Package ‘shinyCohortBuilder’

September 27, 2024

Type Package

Title Modular Cohort-Building Framework for Analytical Dashboards

Version 0.3.0

Maintainer Krystian Igras <krystian8207@gmail.com>

Description You can easily add advanced cohort-building component to your analytical dashboard or simple 'Shiny' app. Then you can instantly start building cohorts using multiple filters of different types, filtering datasets, and filtering steps. Filters can be complex and data-specific, and together with multiple filtering steps you can use complex filtering rules. The cohort-building sidebar panel allows you to easily work with filters, add and remove filtering steps. It helps you with handling missing values during filtering, and provides instant filtering feedback with filter feedback plots. The GUI panel is not only compatible with native shiny bookmarking, but also provides reproducible R code.

License MIT + file LICENSE

Depends R (>= 3.5.0)

Imports magrittr, glue, bslib, jsonlite, purrr, ggplot2, ggiraph, htmltools, shiny (>= 1.7.0), shinyWidgets (>= 0.7.0), htmlwidgets, dplyr, cohortBuilder (>= 0.3.0), tryCatchLog, highr, shinyGizmo (>= 0.4.2), rlang (>= 1.0), tibble, lifecycle

Suggests queryBuilder (>= 0.1.0), shinyQueryBuilder (>= 0.1.0), pkgload, packer, sass, knitr, rmarkdown, testthat (>= 3.0.0)

Encoding UTF-8

RoxygenNote 7.3.1

VignetteBuilder knitr

URL https://r-world-devs.github.io/shinyCohortBuilder/, https://github.com/r-world-devs/shinyCohortBuilder

BugReports https://github.com/r-world-devs/shinyCohortBuilder/issues

Collate 'shinyCohortBuilder-package.R' 'ui_utils.R' 'control_utils.R' 'actions.R' 'renders.R' 'filter_discrete.R' 'filter_range.R' 'filter_date_range.R' 'filter_discrete_text.R' 'filter_multi_discrete.R' 'filter_query.R' 'app.R' 'cb_layer.R' 'source.R' 'source_tblist.R'

Config/testthat/edition 3

NeedsCompilation no

Author Krystian Igras [cre, aut],
Kamil Wais [aut],
Adam Foryś [ctb]

Repository CRAN

Date/Publication 2024-09-27 15:10:03 UTC

Contents

shinyCohortBuilder-package . . . 3
.cb_input . . . 3
.render_filter . . . 4
.save_observer . . . 6
.sendOutput . . . 7
autofilter . . . 8
available-filters-choices . . . 9
cb_changed . . . 10
cb_ui . . . 11
demo_app . . . 13
gui . . . 15
gui-filter-layer . . . 16
keep_na_input . . . 18
pre_post_stats . . . 19
rendering-custom-attrition . . . 20
rendering-filters . . . 22
rendering-step-attrition . . . 24
scb_chart_palette . . . 25
scb_icons . . . 26
scb_labels . . . 26
source-gui-layer . . . 27
trigger-action . . . 27
updating-data-statistics . . . 29

Index **32**

shinyCohortBuilder-package

GUI layer for cohortBuilder package

Description

GUI layer for cohortBuilder package

.cb_input

Create input controller insensitive to server updates

Description

Input controllers created with ‘.cb_input‘ are sending its value to server only when user changes it’s value directly in browser. That means all the ‘update*‘ functions have only visible effect on application output.

The method should be used for each filter input controller and precise which filter value should be updated when the input selection is changes.

Usage

```
.cb_input(ui, data_param, ..., priority = NULL)
```

Arguments

ui	UI defining input controllers.
data_param	Name of the parameter that should be updated in filter whenever user change the input value.
...	Extra attributes passed to the input div container.
priority	Set to ’event’ to force sending value.

Value

A ‘shiny.tag‘ object defining html structure of filter input container.

