Package: ravedash (via r-universe)

November 1, 2024

Type Package

Title Dashboard System for Reproducible Visualization of 'iEEG'

Version 0.1.3.36

Description Dashboard system to display the analysis results produced by 'RAVE' (Magnotti J.F., Wang Z., Beauchamp M.S. (2020), Reproducible analysis and visualizations of 'iEEG' <doi:10.1016/j.neuroimage.2020.117341>). Provides infrastructure to integrate customized analysis pipelines into dashboard modules, including file structures, front-end widgets, and event handlers.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Imports grDevices, stats, dipsaus (>= 0.2.0), logger (>= 0.2.2), raveio (>= 0.0.6), rpymat (>= 0.1.2), shidashi (>= 0.1.1), shiny (>= 1.7.1), shinyWidgets (>= 0.6.2), threeBrain (>= 0.2.4), shinyvalidate, htmlwidgets, R6, checkmate, cachem (>= 1.0.6)

Suggests htmltools, fastmap (>= 1.1.0), rlang (>= 1.0.2), crayon (>= 1.4.2), rstudioapi, knitr, httr, rmarkdown, testthat (>= 3.0.0)

RoxygenNote 7.3.2

URL https://dipterix.org/ravedash/

BugReports https://github.com/dipterix/ravedash/issues

VignetteBuilder knitr

Config/testthat/edition 3

Repository https://rave-ieeg.r-universe.dev

RemoteUrl https://github.com/dipterix/ravedash

RemoteRef HEAD

RemoteSha 83348d5c43c0d68d3f5f475b3e514cd46ea4ce9d

Contents

`card_badge` *Create a badge widget located at card header*

Description

Create a badge widget located at card header

Usage

```
card_badge(text = NULL, class = NULL, ...)

card_recalculate_badge(text = "Recalculate needed", class = NULL, ...)

enable_recalculate_badge(text = "Recalculate needed", ...)

disable_recalculate_badge(text = "Up-to-date", ...)
```

```
set_card_badge(
  id = NULL,
  class = NULL,
  text = NULL,
  add_class = NULL,
  remove_class = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

`text`	inner text content of the badge
`class`	additional 'HTML' class of the badge; for `set_card_badge`, this is the class selector of the badge that is to be changed
`...`	additional 'HTML' tag attributes
`id`	the badge 'HTML' ID to be changed, will be enclosed with session namespace `session$ns(id)` automatically.
`add_class`, `remove_class`	add or remove class
`session`	shiny session

Examples

```
library(ravedash)

# UI: a Bootstrap badge with green background
card_badge("Ready", class = "bg-green rave-output-status")

# server
server <- function(input, output, session) {

  safe_observe({

    # ... check if the inputs have changed

    set_card_badge(
      class = "rave-output-status",
      text = "Refresh needed",
      add_class = "bg-yellow",
      remove_class = "bg-green"
    )

  })

}
```

card_url *Set 'URL' scheme for modules*

Description

Automatically generates `href` for `input_card` and `output_card`

Usage

```
set_card_url_scheme(module_id, root, sep = "/")

card_href(title, type = "input", module_id = NULL)
```

Arguments

`module_id`	the module ID
`root`	'URL' default route
`sep`	separation
`title`	a title string that will be used to generate 'URL'
`type`	type of the card; choices are `'input'` or `'output'`

Value

The hyper reference of suggested card 'URL'

Examples

```
set_card_url_scheme(
  module_id = "power_explorer",
  root = "https://openwetware.org/wiki/RAVE:ravebuiltins",
  sep = ":")

card_href("Set Electrodes", type = "input", module_id = "power_explorer")
```

debug_modules *Debug 'RAVE' modules interactively in local project folder*

Description

Debug 'RAVE' modules interactively in local project folder

Usage

```
debug_modules(
  module_root = rstudioapi::getActiveProject(),
  host = "127.0.0.1",
  port = 17283,
  jupyter = FALSE,
  ...
)
```

Arguments

module_root	root of modules, usually the project folder created from 'shidashi' template
host, port	host and port of the application
jupyter	whether to launch 'Jupyter' server; default is false
...	passed to render

Value

'RStudio' job ID

---

get_active_module_info

Get current active module information, internally used

Description

Get current active module information, internally used

Usage

```
get_active_module_info(session = shiny::getDefaultReactiveDomain())

get_active_pipeline(session = shiny::getDefaultReactiveDomain())
```

Arguments

session	shiny reactive domain, default is current domain

Value

A named list, including module ID, module label, internal 'rave_id'.

group_box *Group input elements into a box with title*

Description

Only works in template framework provided by 'shidashi' package, see use_template

Usage

```
group_box(title, ..., class = NULL)

flex_group_box(title, ..., class = NULL, wrap = "wrap", direction = "row")
```

Arguments

title	the box title
...	elements to be included or to be passed to other methods
class	additional class of the box
wrap, direction	see flex_container

Value

A 'HTML' tag

Examples

```
library(shiny)
library(shidashi)
library(ravedash)

group_box(
  title = "Analysis Group A",
  selectInput("a", "Condition", choices = c("A", "B")),
  sliderInput("b", "Time range", min = 0, max = 1, value = c(0,1))
)

flex_group_box(
  title = "Project and Subject",
  flex_item( "Some input 1" ),
  flex_item( "Some input 2" ),
  flex_break(),
  flex_item( "Some input in new line" )
)
```

logger *Logger system used by 'RAVE'*

Description

Keep track of messages printed by modules

Usage

