package Configuration File

Usage

```
mount_demo_subject(
  subject_code,
  project_name,
  force_reload_subject = FALSE,
  ...,
  download_url
)

## S3 method for class 'rave_module_debug'
mount_demo_subject(
  subject_code,
  project_name,
  force_reload_subject = FALSE,
  ...,
  download_url
)

## S3 method for class 'rave_running'
mount_demo_subject(...)

## S3 method for class 'rave_running_local'
mount_demo_subject(...)
```

Arguments

`subject_code`	optional, subject code
`project_name`	optional, project name
`force_reload_subject`	logical, whether to force reload subject even if another subject is loaded
`...`	further passed to `rave_prepare`
`download_url`	optional, web link to subject archive

Details

When debugging the 'RAVE' modules, it loads demo subject for debugging from according to settings file `"inst/rave.yaml"`.

This function only function properly in `'rave_module_debug'` mode. This means by default it raises errors. In other mode, for example `'rave_running'`, it does nothing.

Value

None

`notch_channel` *Filter line noise out from ECoG channels*

Description

Filter line noise out from ECoG channels

Usage

```
notch_channel(s, sample_rate, bands = c(60, 120, 180), width = c(1, 2, 2))
```

Arguments

`s`	signal, time domain
`sample_rate`	signal sample rate
`bands`	bands that will be filtered out
`width`	along with bands, half of the filter width. For example,if bands is 60Hz and width is 1Hz, then the notch filter lower bound is 60-1=59Hz and upper bound is 60+1=61Hz.

`notch_filter` *Apply Notch Filter to Analog Trace Data*

Description

Apply Notch Filter to Analog Trace Data

Usage

```
notch_filter(s, sample_rate, lb, ub, domain = 1)
```

Arguments

`s`	signal in time or frequency domain
`sample_rate`	signal sample rate
`lb`	filter lower bound (Hz)
`ub`	filter upper bound (Hz)
`domain`	1 if the input signal is in the time domain, 0 if it is in the frequency domain

Details

This function is alternative R version of notch filter

Value

filtered signal in time domain

`plot_signals` *Plot signals line by line*

Description

Plot signals line by line

Usage

```
plot_signals(
  signals,
  sample_rate = 1,
  col = 1,
  space = 0.995,
  space_mode = "quantile",
  start_time = 0,
  duration = NULL,
  compress = TRUE,
  channel_names = NULL,
  ylab = "Channel",
  time_shift = 0,
  lwd = 0.5,
  cex = 2,
  new_plot = TRUE,
  plot = "base",
  xlim = NULL,
  ...
)
```

Arguments

`signals`	signals to plot, with each row one signal
`sample_rate`	sample rate
`col`	Color, either length of 1 or number of signals. Can be numeric or color name
`space`	space between to signals. If `space_mode='quantile'`, then space is determined by quantile of signal (0 - 1). If `space_mode='absoute'`, then space will be as is.
`start_time`	Time in seconds at which time point the signal should be drawn

duration	length of 1. Time in seconds the duration of time to be drawn. Default is NULL (Total time range)
compress	FALSE means no compression for signals, TRUE is auto-detection, 2, 3, 4,... means compress signals by x and then plot. (usually compress signal to save time)
channel_names	Names for each signals. Will be Y tick labels
ylab	Y axis label
plot, xlim, space_mode, time_shift, lwd, cex, new_plot	Deprecated
...	pass to matplot

---

progress *A wrapper for shiny Progress object*

Description

A wrapper for shiny Progress object

Usage

```
progress(
  title,
  max = 1,
  session = getDefaultReactiveDomain(),
  quiet = FALSE,
  ...
)
```

Arguments

title	Main title for progress bar
max	How many steps you have for this process
session	shiny session, default is getDefaultReactiveDomain
quiet	nonreactive-only mode? default is FALSE. If TRUE, then progress bar will be hidden in shiny app
...	other parameters passing to progress2

Details

shiny::Progress class cannot be used under non-reactive environment. rave::progress function wrap it up so that you can use it in non-reactive settings.