Examples

```
if (interactive()) {
  library(shiny)
  library(shinyCohortBuilder)

  shiny::addResourcePath(
    "shinyCohortBuilder",
    system.file("www", package = "shinyCohortBuilder")
  )
```

```
  ui <- fluidPage(
    tags$head(
    shiny::tags$script(type = "text/javascript", src = file.path("shinyCohortBuilder", "scb.js"))
    ),
    actionButton("update", "Update with random value"),
    div(
      class = "cb_container",
      `data-ns_prefix` = "",
      div(
        class = "cb_step",
        `data-step_id` = "1",
        div(
          class = "cb_filter",
          `data-filter_id` = "filid",
          .cb_input(
            numericInput("val", "Value", value = 1),
            data_param = "range"
          )
        )
      )
    )
  )

  server <- function(input, output, session) {
    observeEvent(input$action, {
      # print should be avoided when value is changed due to update
      print(input$action)
    })
    observeEvent(input$update, {
      updateNumericInput(session, "val", value = rnorm(1))
    })
  }

  shinyApp(ui, server)
}
```

`.render_filter` *Define filter related output in filtering panel*

Description

The method exported only for custom extensions use.

Usage

```
.render_filter(filter, step_id, cohort, ns)
```

Arguments

`filter`	Filter object.
`step_id`	Id of the step.
`cohort`	Cohort object.
`ns`	Namespace function.

Value

A ‘shiny.tag‘ class ‘div‘ object defining html structure of filter input panel.

Examples

```
if (interactive()) {
  library(magrittr)
  library(shiny)
  library(cohortBuilder)
  library(shinyCohortBuilder)

  ui <- fluidPage(
    actionButton("add_filter", "Add Filter"),
    div(id = "filter_container")
  )

  server <- function(input, output, session) {
    add_gui_filter_layer <- function(public, private, ...) {
      private$steps[["1"]]$filters$copies$gui <- .gui_filter(
        private$steps[["1"]]$filters$copies
      )
    }
    add_hook("post_cohort_hook", add_gui_filter_layer)
    coh <- cohort(
      set_source(as.tblist(librarian)),
      filter(
        "range", id = "copies", name = "Copies", dataset = "books",
        variable = "copies", range = c(5, 12)
      )
    ) %>% run()
    coh$attributes$session <- session
    coh$attributes$feedback <- TRUE

    observeEvent(input$add_filter, {
      insertUI(
        "#filter_container",
        ui = .render_filter(
          coh$get_filter("1", "copies"),
          step_id = "1",
          cohort = coh,
          ns = function(x) x
        ))
    }, ignoreInit = TRUE, once = TRUE)
  }
```

```
  shinyApp(ui, server)
}
```

`.save_observer` *Save observer to user session*

Description

The method used to store created observers (used to implement extra filter logic). The saved observer are then destroyed when filtering step is removed which prevents duplicated execution of accumulated observers.

Usage

```
.save_observer(observer, id, session)
```

Arguments

`observer`	An ‘observe‘ or ‘observeEvent‘ to be saved.
`id`	Id of the observer. Preferably prefixed with step_id. The saved observer is saved as ‘session$userData$observers[['<id>-observer']]‘ object.
`session`	Shiny session object.

Value

No return value, used for side effect which is saving the observer to ‘session$userData‘ object.

Examples

```
if (interactive()) {
  library(shiny)
  library(shinyCohortBuilder)

  ui <- fluidPage(
    numericInput("power", "Power", min = 0, max = 10, value = 1, step = 1),
    numericInput("value", "Value", min = 0, max = 100, value = 2, step = 0.1),
    actionButton("add", "Observe the selected power"),
    actionButton("rm", "Stop observing the selected power")
  )

  server <- function(input, output, session) {
    observeEvent(input$add, {
      .save_observer(
        observeEvent(input$value, {
           print(input$value ^ input$power)
        }),
        as.character(input$power),
        session = session
```

```
      )
    }, ignoreInit = TRUE)

    observeEvent(input$rm, {
      id <- paste0(input$power, "-observer")
      session$userData$observers[[id]]$destroy()
      session$userData$observers[[id]] <- NULL
    }, ignoreInit = TRUE)
  }

  shinyApp(ui, server)
}
```

`.sendOutput` *Send output rendering*

Description

Functional approach to rendering output. Equivalent of ‘output[[name]] <- rendering‘.

Usage

```
.sendOutput(name, rendering, session = shiny::getDefaultReactiveDomain())
```

Arguments

`name`	Name of the output to be rendered
`rendering`	Rendering expression to be sent.
`session`	Shiny session object.

Value

No return value, used for side effect which is assigning rendering to the output object.