```
logger(
  ...,
  level = c("info", "warning", "error", "fatal", "debug", "trace"),
  calc_delta = "auto",
  .envir = parent.frame(),
  .sep = "",
  use_glue = FALSE,
  reset_timer = FALSE
)

set_logger_path(root_path, max_bytes, max_files)

logger_threshold(
  level = c("info", "warning", "error", "fatal", "debug", "trace"),
  module_id,
  type = c("console", "file", "both")
)

logger_error_condition(cond, level = "error")

error_notification(
  cond,
  title = "Error found!",
  type = "danger",
  class = "error_notif",
  delay = 30000,
  autohide = TRUE,
  collapse = "\n",
  prefix = paste("Found the following error",
    "(details have been printed in the console):"),
  session = shiny::getDefaultReactiveDomain()
)

error_alert(
  cond,
  title = "Error found!",
  type = "error",
  danger_mode = TRUE,
```

```
  auto_close = FALSE,
  prefix = paste("Found the following error",
    "(details have been printed in the console):"),
  buttons = "Confirm",
  session = shiny::getDefaultReactiveDomain()
)

with_error_notification(expr, envir = parent.frame(), quoted = FALSE, ...)

with_error_alert(expr, envir = parent.frame(), quoted = FALSE, ...)
```

Arguments

`...`, `.envir`, `.sep`	passed to `glue`, if `use_glue` is true
`level`	the level of message, choices are `'info'` (default), `'warning'`, `'error'`, `'fatal'`, `'debug'`, `'trace'`
`calc_delta`	whether to calculate time difference between current message and previous message; default is `'auto'`, which prints time difference when `level` is `'debug'`. This behavior can be changed by altering `calc_delta` by a logical `TRUE` to enable or `FALSE` to disable.
`use_glue`	whether to use `glue` to combine `...`; default is false
`reset_timer`	whether to reset timer used by `calc_delta`
`root_path`	root directory if you want log messages to be saved to hard disks; if `root_path` is `NULL`, `""`, or `nullfile`, then logger path will be unset.
`max_bytes`	maximum file size for each logger partitions
`max_files`	maximum number of partition files to hold the log; old files will be deleted.
`module_id`	'RAVE' module identification string, or name-space; default is `'ravedash'`
`type`	which type of logging should be set; default is `'console'`, if file log is enabled through `set_logger_path`, `type` could be `'file'` or `'both'`. Default log level is `'info'` on console and `'debug'` on file.
`cond`	condition to log
`class`, `title`, `delay`, `autohide`	passed to `show_notification`
`collapse`, `danger_mode`, `auto_close`, `buttons`	will be passed to `shiny_alert2` or `show_notification`
`prefix`	additional messages to display in the notification or alert
`session`	shiny session
`expr`	expression to evaluate
`envir`	environment to evaluate `expr`
`quoted`	whether `expr` is quoted; default is false

Value

The message without time-stamps

Examples

```
logger("This is a message")

a <- 1
logger("A message with glue: a={a}")

logger("A message without glue: a={a}", use_glue = FALSE)


logger("Message A", calc_delta = TRUE, reset_timer = TRUE)
logger("Seconds before logging another message", calc_delta = TRUE)


# by default, debug and trace messages won't be displayed
logger('debug message', level = 'debug')

# adjust logger level, make sure `module_id` is a valid RAVE module ID
logger_threshold('debug', module_id = NULL)

# Debug message will display
logger('debug message', level = 'debug')

# Trace message will not display as it's lower than debug level
logger('trace message', level = 'trace')
```

---

module_server_common *Default module server function*

Description

Common shiny server function to enable modules that requires data loader panel.

Usage

```
module_server_common(
  module_id,
  check_data_loaded,
  ...,
  session = shiny::getDefaultReactiveDomain(),
  parse_env = NULL
)
```

Arguments

module_id	'RAVE' module ID
check_data_loaded	a function that takes zero to one argument and must return either TRUE if data has been loaded or FALSE if loader needs to be open to load data.

...	ignored
session	shiny session
parse_env	environment used to parse module

Value

A list of server utility functions; see 'Examples' below.

Examples

```
# Debug in non-reactive session: create fake session
fake_session <- shiny::MockShinySession$new()

# register common-server function
module_server_common(module_id = "mock-session",
                     session = fake_session)
server_tools <- get_default_handlers(fake_session)

# Print each function to see the usage

server_tools$auto_recalculate

server_tools$run_analysis_onchange

server_tools$run_analysis_flag

server_tools$module_is_active

server_tools$simplify_view

# 'RAVE' module server function
server <- function(input, output, session, ...){

  pipeline_path <- "PATH to module pipeline"

  module_server_common(
    module_id = session$ns(NULL),
    check_data_loaded = function(first_time){

      re <- tryCatch({
        # Try to read data from pipeline results
        repo <- raveio::pipeline_read(
          'repository',
          pipe_dir = pipeline_path
        )

        # Fire event to update footer message
        ravedash::fire_rave_event('loader_message',
                                  "Data loaded")

        # Return TRUE indicating data has been loaded
        TRUE
```

```
        }, error = function(e){

          # Fire event to remove footer message
          ravedash::fire_rave_event('loader_message', NULL)

          # Return FALSE indicating no data has been found
          FALSE
        })
      }, session = session
    )

  }
```

`new_rave_shiny_component_container` *Creates a container for preset components*

Description

Creates a container for preset components

Usage

```
new_rave_shiny_component_container(
  module_id,
  pipeline_name,
  pipeline_path = raveio::pipeline_find(pipeline_name),
  settings_file = "settings.yaml"
)
```

Arguments

`module_id`	'RAVE' module ID
`pipeline_name`	the name of pipeline to run
`pipeline_path`	path of the pipeline
`settings_file`	the settings file of the pipeline, usually stores the pipeline input information; default is `"settings.yaml"`

Value

A `'RAVEShinyComponentContainer'` instance

Examples

```
f <- tempfile()
dir.create(f, showWarnings = FALSE, recursive = TRUE)
file.create(file.path(f, "settings.yaml"))

container <- new_rave_shiny_component_container(
  module_id = "module_power_phase_coherence",
  pipeline_name = "power_phase_coherence_pipeline",
  pipeline_path = f
)

loader_project <- presets_loader_project()
loader_subject <- presets_loader_subject()

container$add_components(
  loader_project, loader_subject
)
```

---

output_gadget *'RAVE' dashboard output gadgets*

Description

'RAVE' dashboard output gadgets

Usage