pwelch *Plot "Welch" Periodogram*

Description

Plot "Welch" Periodogram

Usage

```
pwelch(
  x,
  fs,
  window = 64,
  noverlap = 8,
  nfft = 256,
  col = "black",
  xlim = NULL,
  ylim = NULL,
  main = "Welch periodogram",
  plot = TRUE,
  log = "xy",
  spec_func = stats::spectrum,
  cex = 1,
  ...
)
```

Arguments

x	signal
fs	sample rate
window	window length, default 128
noverlap	overlap between two adjacent windows, by default is 8
nfft	number of basis functions
col, xlim, ylim, main, cex, ...	will be passed to plot
plot	logical, plot the result or not
log	indicates which axis should be log10 value. options are '', 'x', 'y', 'xy'.
spec_func	deprecated

rave-cache *Cache R Objects with Different levels*

Description

Cache R Objects with Different levels

Usage

```
cache(
  key,
  val,
  name,
  replace = FALSE,
  global = FALSE,
  persist = FALSE,
  test = FALSE,
  temporary = FALSE,
  ...
)

## S3 method for class 'rave_running'
cache(
  key,
  val,
  name,
  replace = FALSE,
  global = FALSE,
  persist = FALSE,
  test = FALSE,
  temporary = FALSE,
  ...
)

## S3 method for class 'rave_running_local'
cache(..., global = TRUE)

## Default S3 method:
cache(..., global = TRUE)

cache_input(
  inputId,
  val = NULL,
  read_only = TRUE,
  ...,
  session = getDefaultReactiveDomain()
)
```

```
clear_cache(levels = 1)
```

Arguments

key	any R object, a named list would be the best.
val	value to cache, if key exists, then value will not be evaluated nor saved
name, inputId	character name of the dataset or input
replace	if true, force replace the cached object with current one
global	whether to cache the variable in global environment. If true, then the variable will be accessible from other instances and modules.
persist	logical, whether persist on the hard-disk, only used when global=FALSE, the persisted data will be used by each modules
test, read_only	whether not to save the value if cache is not found
temporary	whether to use temporary map to cache, used internally.
...	ignored
session	shiny session instance
levels	levels when clear the cache

Value

Cached value, or val. If cache and val are both missing, then return NULL.

Examples

```
# global can be set to false within RAVE modules
print(cache('a', 1, name = 'data', global = TRUE)) # returns 1
print(cache('a', 2, name = 'data', global = TRUE)) # still returns 1

# clear cache (for global=TRUE)
clear_cache(levels = 1:3)
print(cache('a', 2, name = 'data', global = TRUE)) # Now returns 2

# Not run `Sys.sleep` because a is cached
print(cache('a', 2, name = 'data', global = TRUE))
print(cache('a', {Sys.sleep(10); 1}, name = 'data', global = TRUE))

# get data without key
cache(name = 'data', global = TRUE)

# clear cache that is global-only
clear_cache(levels = 2)

# Test (test=TRUE) if cache exists, if not, return value but no save
cache(name = 'abracadabra', val = 'no cache', global = TRUE, test = TRUE)
cache(name = 'abracadabra', global = TRUE)

# cache module inputs
```

```
## Not run:
# Need to run in package module environment
cache_input('abracadabra', 'no-magic', read_only = TRUE)

## End(Not run)
```

---

rave-tabs *Open/Close a tab in RAVE main application*

Description

Open/Close a tab in RAVE main application

Usage

```
close_tab(module_id, tabname)

open_tab(module_id, tabname)
```

Arguments

module_id	character, module ID
tabname	character, tab box title

---

rave_brain2 *Tools to load and view brain in 3D viewer*

Description

Tools to load and view brain in 3D viewer

Usage

```
rave_brain2(
  subject,
  surfaces = "pial",
  use_141 = TRUE,
  recache = FALSE,
  clean_before_cache = FALSE,
  compute_template = FALSE,
  usetemplateifmissing = FALSE
)
```