Examples

```
if (interactive()) {
  library(shiny)
  library(shinyCohortBuilder)

  rendering <- function(x_max) {
    renderPlot({
      x <- seq(0, x_max, by = 0.01)
      plot(x, sin(x), type = "l")
    })
  }

  ui <- fluidPage(
    numericInput("xmax", "X Axis Limit", min = 0, max = 10, value = pi),
```

```
    plotOutput("out")
  )

  server <- function(input, output, session) {
    observeEvent(input$xmax, {
      .sendOutput("out", rendering(input$xmax))
    })
  }

  shinyApp(ui, server)
}
```

`autofilter` *Generate filters definition based on the Source data*

Description

The method should analyze source data structure, generate proper filters based on the data (e.g. column types) and attach them to source.

Usage

```
autofilter(source, attach_as = c("step", "meta"), ...)

## Default S3 method:
autofilter(source, ...)

## S3 method for class 'tblist'
autofilter(source, attach_as = c("step", "meta"), ...)
```

Arguments

`source`	Source object.
`attach_as`	Choose whether the filters should be attached as a new step, or list of available filters (used in filtering panel when ‘new_step = "configure"‘). By default in `step`.
`...`	Extra arguments passed to a specific method.

Value

Source object having step configuration attached.

See Also

source-gui-layer

Examples

```
library(magrittr)
library(cohortBuilder)
library(shinyCohortBuilder)

iris_source <- set_source(tblist(iris = iris)) %>%
  autofilter()
iris_cohort <- cohort(iris_source)
sum_up(iris_cohort)

if (interactive()) {
  library(shiny)

  ui <- fluidPage(
    cb_ui("mycoh")
  )

  server <- function(input, output, session) {
    cb_server("mycoh", cohort = iris_cohort)
  }

  shinyApp(ui, server)
}
```

---

available-filters-choices

Generate available filters choices based on the Source data

Description

The method should return the available choices for virtualSelect input.

Usage

```
.available_filters_choices(source, cohort, ...)

## Default S3 method:
.available_filters_choices(source, cohort, ...)

## S3 method for class 'tblist'
.available_filters_choices(source, cohort, ...)
```

Arguments

source	Source object.
cohort	cohortBuilder cohort object
...	Extra arguments passed to a specific method.

Value

‘shinyWidgets::prepare_choices‘ output value.

Examples

```
if (interactive()) {
  library(magrittr)
  library(shiny)
  library(cohortBuilder)
  library(shinyCohortBuilder)
  library(shinyWidgets)

  coh <- cohort(
    set_source(as.tblist(librarian), available_filters = list(
      filter(
        "range", id = "copies", name = "Copies", dataset = "books",
        variable = "copies", range = c(5, 12)
      ),
      filter(
        "date_range", id = "registered", name = "Registered",  dataset = "borrowers",
        variable = "registered", range = c(as.Date("2010-01-01"), Inf)
      )
    ))
  ) %>% run()
  filter_choices <- .available_filters_choices(coh$get_source(), coh)

  ui <- fluidPage(
    virtualSelectInput("filters", "Filters", choices = filter_choices, html = TRUE)
  )

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

  }

  shinyApp(ui, server)
}
```

---

cb_changed *Track changes of cohort data in Shiny*

Description

The function returns Shiny input object related to selected cohort that is triggered whenever cohort data filters were applied to it within filtering panel.

Usage

```
cb_changed(session, cohort_id)
```

Arguments

session	Shiny session object.
cohort_id	Id of the cohort.

Details

The function is meant to be used as a trigger for Shiny render functions and observers.

---

cb_ui *Include filtering panel in Shiny*

Description

The function returns filtering panel placeholder, you may use in you custom Shiny application. Use in the UI part of your application.