```
output_gadget(
  outputId,
  icon = NULL,
  type = c("standalone", "download", "download2", "actionbutton", "custom"),
  class = NULL,
  inputId = NULL,
  ...
)

output_gadget_container(
  expr,
  gadgets = c("standalone", "download2"),
  quoted = FALSE,
  env = parent.frame(),
  outputId = NULL,
  class = NULL,
  container = NULL,
  wrapper = TRUE
)
```

Arguments

outputId	output ID in the root scope of shiny session
icon	gadget icon
type, gadgets	gadget type(s), currently supported: 'standalone', 'download', 'actionbutton'
class	additional class to the gadget or its container
inputId	input ID, automatically assigned internally
...	ignored
expr	shiny output call expression, for example, shiny::plotOutput({...})
quoted	whether expr is quoted; default is false
env	environment where expr should be evaluated
container	optional container for the gadgets and outputs; will be ignored if wrapper is false
wrapper	whether to wrap the gadgets and the output within a 'HTML' container

---

plotOutput2 *Shiny plot output with minimum height and additional classes*

Description

Shiny plot output with minimum height and additional classes

Usage

```
plotOutput2(
  outputId,
  class = NULL,
  width = "100%",
  height = "100%",
  min_height = "400px",
  ...
)
```

Arguments

outputId, width, height, ...	passed to plotOutput
class	additional 'HTML' class of the output wrapper
min_height	minimum height of the image; default is 400 pixels

Value

A plot output element that can be included in a panel.

Examples

```
plotOutput2("plot", class = "rounded overflow-hidden",
            min_height = 300)
```

`random-text` *Randomly choose a text from a list of strings*

Description

Randomly choose a text from a list of strings

Usage

```
be_patient_text(candidates)

finished_text(candidates)
```

Arguments

`candidates`	character vectors, a list of candidates

Value

`be_patient_text` returns a text asking users to be patient; `finished_text` returns the text indicating the task has finished.

Examples

```
be_patient_text()

finished_text()
```

`rave-input-output-card`
Input and output card (front-end element)

Description

Input and output card (front-end element)

Usage

```
input_card(
  title,
  ...,
  class = "",
  class_header = "shidashi-anchor",
  class_body = "padding-10",
  class_foot = "padding-10",
  href = "auto",
  tools = NULL,
  footer = NULL,
  append_tools = TRUE,
  toggle_advanced = FALSE,
  module_id = get0("module_id", ifnotfound = NULL, envir = parent.frame())
)

output_card(
  title,
  ...,
  class = "",
  class_body = "padding-10",
  class_foot = "padding-10",
  href = "auto",
  tools = NULL,
  append_tools = TRUE,
  module_id = get0("module_id", ifnotfound = NULL, envir = parent.frame())
)

output_cardset(
  title,
  ...,
  class = "",
  class_body = "no-padding",
  class_foot = "padding-10",
  href = "auto",
  tools = NULL,
  append_tools = TRUE,
  module_id = get0("module_id", ifnotfound = NULL, envir = parent.frame())
)
```

Arguments

`title`	title of the card
`...`	additional elements to be included in the card, see `card`
`class`	the 'HTML' class for card
`class_header`	the 'HTML' class for card header; default is `'shidashi-anchor'`, which will generate shortcuts at the page footers

class_body	the 'HTML' class for card body; default is "padding-10", with '10px' at each direction
class_foot	the 'HTML' class for card footer; default is "padding-10", with '10px' at each direction
href	hyper reference link of the card
tools	a list of additional card tools, see card_tool
footer	footer elements
append_tools	whether to append tools to the default list; default is true
toggle_advanced	whether to show links in the footer to toggle elements with 'HTML' class 'rave-optional'
module_id	the 'RAVE' module ID

Value

'HTML' tags

See Also

card

Examples

```
input_card(title = "Condition selector",
           "Please select experimental conditions:",
           shiny::selectInput(
             inputId = "condition", label = "Condition",
             choices = c("Audio", "Visual")
           ))
```

---

rave-runtime-events *'RAVE' run-time events*

Description

A set of preset behaviors used by 'RAVE' modules

Usage

```
register_rave_session(
  session = shiny::getDefaultReactiveDomain(),
  .rave_id = NULL
)

get_default_handlers(session = shiny::getDefaultReactiveDomain())
```

```
fire_rave_event(
  key,
  value,
  global = FALSE,
  force = FALSE,
  session = shiny::getDefaultReactiveDomain(),
  .internal_ok = FALSE
)

get_session_by_rave_id(rave_id)

get_rave_event(key, session = shiny::getDefaultReactiveDomain())

open_loader(session = shiny::getDefaultReactiveDomain())

close_loader(session = shiny::getDefaultReactiveDomain())

watch_loader_opened(session = shiny::getDefaultReactiveDomain())

watch_data_loaded(session = shiny::getDefaultReactiveDomain())