Arguments

subject	character or 'RAVE' subject instance
surfaces	one or more from "pial", "white", "smoothwm", brain surface types
use_141	logical, whether to use standard 141 brain
recache	whether to force cache data, default is false
clean_before_cache	whether to clean cache before redo cache, default is false
compute_template	logical whether to compute nearest 141 node. Please also check freesurfer_brain.
usetemplateifmissing	whether logical, to display template brain if subject brain not found, default is false

---

rave_checks *Check if data is loaded for current module*

Description

Check if data is loaded for current module

Usage

```
rave_checks(
  ...,
  data = NULL,
  .raise_error = TRUE,
  rave_data = getDefaultDataRepository()
)
```

Arguments

...	see details
data	same as ..., but can be a vector
.raise_error	whether to raise error if data is missing
rave_data	internally used

Details

This function checks whether "ECoG" data is loaded. The format is: "DATA+(blankspace)+TYPE". "DATA" can be "power" (wavelet transform amplitude), "phase" (complex angle), or "volt"/"voltage" (Before wavelet). "TYPE" can be "raw" (no reference), "referenced" (referenced by common average reference, white matter reference, or bipolar reference). For voltage data, there is one more special type "full" which loads voltage data for all electrodes.

rave_context *'RAVE' Context: Read and Set Context of Environments*

Description

'RAVE' Context: Read and Set Context of Environments

Usage

```
rave_context(
  context,
  require_contexts,
  disallowed_context,
  error_msg,
  spos = 2L,
  senv,
  tpos = 1L,
  tenv
)
```

Arguments

context	context string for target environment, optional, see 'Details'
require_contexts	characters, (optional): required context for current function. If any context is missing, the function will raise errors
disallowed_context	characters, (optional): defines the contexts that don't work for the function. If running within such contexts, the function will raise errors
error_msg	characters, (optional): if running in improper contexts, the message to display, will passed to stop
spos	levels to go up to search for senv, passed to parent.frame
senv	environment to read 'RAVE' contexts
tpos	levels to go up to search for tenv, passed to parent.frame
tenv	environment to set 'RAVE' contexts

Details

Context strings tells the function which context it's running, and it will affect the behaviors of functions within its environment. Because 'RAVE' modules are usually R packages, the context strings help the module writers determine where the function is running. For example, running locally, or in 'RAVE' container, or debug mode. A typical example would be get_path function. All external scripts used in R packages require to be obtained using system.file. However, because the files are subject to change, using system file function requires re-compile the package, which is time-consuming. Function get_path returns the file path relative to current working directory during the development (in "default" context), and it calls system.file when 'RAVE' instance is running.

There are four contexts: "default", "rave_module_debug", "rave_running", and "rave_running_local".

default Default context: this means the function is running without any additional information.

rave_module_debug Debug mode: used to develop and debug modules locally. Under the context, the function will be aware of the package that module belongs to

rave_running If the function is running under this context, this means it's running inside of shiny application (usually within start_rave). The function will be able to get more contexts such as module ID, and current runtime environment (ExecEnvir)

rave_running_local Similar to "rave_running", but without run-time environment. Under this context, the module is running locally without shiny. All reactive observers are disabled, and the modules will be compiled into a function with all the inputs defined by define_input as arguments, and code within "main.R" as the main body of the function.

Function rave_context uses reserved variables in the environment: .__rave_context__., .__rave_package__., .__rave_module__., and .__rave_module_instance__.. Please don't use these variables for other purposes. See 'Examples' for how to set and read the context.

Value

A list of current context, including the package name, module ID, and current ExecEnvir instance if running under "rave_running" context.