Usage

```
cb_ui(
  id,
  ...,
  state = FALSE,
  steps = TRUE,
  code = TRUE,
  attrition = TRUE,
  new_step = c("clone", "configure")
)

cb_server(
  id,
  cohort,
  run_button = "none",
  stats = c("pre", "post"),
  feedback = FALSE,
 enable_bookmarking = shiny::getShinyOption("bookmarkStore", default = "disable"),
  show_help = TRUE,
  ...
)
```

Arguments

id	Id of the module used to render the panel.
...	Extra attributes passed to the panel div container.
state	Set to TRUE (default) to enable get/set state panel.
steps	Set to TRUE (default) if multiple steps should be available.

code	Set to TRUE (default) to enable reproducible code panel.
attrition	Set to TRUE (default) to enable attrition plot panel.
new_step	Choose which add step method should be used for creating new step. Possible options are: "clone" - copy filters from last step, "configure" - opening modal and allow to chose filters from available filters.
cohort	Cohort object storing filtering steps configuration.
run_button	Should Run button be displayed? If so, the current step computations are run only when clicked. Three options are available "none" - no button, "local" - button displayed at each step panel, "global" - button visible in top filtering panel.
stats	Choose which statistics should be displayed for data (and some filters). Possible options are: "pre" - previous step stat, "post" - current step stats, ‘c("pre", "post")‘ - for both and NULL for no stats.
feedback	Set to TRUE (default) if feedback plots should be displayed at each filter.
enable_bookmarking	Set to TRUE (default) if panel should be compatible with native shiny bookmarking.
show_help	Set to TRUE (default) to enable help buttons.

Value

Nested list of ‘shiny.tag‘ objects - html structure of filtering panel module.

‘shiny::moduleServer‘ output providing server logic for filtering panel module.

Examples

```
if (interactive()) {
  library(cohortBuilder)
  library(shiny)
  library(shinyCohortBuilder)

  librarian_source <- set_source(as.tblist(librarian))
  librarian_cohort <- cohort(
    librarian_source,
    filter(
      "discrete", id = "author", dataset = "books",
      variable = "author", value = "Dan Brown",
      active = FALSE
    ),
    filter(
      "range", id = "copies", dataset = "books",
      variable = "copies", range = c(5, 10),
      active = FALSE
    ),
    filter(
      "date_range", id = "registered", dataset = "borrowers",
      variable = "registered", range = c(as.Date("2010-01-01"), Inf),
      active = FALSE
```

```
    )
  )

  ui <- fluidPage(
    sidebarLayout(
      sidebarPanel(
        cb_ui("librarian")
      ),
      mainPanel()
    )
  )

  server <- function(input, output, session) {
    cb_server("librarian", librarian_cohort)
  }

  shinyApp(ui, server)
}
```

---

demo_app *Run demo application*

Description

The demo presents available filters and toolbox features.

Usage

```
demo_app(
  steps = TRUE,
  stats = c("pre", "post"),
  run_button = "none",
  feedback = TRUE,
  state = TRUE,
  bootstrap = 5,
  enable_bookmarking = TRUE,
  code = TRUE,
  attrition = TRUE,
  show_help = TRUE,
  new_step = c("clone", "configure"),
  ...,
  run_app = TRUE
)
```

Arguments

steps	Set to TRUE (default) if multiple steps should be available.

`stats` Choose which statistics should be displayed for data (and some filters). Possible options are: "pre" - previous step stat, "post" - current step stats, ‘c("pre", "post")‘ - for both and NULL for no stats.

`run_button` Should Run button be displayed? If so, the current step computations are run only when clicked. Three options are available "none" - no button, "local" - button displayed at each step panel, "global" - button visible in top filtering panel.

`feedback` Set to TRUE (default) if feedback plots should be displayed at each filter.

`state` Set to TRUE (default) to enable get/set state panel.

`bootstrap` Boostrap version to be used for filtering panel. See bs_theme version argument.

`enable_bookmarking` Set to TRUE (default) if panel should be compatible with native shiny bookmarking.

`code` Set to TRUE (default) to enable reproducible code panel.

`attrition` Set to TRUE (default) to enable attrition plot panel.

`show_help` Set to TRUE (default) to enable help buttons.

`new_step` Choose which add step method should be used for creating new step. Possible options are: "clone" - copy filters from last step, "configure" - opening modal and allow to chose filters from available filters.

`...` Extra parameters passed to selected cohort methods. Currently unused.

`run_app` If ’TRUE’ the application will run using runApp, otherwise shinyApp object is returned.

Value

In case of ‘run_app=TRUE‘ no return value, used for side effect which is running a Shiny application. Otherwise shinyApp object.