current_shiny_theme(default, session = shiny::getDefaultReactiveDomain())
```

Arguments

`session`	shiny session, usually automatically determined
`key`	event key to fire or to monitor
`value`	event value
`global`	whether to notify other sessions (experimental and not recommended)
`force`	whether to force firing the event even the `value` hasn't changed
`.internal_ok`	internally used
`rave_id`, `.rave_id`	internally used to store unique session identification key
`default`	default value if not found

Details

These goal of these event functions is to simplify the dashboard logic without understanding the details or passing global variables around. Everything starts with `register_rave_session`. This function registers a unique identification to session, and adds bunch of registry to monitor the changes of themes, built-in, and custom events. If you have called `module_server_common`, then `register_rave_session` has already been called.

`register_rave_session` make initial registries, must be called, returns a list of registries

`fire_rave_event` send signals to make changes to a event; returns nothing

`get_rave_event` watch and get the event values; must run in shiny reactive context

`open_loader` fire an event with a special key `'open_loader'` to open the data-loading panel; returns nothing

`close_loader` reset an event with a special key `'open_loader'` to close the data-loading panel if possible; returns nothing

`watch_loader_opened` watch in shiny reactive context whether the loader is opened; returns a logical value, but raise errors when reactive context is missing

`watch_data_loaded` watch a special event with key `'data_loaded'`; returns a logical value of whether new data has been loaded, or raise errors when reactive context is missing

`current_shiny_theme` watch and returns a list of theme parameters, for example, light or dark theme

Value

See 'Details'

Built-in Events

The following event keys are built-in. Please do not fire them using `fire_rave_event` or the 'RAVE' application might will crash

`'simplify_toggle'` toggle visibility of 'HTML' elements with class `'rave-option'`

`'run_analysis'` notifies the module to run pipeline

`'save_pipeline'`, `'load_pipeline'` notifies the module to save or load pipeline

`'data_loaded'` notifies the module that new data has been loaded

`'open_loader'`, `'toggle_loader'` notifies the internal server code to show or hide the data loading panel

`'active_module'` internally used to store current active module information

Examples

```
library(shiny)
library(ravedash)

ui <- fluidPage(
  actionButton("btn", "Fire event"),
  actionButton("btn2", "Toggle loader")
)

server <- function(input, output, session) {
  # Create event registries
  register_rave_session()

  shiny::bindEvent(
    shiny::observe({
      fire_rave_event("my_event_key", Sys.time())
    }),
    input$btn,
```

```
    ignoreInit = TRUE,
    ignoreNULL = TRUE
  )
  shiny::bindEvent(
    shiny::observe({
      cat("An event fired with value:", get_rave_event("my_event_key"), "\n")
    }),
    get_rave_event("my_event_key"),
    ignoreNULL = TRUE
  )

  shiny::bindEvent(
    shiny::observe({
      if(watch_loader_opened()){
        close_loader()
      } else {
        open_loader()
      }
    }),
    input$btn2,
    ignoreInit = TRUE,
    ignoreNULL = TRUE
  )

  shiny::bindEvent(
    shiny::observe({
      cat("Loader is", ifelse(watch_loader_opened(), "opened", "closed"), "\n")
    }),
    watch_loader_opened(),
    ignoreNULL = TRUE
  )

}

if(interactive()){
  shinyApp(ui, server)
}
```

`rave-session` *Create, register, list, and remove 'RAVE' sessions*

Description

Create, register, list, and remove 'RAVE' sessions

Usage

```
new_session(update = FALSE, app_root = NULL)
```

```
use_session(x, ...)

launch_session(
  x,
  host = "127.0.0.1",
  port = NULL,
  modules = NULL,
  dry_run = FALSE,
 options = list(jupyter = TRUE, jupyter_port = NULL, as_job = TRUE, launch_browser =
    TRUE, single_session = FALSE, page_title = NULL, sidebar_open = TRUE)
)

session_getopt(keys, default = NA, namespace = "default")

session_setopt(..., .list = NULL, namespace = "default")

remove_session(x)

remove_all_sessions()

list_session(path = session_root(), order = c("none", "ascend", "descend"))

start_session(
  session,
  new = NA,
  modules = NULL,
  page_title = NULL,
  sidebar_open = TRUE,
  host = "127.0.0.1",
  port = NULL,
  jupyter = NA,
  jupyter_port = NULL,
  as_job = TRUE,
  launch_browser = TRUE,
  single_session = FALSE,
  app_root = NULL,
  dry_run = FALSE
)

shutdown_session(
  returnValue = invisible(NULL),
  jupyter = TRUE,
  session = shiny::getDefaultReactiveDomain()
)

session_log(x, max_lines = 200, modules = NULL)
```

Arguments

update	logical, whether to update to latest 'RAVE' template
..., .list	named list of key-value pairs of session options. The keys must be characters, and values must be simple data types (such as numeric vectors, characters)
host	host 'IP' address, default is 'localhost'
port	port to listen
modules	selected module ID to launch; used to only show a subset of modules; default is NULL (select all modules); hidden modules are always selected
dry_run	whether to dry-run (do not launch) the 'RAVE' session
options	additional options, including jupyter, jupyter_port, as_job, and launch_browser
keys	vector of characters, one or more keys of which the values should be obtained
default	default value if key is missing
namespace	namespace of the option; default is 'default'
path, app_root	root path to store the sessions; default is the "tensor_temp_path" in raveio_getopt
order	whether to order the session by date created; choices are 'none' (default), 'ascend', 'descend'
session, x	session identification string, or session object; use list_session to list all existing sessions
new	whether to create a new session instead of using the most recent one, default is false
page_title	session web page title and logo text; can have length of either one (page title and logo text are the same); or length of two, with page title be the first element and logo text be the second.
sidebar_open	whether to open the side-bar by default; default TRUE when more than one module is to be displayed
jupyter	logical, whether to launch 'jupyter' instances when starting 'RAVE' sessions, or to stop the 'jupyter' instances when shutting down. It requires additional setups to enable 'jupyter' lab; see 'Installation Guide Step 3' in the 'RAVE' wiki page.
jupyter_port	port used by 'jupyter' lab, can be set by 'jupyter_port' option in raveio_setopt
as_job	whether to launch the application as 'RStudio' job, default is true if 'RStudio' is detected; when running without 'RStudio', this option is always false
launch_browser	whether to launch browser, default is true
single_session	whether to enable single-session mode. Under this mode, closing the main frame will terminate 'RAVE' run-time session, otherwise the 'RAVE' instance will still open in the background
returnValue	passed to stopApp
max_lines	maximum number of log entries to return; default is 200

Value

`new_session` returns a session object with character `'session_id'` and a function `'launch_session'` to launch the application from this session

`use_session` returns a session object, the same as `new_session` under the condition that corresponding session exists, or raise an error if the session is missing

`list_session` returns a list of all existing session objects under the session root

`remove_session` returns a logical whether the corresponding session has been found and removed

Examples