Examples

```
# ------- 1. Read/Set Context ---------

library(dipsaus)
library(rave)
# Reset context for current environment
rave_context('default')

# Read from current caller's environment
fun <- function(...){
  ctx <- rave_context()
  cat2('The function is running under context - ', ctx$context)
  cat2('The package under the context - ', ctx$package)
  cat2('Module ID is - ', ctx$module_id)
  cat2('Running instance is - ', ctx$instance)
}
fun()
## The function is running under context - default
## The package under the context -
## ...

# Set debug context
debug_env <- new.env()
rave_context('rave_module_debug', tenv = debug_env)
debug_env$.__rave_package__. <- 'ravebuiltins'

# With debug_env, the function is aware of the package it's under
with(debug_env, { fun() })
```

```
## The function is running under context - rave_module_debug
## The package under the context - ravebuiltins
## ...

# To set context within the function and affect the functions inide
fun2 <- function(module_id){
  # Run rave_context and then set module ID
  rave_context('rave_running_local')
  .__rave_module__. <- module_id
  fun()
}
with(debug_env, { fun2('power_explorer') })
## The function is running under context - rave_running_local
## The package under the context - ravebuiltins
## Module ID is - power_explorer
## ...

# Let's see what we can do with rave_module_debug
with(debug_env, { get_path('inst/rave.yaml') })
# When I develop the package, it returns:
## "/Users/beauchamplab/.../ravebuiltins/inst/settings.yaml"
# When I run in other places, it returns
## "/Users/beauchamplab/Library/R/3.6/library/ravebuiltins/rave.yaml"


# ------- 2. Setting behaviors for context ---------
# One way to set different behaviors is to using `ctx`
## Not run:
fun <- function(){
  ctx <- rave_context()
  switch(ctx$context, ...)
}

## End(Not run)

# The other way is to use S3 generics provided by R syntax
fun <- rave_context_generics('fun', function(module_id, ...){})

# action for default
fun.default <- function(...){
  cat2('Function is not supposed to run under default context...',
       level = 'ERROR')
}

# for debug, set module ID and run with rave_running_local
fun.rave_module_debug <- function(module_id, ...){
  cat2('Debug mode... loading a test subject')
  # Do something ... like automatically mount_demo_subject
  # by running mount_demo_subject()

  rave_context('rave_running_local')
  .__rave_module__. <- module_id
  # Recall the function under rave_running_local context
```

```
  fun(module_id, ...)
}

# When running within RAVE container, local and with shiny
fun.rave_running_local <- function(...){
  ctx <- rave_context()
  cat2('Yay, running ', ctx$module_id, ' under context ',
       ctx$context, level='INFO')
}
fun.rave_running <- fun.rave_running_local

# Run in default mode, expect error message
fun('power_explorer')

# Run in debug mode
debug_env <- new.env()
rave_context('rave_module_debug', tenv = debug_env)
debug_env$.__rave_package__. <- 'ravebuiltins'

# The function will run in debug mode, then rave_running_local
with(debug_env, { fun('power_explorer') })
```

`rave_context_generics` *Create S3 Generics that Respects 'RAVE' Context*

Description

Create S3 Generics that Respects 'RAVE' Context

Usage

```
rave_context_generics(
  fun_name,
  fun = function() {
 }
)
```

Arguments

`fun_name`	generic function name
`fun`	function that set the arguments of the generic

Value

A generic function

rave_failure *RAVE Failure Message*

Description

RAVE Failure Message

Usage

```
rave_failure(message, level = "ERROR", call = NULL, .stop = TRUE)
```

Arguments

message	error message, character
level	level of error message; can be chosen from "INFO", "WARNING", or "ERROR"
call	call expression
.stop	stop or just return the condition

Value

Error condition or stop

rave_ignore *Functions for development use*

Description

Functions for development use

Usage

```
rave_ignore(...)
```

Arguments

...	Expressions

rave_import_rawdata *Import Raw Signal from Non-standard Formats*

Description

Import Raw Signal from Non-standard Formats

Usage

```
rave_import_rawdata(subject_code, project_name, launch_preprocess = TRUE)
```

Arguments

subject_code	subject code to search for in the raw folder
project_name	project name to create
launch_preprocess	whether to launch preprocess app, default is true

rave_module_tools *Tools for module writers*

Description

Tools for module writers

Usage

```
rave_module_tools(
  env = NULL,
  data_env = getDefaultDataRepository(),
  quiet = FALSE
)
```

Arguments

env	environment to save tools in
data_env	rave data repository returned by rave_prepare, internally used
quiet	logical

rave_options *Function to change rave-options*

Description

Function to change rave-options

Usage

```
rave_options(
  ...,
  .save = TRUE,
  launch_gui = TRUE,
  host = "127.0.0.1",
  port = NULL
)
```

Arguments

...	Key-Value option pairs
.save	save to disk? ignored most of the time
launch_gui	launch shiny app?
host	IP address of host
port	Port number

rave_prepare *Load Subject and Create* iEEG/ECoG *Data Environment*

Description

Loads subject data along with iEEG/ECoG data into memory.