Examples

```
if (interactive()) {
  library(shinyCohortBuilder)
  demo_app(steps = FALSE)
}
if (interactive()) {
  library(shinyCohortBuilder)
  demo_app(run_button = "local", state = FALSE)
}
```

gui *Run filtering panel locally*

Description

Run filtering panel locally

Usage

```
gui(
  cohort,
  steps = TRUE,
  stats = c("pre", "post"),
  run_button = "none",
  feedback = TRUE,
  state = TRUE,
  bootstrap = 5,
  enable_bookmarking = TRUE,
  code = TRUE,
  attrition = TRUE,
  show_help = TRUE,
  new_step = c("clone", "configure")
)
```

Arguments

cohort	Cohort object with configured filters.
steps	Set to TRUE (default) if multiple steps should be available.
stats	Choose which statistics should be displayed for data (and some filters). Possible options are: "pre" - previous step stat, "post" - current step stats, ‘c("pre", "post")‘ - for both and NULL for no stats.
run_button	Should Run button be displayed? If so, the current step computations are run only when clicked. Three options are available "none" - no button, "local" - button displayed at each step panel, "global" - button visible in top filtering panel.
feedback	Set to TRUE (default) if feedback plots should be displayed at each filter.
state	Set to TRUE (default) to enable get/set state panel.
bootstrap	Boostrap version to be used for filtering panel. See bs_theme version argument.
enable_bookmarking	Set to TRUE (default) if panel should be compatible with native shiny bookmarking.
code	Set to TRUE (default) to enable reproducible code panel.
attrition	Set to TRUE (default) to enable attrition plot panel.
show_help	Set to TRUE (default) to enable help buttons.

`new_step` Choose which add step method should be used for creating new step. Possible options are: "clone" - copy filters from last step, "configure" - opening modal and allow to chose filters from available filters.

Value

No return value, used for side effect which is running a Shiny application.

Examples

```
if (interactive()) {
  library(magrittr)
  library(cohortBuilder)
  library(shinyCohortBuilder)
  mtcars_source <- set_source(tblist(mtcars = mtcars))
  mtcars_cohort <- cohort(
    mtcars_source,
    filter("discrete", id = "am", dataset = "mtcars", variable = "am", value = 1)
  ) %>% run()
  gui(mtcars_cohort)
}
```

`gui-filter-layer` *Return GUI layer methods for filter of specified type*

Description

For each filter type ‘.gui_filter‘ method should return a list of the below objects:

- `input` - UI structure defining filter input controllers.
- `feedback` - List defining feedback plot output.
- `server` - Optional server-side expression attached to filter panel (e.g. filter specific observers).
- `update` - An expression used for updating filter panel based on its configuration.
- `post_stats` - TRUE if post statistics are displayed in filter controller (e.g. for discrete filter). If FALSE, some operations are skipped which results with better performance.
- `multi_input` - TRUE if multiple input controllers are used for providing filter value (e.g. range input where both numericInput and sliderInput are used). If FALSE, some operations are skipped which results with better performance.

If you want to learn more about creating filter layers see ‘vignette("gui-filter-layer")‘.

Usage

```
.gui_filter(filter, ...)

## S3 method for class 'discrete'
.gui_filter(filter, ...)

## S3 method for class 'range'
.gui_filter(filter, ...)

## S3 method for class 'date_range'
.gui_filter(filter, ...)

## S3 method for class 'discrete_text'
.gui_filter(filter, ...)

## S3 method for class 'multi_discrete'
.gui_filter(filter, ...)

## S3 method for class 'query'
.gui_filter(filter, ...)
```

Arguments

`filter`	Filter object.
`...`	Extra arguments passed to a specific method.

Value

List consisting filter metadata and methods that allow to perform filter based operations. See ‘vignette("custom-filters")‘.

See Also

source-gui-layer

Examples

```
library(cohortBuilder)
librarian_source <- set_source(as.tblist(librarian))
copies_filter <- filter(
  "range", id = "copies", name = "Copies", dataset = "books",
  variable = "copies", range = c(5, 12)
)
copies_filter_evaled <- copies_filter(librarian_source)
copies_filter_evaled$gui <- .gui_filter(copies_filter_evaled)

str(copies_filter_evaled$gui)
```

keep_na_input *Generate NA's filter selection GUI input*

Description

When used within filter's GUI input method, the component is responsible for updating 'keep_na' filter parameter.