```
if(interactive()){

  sess <- new_session()
  sess$launch_session()

  all_sessions <- list_session()
  print(all_sessions)

  # Use existing session
  session_id <- all_sessions[[1]]$session_id
  sess <- use_session(session_id)
  sess$launch_session()

  # Remove session
  remove_session(session_id)
  list_session()
}
```

---

rave-ui-preset *Preset reusable front-end components for 'RAVE' modules*

Description

For examples and use cases, please check `new_rave_shiny_component_container`.

Usage

```
presets_analysis_electrode_selector2(
  id = "electrode_text",
  varname = "analysis_electrodes",
  label = "Select Electrodes",
  loader_project_id = "loader_project_name",
  loader_subject_id = "loader_subject_code",
  pipeline_repository = "repository",
  start_simple = FALSE,
  multiple = TRUE
```

```
)

presets_analysis_ranges(
  id = "analysis_ranges",
  varname = "analysis_ranges",
  label = "Configure Analysis",
  pipeline_repository = "repository",
  max_components = 2
)

presets_baseline_choices(
  id = "baseline_choices",
  varname = "baseline",
  label = "Baseline Settings",
  pipeline_repository = "repository",
  baseline_choices = c("Decibel", "% Change Power", "% Change Amplitude",
    "z-score Power", "z-score Amplitude"),
 baseline_along_choices = c("Per frequency, trial, and electrode", "Across electrode",
    "Across trial", "Across trial and electrode")
)

presets_condition_groups(
  id = "condition_groups",
  varname = "condition_groups",
  label = "Create Condition Contrast",
  pipeline_repository = "repository"
)

presets_import_export_subject_pipeline(
  id = "im_ex_pipeline",
  loader_project_id = "loader_project_name",
  loader_subject_id = "loader_subject_code",
  pipeline_repository = "repository",
 settings_entries = c("loaded_electrodes", "epoch_choice", "epoch_choice__trial_starts",
    "epoch_choice__trial_ends", "reference_name"),
  fork_mode = c("exclude", "include")
)

presets_import_setup_blocks(
  id = "import_blocks",
  label = "Format & session blocks",
  import_setup_id = "import_setup",
  max_components = 5
)

presets_import_setup_channels(
  id = "import_channels",
  label = "Channel information",
```

```
  import_setup_id = "import_setup",
  import_blocks_id = "import_blocks"
)

presets_import_setup_native(
  id = "import_setup",
  label = "Select project & subject"
)

presets_loader_3dviewer(
  id = "loader_3d_viewer",
  height = "600px",
  loader_project_id = "loader_project_name",
  loader_subject_id = "loader_subject_code",
  loader_reference_id = "loader_reference_name",
  loader_electrodes_id = "loader_electrode_text",
  gadgets = c("standalone", "download")
)

presets_loader_3dviewer2(
  id = "loader_3d_viewer",
  height = "600px",
  loader_project_id = "loader_project_name",
  loader_subject_id = "loader_subject_code",
  loader_electrodes_id = "loader_electrode_text",
  gadgets = c("standalone", "download")
)

presets_loader_electrodes(
  id = "loader_electrode_text",
  varname = "loaded_electrodes",
  label = "Electrodes",
  loader_project_id = "loader_project_name",
  loader_subject_id = "loader_subject_code"
)

presets_loader_epoch(
  id = "loader_epoch_name",
  varname = "epoch_choice",
  label = "Epoch and Trial Duration",
  loader_project_id = "loader_project_name",
  loader_subject_id = "loader_subject_code"
)

presets_loader_project(
  id = "loader_project_name",
  varname = "project_name",
  label = "Project"
```

```
)

presets_loader_reference(
  id = "loader_reference_name",
  varname = "reference_name",
  label = "Reference name",
  loader_project_id = "loader_project_name",
  loader_subject_id = "loader_subject_code",
  mode = c("default", "create")
)

presets_loader_subject(
  id = "loader_subject_code",
  varname = "subject_code",
  label = "Subject",
  loader_project_id = "loader_project_name",
  checks = c("notch", "wavelet"),
  allow_new = FALSE
)

presets_loader_subject_only(
  id = "loader_subject_code",
  varname = "subject_code",
  label = "Subject",
  multiple = FALSE
)

presets_loader_sync_project_subject(
  id = "loader_sync_project_subject",
  label = "Sync subject from most recently loaded",
  varname = "loader_sync_project_subject",
  loader_project_id = "loader_project_name",
  loader_subject_id = "loader_subject_code",
  from_module = NULL,
  project_varname = "project_name",
  subject_varname = "subject_code"
)
```

Arguments

`id`	input or output ID of the element; this ID will be prepended with module namespace
`varname`	variable name(s) in the module's settings file
`label`	readable label(s) of the element
`loader_project_id`	the ID of `presets_loader_project` if different to the default
`loader_subject_id`	the ID of `presets_loader_subject` if different to the default

`pipeline_repository`
: the pipeline name that represents the 'RAVE' repository from functions such as `prepare_subject_bare`, `prepare_subject_with_epoch`, and `prepare_subject_power`

`start_simple`
: whether to start in simple view and hide optional inputs

`multiple`
: whether to allow multiple inputs

`max_components`
: maximum number of components for compound inputs

`baseline_choices`
: the possible approaches to calculate baseline

`baseline_along_choices`
: the units of baseline

`settings_entries`
: used when importing pipelines, pipeline variable names to be included or excluded, depending on `fork_mode`

`fork_mode`
: `'exclude'` (default) or `'include'`; in `'exclude'` mode, `settings_entries` will be excluded from the pipeline settings; in `'include'` mode, only `settings_entries` can be imported.

`import_setup_id`
: the ID of `presets_import_setup_native` if different to the default

`import_blocks_id`
: the ID of `presets_import_setup_blocks` if different to the default

`height`
: height of the element

`loader_reference_id`
: the ID of `presets_loader_reference` if different to the default

`loader_electrodes_id`
: the ID of `presets_loader_electrodes` if different to the default

`gadgets`
: gadget types to include; see `type` argument in function `output_gadget`

`mode`
: whether to create new reference, or simply to choose from existing references

`checks`
: whether to check if subject has been applied with 'Notch' filters or 'Wavelet'; default is both.

`allow_new`
: whether to allow new subject to be created; ignored when checks exist

`from_module`
: which module to extract input settings

`project_varname, subject_varname`
: variable names that should be extracted from the settings file

Value

A `'RAVEShinyComponent'` instance.

See Also

`new_rave_shiny_component_container`

ravedash_footer *A hovering footer at bottom-right*

Description

Internally used. Do not call explicitly

Usage