Usage

```
rave_prepare(
  subject,
  electrodes,
  epoch,
  time_range,
  frequency_range,
  data_types = c("power"),
  reference = "default",
  attach = "r",
  data_env = getDefaultDataRepository(),
  strict = FALSE,
  ...
)
```

Arguments

subject	characters, format: "PROJECT/SUBJECT"
electrodes	integer vector, which electrodes to be loaded
epoch	characters, name for epoch data. For example, "epoch1" refers to epoch file "epoch_epoch1.csv" in subject meta folder
time_range	vector of length 2. time_range[1]=1 indicates 1 second before onset will be loaded; time_range[2]=1.5 means 1.5 seconds after onset will be loaded. Make sure both are positive number in seconds
frequency_range	vector of length 2 - lowest and highest frequencies. By default is all frequencies. Only applied to power and phase data.
data_types	vector of characters, data to be pre-loaded. "power" refers to referenced power (power spectrum) data, "phase" refers to referenced phase data, and "volt" is referenced voltage (original analog traces) data
reference	name of reference data. For example, "default" refers to reference file "reference_default.csv" in subject meta folder
attach	characters or NULL, NULL if you don't want to attach it, "r" if want to load data as R environment, "py" is for python, and "matlab" is for Matlab. (python and Matlab are under construction)
data_env	environment to load data into.
strict	whether to check if raw data exists. Default is no (suggested)
...	ignored

---

rave_preprocess *RAVE Preprocess Function*

Description

RAVE Preprocess Function

Usage

```
rave_preprocess(
  sidebar_width = 3,
  launch.browser = TRUE,
  host = "127.0.0.1",
  port = NULL,
  quiet = TRUE,
  beta = FALSE,
  test.mode = FALSE,
  modules,
  ver = "3",
  theme = "purple",
  ...
)
```

Arguments

sidebar_width	sidebar width from 1 to 11.
launch.browser	whether to launch browser, default is on
host	default is "localhost"
port	integer port of the app
quiet	soft deprecated
beta	whether to load experimental modules, default is false
test.mode	passed to shinyApp
modules	preprocess modules to load, reserved
ver	internally used please don't change
theme	color theme
...	used for other functions for configuration and debug only

---

rave_preprocess_tools *Function to create RAVE preprocess tools*

Description

Function to create RAVE preprocess tools

Usage

```
rave_preprocess_tools(env = new.env(), ...)
```

Arguments

env	environment to save tools to
...	ignored

---

rave_version *Get RAVE version*

Description

Get RAVE version

Usage

```
rave_version()
```

read_mgrid *Make* iElvis mgrid *file*

Description

Make iElvis mgrid file

Usage

```
read_mgrid(con, raw = FALSE)
```

Arguments

con	mgrid file
raw	raw file or processed I guess

Author(s)

John Magnotti

reload_module_package *Reload 'RAVE' module package without restarting 'RStudio'*

Description

For debugging module packages. In all other contexts it will raise error.