Use '.update_keep_na_input' inside filter's GUI update method to update the output based on the filter state.

Usage

```
.keep_na_input(
  input_id,
  filter,
  cohort,
  msg_fun = function(x) glue::glue("Keep missing values ({x})")
)

.update_keep_na_input(
  session,
  input_id,
  filter,
  cohort,
  msg_fun = function(x) glue::glue("Keep missing values ({x})")
)
```

Arguments

input_id	Id of the keep na input.
filter	Filter object.
cohort	Cohort object.
msg_fun	Function taking number of missing values as an argument and returning missing values label.
session	Shiny session object.

Value

Nested list of 'shiny.tag' objects storing html structure of the input, or no value in case of usage 'update' method.

Examples

```
library(magrittr)
library(cohortBuilder)
```

```
librarian_source <- set_source(as.tblist(librarian))
coh <- cohort(
  librarian_source,
  filter(
    "range", id = "copies", name = "Copies", dataset = "books",
    variable = "copies", range = c(5, 12)
  )
) %>% run()
.keep_na_input("keep_na", coh$get_filter("1", "copies"), coh)
```

---

pre_post_stats *Generate structure of pre/post statistics*

Description

The method exported only for custom extensions use.

‘.pre_post_stats‘ returns the statistics having html tags structure. ‘.pre_post_stats_text‘ returns the same output but flatten to a single character object. The latter function works faster and supports vector arguments.

Usage

```
.pre_post_stats(
  current,
  previous,
  name,
  brackets = FALSE,
  percent = FALSE,
  stats = c("pre", "post")
)

.pre_post_stats_text(
  current,
  previous,
  name,
  brackets = TRUE,
  percent = FALSE,
  stats = c("pre", "post")
)
```

Arguments

current	Current step statistic value.
previous	Previous step statistic value.
name	Name displayed nearby the statistics output.
brackets	If TRUE, statistics will be displayed in brackets.

percent	Should current/previous ration in percentages be displayed?
stats	Vector of "pre" and "post" defining which statistics should be returned. "pre" for previous, "post" for current and NULL for none.

Value

A ‘shiny.tag‘ class ‘span‘ object defining html structure of data/value statistics, or character object.

Examples

```
.pre_post_stats(5, 10, "books")
.pre_post_stats_text(5, 10, "books")
.pre_post_stats(5, 10, "books", brackets = TRUE)
.pre_post_stats_text(5, 10, "books", brackets = TRUE)
.pre_post_stats(5, 10, "books", percent = TRUE)
.pre_post_stats_text(5, 10, "books", percent = TRUE)
.pre_post_stats_text(5:6, 10:11, "books", percent = TRUE)
```

---

rendering-custom-attrition

Method for generating custom attrition output

Description

When method is defined for selected source, the output is displayed in attrition modal tab.

Usage

```
.custom_attrition(source, ...)
```

Arguments

source	Source object.
...	Extra arguments passed to specific method.

Details

Similar to .step_attrition the method should return list of ‘render‘ and ‘output‘ expressions.

Value

List of two objects: ‘render‘ and ‘output‘ defining rendering and output placeholder for custom attrition plot feature.

See Also

source-gui-layer

Examples

```
if (interactive()) {
  library(magrittr)
  library(shiny)
  library(cohortBuilder)
  library(shinyCohortBuilder)

  .custom_attrition.tblist <- function(source, id, cohort, session, ...) {
    ns <- session$ns
    choices <- names(source$dtconn)

    list(
      render = shiny::renderPlot({
        cohort$show_attrition(dataset = session$input$attrition_input)
      }),
      output = shiny::tagList(
        shiny::h3("Step-wise Attrition Plot"),
        shiny::selectInput(ns("attrition_input"), "Choose dataset", choices),
        shiny::plotOutput(id)
      )
    )
  }
  coh <- cohort(
    set_source(as.tblist(librarian)),
    step(
      filter(
        "range", id = "copies", dataset = "books",
        variable = "copies", range = c(5, 12)
      )
    ),
    step(
      filter(
        "range", id = "copies", dataset = "books",
        variable = "copies", range = c(6, 8)
      )
    )
  ) %>% run()

  ui <- fluidPage(
    div(id = "attrition")
  )

  server <- function(input, output, session) {
    rendering <- .custom_attrition(
      coh$get_source(), id = "attr", cohort = coh, session = session, dataset = "books"
    )
    insertUI("#attrition", ui = rendering$output)
    output$attr <- rendering$render
  }

  shinyApp(ui, server)
}
```

rendering-filters *Render filtering panels for all the filters included in Cohort*

Description

The method exported only for custom extensions use.