```
ravedash_footer(
  module_id = NULL,
  label = "Run Analysis",
  auto_recalculation = TRUE,
  message_action = "toggle_loader",
  class = NULL,
  style = NULL
)
```

Arguments

module_id	'RAVE' module ID
label	run-analysis button label; default is "Run Analysis"
auto_recalculation	whether to show the automatic calculation button; default is true
message_action	message to send when clicking on message button; default is 'toggle_loader', which opens up loading screen
class	additional class for the footer
style	additional style for the footer

Value

'HTML' tags

Examples

```
library(shiny)
# dummy variables for the example
data_loaded <- TRUE

# UI code
ravedash_footer("my_module")

# server code to set message
server <- function(input, output, session){

  module_server_common(input, output, session, function(){
```

```
    # check if data has been loaded
    if(data_loaded) {

      # if yes, then set the footer message
      fire_rave_event("loader_message",
                      "my_project/subject - Epoch: Auditory")
      return(TRUE)
    } else {

      # No data found, unset the footer message
      fire_rave_event("loader_message", NULL)
      return(FALSE)
    }

  })
}
```

---

register_output *Register output and output options*

Description

Enable advanced output gadgets such as expanding the output in another browser window, or downloading the rendered data.

Usage

```
register_output_options(
  outputId,
  ...,
  .opt = list(),
  extras = list(),
  session = shiny::getDefaultReactiveDomain()
)

get_output_options(outputId, session = shiny::getDefaultReactiveDomain())

register_output(
  render_function,
  outputId,
  ...,
  output_opts = list(),
  quoted = FALSE,
  download_function = NULL,
  download_fileformat =
    "{ outputId }-{ format(Sys.time(), '%b_%d_%Y_%H_%M_%S') }.{ extension }",
  output_type = c("image", "data", "threeBrain", "no-download"),
```

```
  extensions = NULL,
  title = "Download widget",
  cancel_btn = "Cancel",
  confirm_btn = "Download",
  session = shiny::getDefaultReactiveDomain()
)

get_output(outputId, session = shiny::getDefaultReactiveDomain())
```

Arguments

`outputId`	output ID in the scope of current shiny session
`extras`	extra information to store
`session`	shiny session instance
`render_function`	shiny render function
`output_opts`, `.opt`	output options
`quoted`	whether `render_function` is quoted; default is false
`download_function`	core function that writes the data into the files; default is set for `'image'` and `'threeBrain'` automatically; see 'Default' and 'Examples'.
`download_fileformat`	download file format, supports 'glue'
`output_type`	type of export file formats supported, options are `'image'` (for figures, default), `'data'`, `'threeBrain'` (for 'RAVE' 3D viewers), and `'no-download'` (do not export).
`extensions`	a list of file extensions and their descriptions; the names will be used to display the modal selectors, and values are the underlying extension read by `download_fileformat`
`title`, `cancel_btn`, `confirm_btn`, `...`	title, button labels, and additional 'HTML' elements that are to be shown in the modal

Details

The following steps are done when `register_output` is called:

* Register the render function to shiny output `output[[outputId]]` * Register the render information to session which can be retrieved via `get_output` * Register (if `download_function` is a function) a download handler that listen to the shiny event. The event ID is `paste0(outputId, '__download2')`.

When downloading event is triggered, a modal will pop up asking for exporting format (always exists) and image dimensions (if output type is `'image'`) or title (when output type is `'threeBrain'`). Users will choose the proper inputs, which will be passed into `download_function`.

The file `extensions` is a named list. Its names are printable descriptions of the formats, and values are the file extensions (without the leading `'.'`). for example, `list("compressed CSV" = "csv")`. Users will see `"compressed CSV"` in the format selector, and `download_function` sees `"csv"`.

When output type is image, users will be asked to enter the image size in inches; default width is 7, and height is calculated based on current image aspect ratio.

If you would like to show more on the modal, pass 'HTML' elements to ...

Function `download_function` is a function containing four inputs:

* `con`: file where the data should be written into * `params`: a named list of `params$extension` (file extension), `width`, `height` (type is image), or `title` (3D viewer) * `render_expr` a quoted rendering expression of the rendering function * `render_env` the rendering environment of the rendering function.

Default `download_function` is provided when not specified.

Value

Registered output or output options.

Examples

```
if(interactive()) {

library(shiny)
library(ravedash)

# ---- Use this in RAVE --------

# UI
output_gadget_container(
  plotOutput("plot", brush = shiny::brushOpts("plot__brush")),
)

# server
server <- function(input, output, session) {
  register_output(
    renderPlot({
      # ... plot it
    }),
    outputId = "plot",
    output_opts = list(brush = shiny::brushOpts("plot__brush"))
  )
}