Usage

```
reload_module_package(expose = FALSE, clear_env = FALSE)
```

Arguments

expose	whether to expose development tools to the global environment; default is no
clear_env	whether to clear the global environment before reloading; default is no

r_to_py.Subject *Convert subject to python object*

Description

Convert subject to python object

Usage

```
r_to_py.Subject(obj, convert = FALSE)
```

Arguments

obj	Subject class
convert	pass to r_to_py in reticulate package

safe_write_csv *Save data to "CSV", if file exists, rename old file*

Description

Save data to "CSV", if file exists, rename old file

Usage

```
safe_write_csv(data, file, ..., quiet = FALSE)
```

Arguments

data	data frame
file	"CSV" file to save
...	pass to write.csv
quiet	suppress overwrite message

save_meta *Function to save meta data to subject*

Description

Function to save meta data to subject

Usage

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

Arguments

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

save_options *Function to locally save options (deprecated)*

Description

Function to locally save options (deprecated)

Usage

```
save_options()
```

session-reactives *Get 'shiny' "input" and "output" objects under current context*

Description

Get 'shiny' "input" and "output" objects under current context

Usage

```
getDefaultReactiveInput(session)

## Default S3 method:
getDefaultReactiveInput(session)

## S3 method for class 'rave_module_debug'
getDefaultReactiveInput(session)

## S3 method for class 'rave_running'
getDefaultReactiveInput(session = shiny::getDefaultReactiveDomain())

## S3 method for class 'rave_running_local'
getDefaultReactiveInput(session)

getDefaultReactiveOutput(session = shiny::getDefaultReactiveDomain())

## Default S3 method:
getDefaultReactiveOutput(session = shiny::getDefaultReactiveDomain())

## S3 method for class 'rave_module_debug'
getDefaultReactiveOutput(session = shiny::getDefaultReactiveDomain())

## S3 method for class 'rave_running'
getDefaultReactiveOutput(session = shiny::getDefaultReactiveDomain())

## S3 method for class 'rave_running_local'
getDefaultReactiveOutput(session = shiny::getDefaultReactiveDomain())
```

Arguments

session	shiny session instance

Value

In shiny context, returns special reactiveValues that refers to the inputs and outputs of shiny applications. In non-shiny contexts, returns a fake environment related to current fake session, for debug purpose.

---

set_rave_theme *Set and Return RAVE theme*

Description

Set and Return RAVE theme

Usage

```
set_rave_theme(
  theme,
  .set_default = FALSE,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

theme	"light" or "dark". See details if missing
.set_default	whether to save current theme as default, default is no.
session	shiny session

Details

RAVE support two themes: "light" mode and "dark" mode. In "light" mode, the web application background will be light gray and white. In "dark" mode, the application background will be gray and foreground will be white.

If theme is missing and RAVE is running as web application, then it is set from current session, otherwise, the default theme is retrieved from `rave_options('default_theme')`. If option "default_theme" is missing, then it defaults to "light".

Value

theme under current context.

Examples

```
# Retrieve current theme
get_val(rave_options('default_theme'), default = 'light')

# Set light mode
set_rave_theme('light')
plot(1:10, main = 'test light mode')

# Set dark mode
set_rave_theme('dark')
plot(1:10, main = 'test dark mode')
```

---

shinirize *Convert module to objects used in shiny*

Description

Convert module to objects used in shiny

Usage

```
shinirize(
  module,
  session = getDefaultReactiveDomain(),
  test.mode = TRUE,
  data_env = getDefaultDataRepository()
)
```

Arguments

module	ModuleEnvir object
session	shiny session, default is current shiny session
test.mode	passed by start_rave or init_app
data_env	internally used

---

start_rave *Start RAVE main application*

Description

Start RAVE main application

Usage

```
start_rave(
  modules = NULL,
  active_module = NULL,
  launch.browser = TRUE,
  theme = "purple",
  disable_sidebar = FALSE,
  simplify_header = FALSE,
  token = NULL,
  data_repo = getDefaultDataRepository(),
  ...
)

launch_demo(
  modules = "power_explorer",
  launch.browser = TRUE,
  theme = "green",
  disable_sidebar = TRUE,
  simplify_header = FALSE,
  ...
)

start_rave2(
```

```
  host = "127.0.0.1",
  port = NULL,
  launch.browser = TRUE,
  jupyter = FALSE,
  as_job = FALSE,
  ...
)
```

Arguments

modules	character vector, modules modules to load before starting application.
active_module	character, which module to show as default.
launch.browser	logical, whether to launch browser.
theme	character, color theme, default is 'purple'.
disable_sidebar	logical, whether to hide sidebar.
simplify_header	logical, whether to show simplified header.
token	character vector, default is NULL. If specified, then a ?token=... is needed in url to access to the application.
data_repo	internally used.
...	other parameters. See details.
host, port, jupyter, as_job	'RAVE' 2.0 related arguments; see start_session

---

start_yael *Start 'YAEL' electrode localization*

Description

Start 'YAEL' electrode localization

Usage