Usage

```
.render_filters(source, ...)

## Default S3 method:
.render_filters(source, cohort, step_id, ns, ...)

## S3 method for class 'tblist'
.render_filters(source, cohort, step_id, ns, ...)
```

Arguments

source	Source object.
...	Extra arguments passed to a specific method.
cohort	Cohort object.
step_id	Id of the step.
ns	Namespace function.

Details

Within the method you should define source data stats output (see .update_data_stats), and define a loop that renders filtering panel for each filter (using .render_filter).

Value

Nested list of ‘shiny.tag‘ objects storing html structure of filter input panels.

See Also

source-gui-layer

Examples

```
if (interactive()) {
  library(magrittr)
  library(shiny)
  library(cohortBuilder)
  library(shinyCohortBuilder)

  ui <- fluidPage(
    actionButton("add_filter", "Add Filter"),
    div(id = "filter_container")
  )

  server <- function(input, output, session) {
    add_gui_filter_layer <- function(public, private, ...) {
      private$steps[["1"]]$filters$copies$gui <- .gui_filter(
        private$steps[["1"]]$filters$copies
      )
      private$steps[["1"]]$filters$registered$gui <- .gui_filter(
        private$steps[["1"]]$filters$registered
      )
    }
    add_hook("post_cohort_hook", add_gui_filter_layer)
    coh <- cohort(
      set_source(as.tblist(librarian)),
      filter(
        "range", id = "copies", name = "Copies", dataset = "books",
        variable = "copies", range = c(5, 12)
      ),
      filter(
        "date_range", id = "registered", name = "Registered",  dataset = "borrowers",
        variable = "registered", range = c(as.Date("2010-01-01"), Inf)
      )
    ) %>% run()
    coh$attributes$session <- session
    coh$attributes$feedback <- TRUE

    observeEvent(input$add_filter, {
      insertUI(
        "#filter_container",
        ui = .render_filters(
          coh$get_source(),
          cohort = coh,
          step_id = "1",
          ns = function(x) x
        ))
    }, ignoreInit = TRUE, once = TRUE)
  }

  shinyApp(ui, server)
}
```

rendering-step-attrition

Generate output of attrition plot

Description

The method should return list of two object:

- `render` - Rendering expression of attrition output.
- `output` - Output expression related to rendering (with id equal to ‘id‘ parameter).

For example:

```
list(
  render = shiny::renderPlot({
    cohort$show_attrition()
  }),
  output = shiny::plotOutput(id)
)
```

Usage

```
.step_attrition(source, ...)

## Default S3 method:
.step_attrition(source, id, cohort, session, ...)

## S3 method for class 'tblist'
.step_attrition(source, id, cohort, session, ...)
```

Arguments

`source`	Source object.
`...`	Extra arguments passed to specific method.
`id`	Id of attrition output.
`cohort`	Cohort object.
`session`	Shiny session object.

Value

List of two objects: ‘render‘ and ‘output‘ defining rendering and output placeholder for step attrition plot feature.

See Also

source-gui-layer

Examples

```
if (interactive()) {
  library(magrittr)
  library(shiny)
  library(cohortBuilder)
  library(shinyCohortBuilder)

  coh <- cohort(
    set_source(as.tblist(librarian)),
    step(
      filter(
        "range", id = "copies", dataset = "books",
        variable = "copies", range = c(5, 12)
      )
    ),
    step(
      filter(
        "range", id = "copies", dataset = "books",
        variable = "copies", range = c(6, 8)
      )
    )
  ) %>% run()

  ui <- fluidPage(
    div(id = "attrition")
  )

  server <- function(input, output, session) {
    rendering <- .step_attrition(
      coh$get_source(), id = "attr", cohort = coh, session = session, dataset = "books"
    )
    insertUI("#attrition", ui = rendering$output)
    output$attr <- rendering$render
  }

  shinyApp(ui, server)
}
```

---

scb_chart_palette *Default color palette used for filter feedback plots*

Description

It's a list of the following elements:

Usage

```
scb_chart_palette
```

Format

An object of class `list` of length 3.