# ---- Low-level method ------------

rave_id <- paste(sample(c(letters, LETTERS, 0:9), 20, replace = TRUE),
                 collapse = "")

ui <- function(req) {
  query_string <- req$QUERY_STRING
  if(length(query_string) != 1) {
    query_string <- "/"
  }
```

```
  query_result <- httr::parse_url(query_string)

  if(!identical(toupper(query_result$query$standalone), "TRUE")) {
    # normal page
    basicPage(
      output_gadget_container(
        plotOutput("plot", brush = shiny::brushOpts("plot__brush")),
      )
    )
  } else {
    # standalone viewer
    uiOutput("viewer")
  }
}

server <- function(input, output, session) {

  bindEvent(
    safe_observe({
      query_string <- session$clientData$url_search
      query_result <- httr::parse_url(query_string)

      if(!identical(toupper(query_result$query$module), "standalone_viewer")) {
        # normal page
        register_rave_session(session = session, .rave_id = rave_id)
        register_output(
          renderPlot({
            input$btn
            plot(rnorm(100), pch = 16)
          }),
          outputId = "plot",
          output_opts = list(brush = shiny::brushOpts("plot__brush"))
        )
      } else {
        # standalone viewer
        standalone_viewer(outputId = "plot", rave_id = rave_id)
      }
    }),
    session$clientData$url_search
  )


}

shinyApp(ui, server, options = list(port = 8989))
}
```

`run_analysis_button` *Button to trigger analysis*

Description

A button that triggers 'run_analysis' event; see also get_rave_event

Usage

```
run_analysis_button(
  label = "Run analysis (Ctrl+Enter)",
  icon = NULL,
  width = NULL,
  type = "primary",
  btn_type = c("button", "link"),
  class = "",
  style = "",
  ...
)
```

Arguments

label	label to display
icon	icon before the label
width, class, style, ...	passed to 'HTML' tag
type	used to calculate class
btn_type	button style, choices are 'button' or 'link'

Value

A 'HTML' button tag

---

safe_observe *Safe-wrapper of 'shiny' observe function*

Description

Safely wrap expression x such that shiny application does no hang when when the expression raises error.

Usage

```
safe_observe(
  x,
  env = NULL,
  quoted = FALSE,
  priority = 0L,
  domain = NULL,
  ...,
```

```
  error_wrapper = c("none", "notification", "alert"),
  watch_data = getOption("ravedash.auto_watch_data", FALSE)
)
```

Arguments

x, env, quoted, priority, domain, ...	passed to observe
error_wrapper	handler when error is encountered, choices are 'none', 'notification' (see error_notification), or 'alert' (see error_alert)
watch_data	whether to invalidate only when watch_data_loaded is TRUE

Value

'shiny' observer instance

Examples

```
values <- shiny::reactiveValues(A=1)

obsB <- safe_observe({
  print(values$A + 1)
})
```

---

shiny_cache *Obtain caching object for current run-time shiny session*

Description

Cache small objects such as inputs or configurations

Usage

```
shiny_cache(namespace, session = shiny::getDefaultReactiveDomain())
```

Arguments

namespace	characters, usually the module ID
session	shiny interactive context domain

Value

A caching object. The caching object is identical within the same context and namespace.

shiny_check_input *Check shiny inputs and modify if validation fails*

Description

Check shiny inputs and modify if validation fails

Usage

```
shiny_check_input(
  inputId,
  check = NULL,
  on_check_fails,
  ...,
  quoted = FALSE,
  env = parent.frame(),
  logger_level = c("trace", "none", "debug", "info", "warning", "error"),
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

inputId	character, input ID
check	either a function that takes the input value or a character of a checkmate function; when check is a character, this function will look for check_* functions in the checkmate package
on_check_fails	value to substitute when check fails, and the input value will be the result of on_check_fails. This argument can be missing; when missing, input value will not be altered
...	passed to check function
quoted	whether on_check_fails is quoted
env	environment to evaluate on_check_fails
logger_level	log level when validation fails
session	shiny session; default is current session

Value

A shiny observe instance

Examples

```
if(interactive()) {

library(ravedash)
shiny::shinyApp(
  ui = shiny::basicPage(
```

```
    shiny::textInput("id1", "Enter a text"),
    shiny::textOutput("id2")
  ),
  server = function(input, output, session) {
    # input$id1 must have at least 1 character
    # the check uses `checkmate::check_character`
    shiny_check_input(
      "id1",
      check = "character",
      min.chars = 1,
      on_check_fails = "altered text"
    )

    output$id2 <- shiny::renderText({
      print(input$id1)
      sprintf("The final value is: %s", input$id1)
    })
  }
)

}
```

`shiny_icons` *Shiny icons*

Description

Shiny icons

Usage

```
shiny_icons
```

Format

An object of class `ravedash_shiny_icons` of length 0.

Details

The goal of create this list is to keep 'shiny' icons (which are essentially 'font-awesome' icons) up-to-date.

`simple_layout` *Simple input-output layout*

Description

Provides simple layout, with inputs on the left, and outputs on the right. Only useful in `'shidashi'` framework.

Usage

```
simple_layout(
  input_ui,
  output_ui,
  input_width = 4L,
  container_fixed = FALSE,
  container_style = NULL,
  scroll = FALSE
)
```

Arguments

`input_ui`	the 'HTML' tags for the inputs
`output_ui`	the 'HTML' tags for the outputs
`input_width`	width of inputs, must be an integer from 1 to 11
`container_fixed`	whether the maximum width of the container should be fixed; default is no
`container_style`	additional 'CSS' style of the container
`scroll`	whether to stretch the container to full-heights and scroll the input and output separately.

