Package 'ravedash'

June 23, 2022

Type Package

Title Dashboard System for Reproducible Visualization of 'iEEG'

Version 0.1.1

Description Dashboard system to display the analysis results produced by 'RAVE' (Magnotti J.F., Wang Z., Beauchamp M.S. (2020), R 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 dipsaus (>= 0.2.0), logger (>= 0.2.2), raveio (>= 0.0.5.9000), rpymat (>= 0.1.2), shidashi (>= 0.1.0.9000), shiny (>= 1.7.1), shinyWidgets (>= 0.6.2), threeBrain (>= 0.2.4), shinyvalidate

Suggests fastmap (>= 1.1.0), rlang (>= 1.0.2), crayon (>= 1.4.2), rstudioapi, knitr, rmarkdown

RoxygenNote 7.2.0

URL https://dipterix.org/ravedash/

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

VignetteBuilder knitr

NeedsCompilation no

Author Zhengjia Wang [aut, cre, cph]

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

Repository CRAN

Date/Publication 2022-06-23 07:30:02 UTC

R topics documented:

`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

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")
```

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

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
)
```

`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())
)
```

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_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
.rave_id	internally used to store unique session identification
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
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)

use_session(x)

launch_session(
  x,
  host = "127.0.0.1",
  port = NULL,
 options = list(jupyter = TRUE, jupyter_port = NULL, as_job = TRUE, launch_browser =
    TRUE)
)

remove_session(x)

remove_all_sessions()

list_session(path = session_root())
```

Arguments

update	logical, whether to update to latest 'RAVE' template
x	session identification string, or session object; use `list_session` to list all existing sessions
host, port, options	configurations needed to launch the session
path	root path to store the sessions; default is the "tensor_temp_path" in raveio_getopt

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"
)

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"
)

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")
)
```

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
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
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.

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
)
```

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

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)
    }

  })
}
```

`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 = "button",
  class = "",
  ...
)
```

Arguments

`label`	label to display
`icon`	icon before the label
`width, btn_type, class, ...`	passed to 'HTML' `button` tag
`type`	used to calculate `class`

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, ...)
```

Arguments

x, env, quoted, priority, domain, ...	passed to observe

Value

'shiny' observer instance

Examples

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

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

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"
    )
  )
)
```

Index