```
start_yael(
  host = "127.0.0.1",
  port = NULL,
  launch.browser = TRUE,
  as_job = FALSE,
  ...
)
```

Arguments

host	host IP address
port	integer port number; default is random
launch.browser	whether to launch browsers
as_job	whether to launch in background; available only in 'RStudio'
...	passed to start_session

Subject *R6 Class for 'RAVE' Subject*

Description

contains subject meta information after preprocessing.

Public fields

`meta` environment stores subject meta data

`subject_id` character, subject ID, generated from project name and subject code. For example, project name is `"congruency"` and subject code is `"YAB"`, then the `subject_id="congruency/YAB"`

`subject_code` identifier for subject

`project_name` project name

`dirs` stores folder paths for subject data

`is_strict` whether preprocess directory is checked when initializing the instance

Active bindings

`electrodes` electrode table (read-only)

`frequencies` frequency table (read-only)

`time_points` time-point table (read-only)

`time_excluded` (deprecated) excluded time-point table (read-only)

`sample_rate` time-point table (read-only, for compatibility issues)

`volt_sample_rate` voltage (trace) sampling rate in Hertz (read-only)

`power_sample_rate` power (amplitude) sampling rate in Hertz (read-only)

`phase_sample_rate` phase sampling rate in Hertz (read-only)

`valid_electrodes` all valid electrodes in current reference scheme (read-only)

`id` read-only version of subject ID

Methods

Public methods:

- Subject$info()
- Subject$print()
- Subject$finalize()
- Subject$new()
- Subject$preprocess_info()
- Subject$filter_all_electrodes()
- Subject$filter_valid_electrodes()
- Subject$has_bad_time_point()
- Subject$clone()

Method `info()`**:** print the information of the subject

Usage:

```
Subject$info()
```

Returns: none

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

Usage:

```
Subject$print(...)
```

Arguments:

`...` ignored

Returns: default memory address of the environment

Method `finalize()`**:** called when garbage collected

Usage:

```
Subject$finalize()
```

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

Usage:

```
Subject$new(project_name, subject_code, reference = NULL, strict = TRUE)
```

Arguments:

`project_name` project name

`subject_code` subject code

`reference` what kind of reference is default for the subject, default is "default", referring to `"reference_default.csv"` in subject meta folder

`strict` whether to check if the raw folder exists

Method `preprocess_info()`**:** Obtain preprocessing information. This methods is rarely directly called, I wrap up most commonly used fields in other functions

Usage:

```
Subject$preprocess_info(key, default = NULL, customized = FALSE)
```

Arguments:

`key` the fields or items store in `SubjectInfo2` instance

`default` default value if the key is not found

`customized` indicates whether the key refers to additional items or fields in `SubjectInfo2`. Default is false, meaning the key is the fields.

Returns: the preprocess information correspond to the key

Method `filter_all_electrodes()`**:** filter, and returns existing electrodes

Usage:

```
Subject$filter_all_electrodes(electrodes)
```

Arguments:

`electrodes` integer vector

Returns: the electrodes that the subject has, including bad, or invalid electrodes.

Method `filter_valid_electrodes()`**:** filter, and returns valid electrodes

Usage:

```
Subject$filter_valid_electrodes(electrodes)
```

Arguments:

`electrodes` integer vector

Returns: the valid electrodes. Invalid electrodes refers to bad electrodes, or the end of bipolar reference. If `"Reference"` column is blank in the reference file, then the electrode is invalid.

Method `has_bad_time_point()`**:** (deprecated) check whether the selected time is excluded

Usage:

```
Subject$has_bad_time_point(block, electrode, start, end)
```

Arguments:

`block` block name

`electrode` electrode number

`start` start time

`end` end time

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Zhengjia Wang

Examples

```
## Not run:

# Load subject, use `strict=FALSE` if not sure the existence of raw files
subject <- Subject$new(project_name = 'demo', 'YAB', strict = FALSE)

# Filter 1:14 to see which numbers refer to the valid electrodes
subject$filter_valid_electrodes(1:14)
#> [1] 13 14


## End(Not run)
```

`subject_tmpfile` *Create temp file in subject module folder*

Description

Create temp file in subject module folder

Usage

```
subject_tmpfile(
  module_id,
  fun_name = "",
  project_name,
  subject_code,
  pattern = "file_",
  data_env = getDefaultDataRepository()
)
```

Arguments

`module_id`	module id
`fun_name`	function name (usually export_"function_name" in the module)
`project_name`	project name
`subject_code`	subject code
`pattern`	passed to `tempfile`
`data_env`	internally used

suma_spec_parse *Parse* SUMA spec *file*

Description

Parse SUMA spec file

Usage

```
suma_spec_parse(subject, spec_file)
```

Arguments

subject	either characters in format like 'Project/Subject' or Subject object created by Subject$new(...)
spec_file	default decided by rave_options('suma_spec_file'), depending on subjects

Examples

```
## Not run:
subject = 'Demo/YAB'
# or create subject object
subject = Subject$new('Demo', 'YAB')

# Please download sample subjects first to run
suma_spec_parse(subject)

## End(Not run)
```

suma_surface_volume_parse *Parse* AFNI BRIK/HEAD *file*

Description

Parse AFNI BRIK/HEAD file

Usage

```
suma_surface_volume_parse(file_path)
```

Arguments

file_path	path to BRIK or HEAD file

to_module *Parse 'RAVE' Module and Returns Parsed Content in Environments*

Description

Parse 'RAVE' Module and Returns Parsed Content in Environments

Usage

```
to_module(
  module_id,
  sidebar_width = 3,
  parse_context = c("rave_running_local", "rave_running")
)
```

Arguments

module_id	module ID
sidebar_width	input sidebar width
parse_context	parse context, default is "rave_running_local"

view_layout *Debug-use only, reload package, mount demo subject, and launch shiny app*

Description

Debug-use only, reload package, mount demo subject, and launch shiny app

Usage

```
view_layout(
  module_id,
  sidebar_width = 3,
  launch.browser = TRUE,
  reload = TRUE,
  ...
)
```

Arguments

module_id	module ID to debug
sidebar_width	input width, from 1 to 11, default is 3
launch.browser	whether to launch browser, default is true, other options are rstudioapi::viewer or false.
reload	whether to reload package first. default is true,
...	passed to init_app

---

wavelet *Wavelet Transformation With Phase*

Description

The code was translated from Matlab script written by Brett Foster, Stanford Memory Lab, 2015 with permission to use in 'RAVE'.

Usage

```
wavelet(data, freqs, srate, wave_num, demean = TRUE)
```

Arguments

data	- vector of time series to be decomposed
freqs	- vector of center frequencies for decomposition
srate	- sample rate (in Hz)
wave_num	- desired number of cycles in wavelet (typically 3-20 for frequencies 2-200).
demean	- whether to remove the mean of data first?

Details

Decompose time series data into time-frequency representation (spectral decomposition) using wavelet transform. Employs "Morlet" wavelet method (gaussian taper sine wave) to obtain the analytic signal for specified frequencies (via convolution).

`wavelet_kernels` *Returns wavelets to be used for wavelet function*

Description

Returns wavelets to be used for wavelet function

Usage

```
wavelet_kernels(freqs, srate, wave_num)
```

Arguments

`freqs`	vector of center frequencies for decomposition
`srate`	sample rate (in Hz)
`wave_num`	desired number of cycles in wavelet (typically 3-20 for frequencies 2-200).

Index