Value

'HTML' tags

Examples

```
library(shiny)
library(ravedash)

simple_layout(
  input_ui = list(
    ravedash::input_card(
      title = "Data Selection",
      "Add inputs here"
    )
  ),
```

```
  output_ui = list(
    ravedash::output_card(
      title = "Result A",
      "Add outputs here"
    )
  )
)
```

`standalone_viewer` *Register shiny-output options to allow display in stand-alone viewers*

Description

Save the output options such that the additional configurations can be used by stand-alone viewer

Usage

```
standalone_viewer(
  outputId,
  module_session,
  rave_id,
  session = shiny::getDefaultReactiveDomain(),
  wrapper_id = "viewer"
)
```

Arguments

`outputId`	the full shiny output ID
`module_session`	the module shiny session; if not provided, then the session will be inferred by `rave_id`
`rave_id`	the unique identification key for 'RAVE' module sessions, can be obtained via `get_active_module_info`
`session`	shiny session object
`wrapper_id`	the wrapping render ID, default is `"viewer"`

Details

'RAVE' dashboard provides powerful stand-alone viewers where users can display almost any outputs from other modules and interact with these viewers while sending messages back.

Value

nothing

Examples

```
if(interactive()) {

library(shiny)
library(ravedash)

rave_id <- paste(sample(c(letters, LETTERS, 0:9), 20, replace = TRUE),
                 collapse = "")

ui <- function(req) {
  query_string <- req$QUERY_STRING
  if(length(query_string) != 1) {
    query_string <- "/"
  }
  query_result <- httr::parse_url(query_string)

  if(!identical(toupper(query_result$query$standalone), "TRUE")) {
    # normal page
    basicPage(
      actionButton("btn", "Click Me"),
      plotOutput("plot")
    )
  } else {
    # standalone viewer
    uiOutput("viewer")
  }
}

server <- function(input, output, session) {

  bindEvent(
    safe_observe({
      query_string <- session$clientData$url_search
      query_result <- httr::parse_url(query_string)

      if(!identical(toupper(query_result$query$standalone), "TRUE")) {
        # normal page
        register_rave_session(session = session, .rave_id = rave_id)
        output$plot <- renderPlot({
          input$btn
          plot(rnorm(100), pch = 16)
        })
      } else {
        # standalone viewer
        standalone_viewer(outputId = "plot", rave_id = rave_id)
      }
    }),
    session$clientData$url_search
  )


}
```

```
shinyApp(ui, server, options = list(port = 8989))

# Now open http://127.0.0.1:8989/?standalone=TRUE

}
```

`switch_module` *Drive 'RAVE' browser to switch to another module*

Description

Switch to another 'RAVE' module to continue the procedures.

Usage

```
switch_module(module_id, title, session = shiny::getDefaultReactiveDomain())
```

Arguments

`module_id`	the module ID, see `'modules.yaml'` in the pipeline directory
`title`	the module title to display
`session`	shiny session

Value

Nothing

`temp_file` *Create a random temporary file path for current session*

Description

Create a random temporary file path for current session

Usage

```
temp_file(
  pattern = "file",
  fileext = "",
  persist = c("process", "app-session", "package-cache")
)

temp_dir(check = FALSE, persist = c("process", "app-session", "package-cache"))
```

Arguments

pattern, fileext	see tempfile
persist	persist level, choices are 'app-session', 'package-cache', and 'process'; see 'Details'. 'RAVE' application session, default), 'package-cache' (package-level cache directory)
check	whether to create the temporary directory

Details

R default tempdir usually gets removed once the R process ends. This behavior might not meet all the needs for 'RAVE' modules. For example, some data are 'RAVE' session-based, like current or last visited subject, project, or state data (like bookmarks, configurations). This session-based information will be useful when launching the same 'RAVE' instance next time, hence should not be removed when users close R. Other data, such as subject-related, or package-related should last even longer. These types of data may be cache of subject power, package-generated color schemes, often irrelevant from R or 'RAVE' sessions, and can be shared across different 'RAVE' instances.

The default scheme is persist='process'. Under this mode, this function behaves the same as tempfile. To store data in 'RAVE' session-based manner, please use persist='app-session'. The actual path will be inside of 'RAVE' session folder, hence this option is valid only if 'RAVE' instance is running. When 'RAVE' instance is not running, the result falls back to persist='process'. When persist='process', To cache larger and session-irrelevant data, use 'package-cache'.

The 'RAVE' session and package cache are not cleared even when R process ends. Users need to clean the data by themselves. See remove_session or remove_all_sessions about removing session-based folders, or clear_cached_files to remove package-based cache.

Value

A file or a directory path to persist temporary data cache

Examples

```
temp_dir()
temp_dir(persist = "package-cache")
```

---

with_log_modal *Evaluate script in the background and show the results from shiny modal dialogue*

Description

Evaluate script in the background and show the results from shiny modal dialogue

Usage

```
with_log_modal(
  expr,
  quoted = FALSE,
  callback = NULL,
  title = "Running...",
  size = "l",
  session = shiny::getDefaultReactiveDomain(),
  ...
)
```

Arguments

expr	R expression to evaluate The script must be standalone
quoted	whether the expression has been quoted
callback	callback function to run once the evaluate finishes; must take one argument. The passed variable will be the evaluation results or an error condition (if error occurs)
title, size	modal title and size, see showModal
session	shiny session object
...	ignored, reserved for future use

Value

A promise object

Examples

```
# Shiny server function
server <- function(input, output, session) {
  shiny::bindEvent(
    shiny::observe({
      with_log_modal(
        title = "Roll the dice",
        expr = {
          for(i in 1:10) {
            Sys.sleep(runif(1, min = 0.5, max = 2))
            cat(sprintf("Rolling dice result: %.0f\n", sample(6, 1)))
          }
        }
      )
      return()
    }),
    input$btn,
    ignoreNULL = TRUE, ignoreInit = TRUE
  )
}

if(interactive()) {
```

```
  shiny::shinyApp(
    ui = shiny::basicPage(
      shiny::actionButton('btn', "Click me")
    ),
    server = server,
    options = list(launch.browser = TRUE)
  )
}
```

Index