Details

- `discrete` - Discrete filter plot colors.
- `histogram` - Range and date range histogram color.
- `no_data` - Color used to mark missing variables on feedback plots.

The palette is used as default `scb_chart_palette` option, that can be overwritten with custom palettes.

`scb_icons` *Default filtering panel icons*

Description

Icons can be overwritten with using `sbc_icons` option.

Usage

```
scb_icons
```

Format

An object of class `list` of length 15.

`scb_labels` *Default filtering panel labels*

Description

Labels can be overwritten with using `sbc_labels` option.

Usage

```
scb_labels
```

Format

An object of class `list` of length 15.

source-gui-layer *Source compatibility methods.*

Description

List of methods that allow compatibility of different source types. Most of the methods should be defined in order to make new source layer functioning. See 'Details' section for more information.

Details

The package is designed to make the functionality work with multiple data sources. Data source can be based for example on list of tables, connection to database schema or API service that allows to access and operate on data. In order to make new source type layer functioning, the following list of methods should be defined:

- `.render_filters`
- `.update_data_stats`
- `.step_attrition`
- `.custom_attrition`
- `.available_filter_choices`
- `autofilter`

Except from the above methods, you may extend the existing or new source with providing custom gui filtering methods. See gui-filter-layer. In order to see more details about how to implement custom source check ‘vignette("custom-gui-layer")‘.

Value

Various type outputs dependent on the selected method. See each method documentation for details.

trigger-action *Trigger filtering panel action*

Description

The two functions that allow to trigger a specific filtering panel action directly from Shiny server (.trigger_action) or application browser (.trigger_action_js) attached to a specific JS event, e.g. onclick.

Check Details section to see possible options.

Usage

```
.trigger_action(session, action, params = NULL)

.trigger_action_js(action, params = list(), ns = function(id) id)
```

Arguments

`session`	Shiny session object.
`action`	Id of the action.
`params`	List of parameters passed to specific action method.
`ns`	Namespace function (if used within Shiny modal).

Details

The list of possible actions:

- `update_filter` - Calls ‘shinyCohortBuilder:::gui_update_filter‘ that triggers filter arguments update.
- `add_step` - Calls ‘shinyCohortBuilder:::gui_add_step‘ that triggers adding a new filtering step (based on configuration of the previous one).
- `rm_step` - Calls ‘shinyCohortBuilder:::gui_rm_step‘ used to remove a selected filtering step.,
- `clear_step` - Calls ‘shinyCohortBuilder:::gui_clear_step‘ used to clear filters configuration in selected step.
- `update_step` - Calls ‘shinyCohortBuilder:::gui_update_step‘ used to update filters and feedback plots for the specific filter GUI panel.
- `update_data_stats` - Calls ‘shinyCohortBuilder:::gui_update_data_stats‘ that is called to update data statistics.
- `show_repro_code` - Calls ‘shinyCohortBuilder:::gui_show_repro_code‘ that is used to show reproducible code.
- `run_step` - Calls ‘shinyCohortBuilder:::gui_run_step‘ used to trigger specific step data calculation.
- `show_state` - Calls ‘shinyCohortBuilder:::gui_show_state‘ that is used to show filtering panel state json.
- `input_state` - Calls ‘shinyCohortBuilder:::gui_input_state‘ that is used to generate modal in which filtering panel state can be provided (as json).
- `restore_state` - Calls ‘shinyCohortBuilder:::gui_restore_state‘ used for restoring filtering panel state based on provided json.
- `show_attrition` - Calls ‘shinyCohortBuilder:::gui_show_attrition‘ a method used to show attrition data plot(s).

Both ‘.trigger_action‘ and ‘.trigger_action_js‘ methods are exported for advanced use only.

Value

No return value (‘.trigger_action‘ - sends message to the browser) or character string storing JS code for sending input value to Shiny server (‘.trigger_action_js‘).

Examples

```
if (interactive()) {
  library(shiny)
  library(shinyCohortBuilder)

  shiny::addResourcePath(
    "shinyCohortBuilder",
    system.file("www", package = "shinyCohortBuilder")
  )
  ui <- fluidPage(
    tags$head(
    shiny::tags$script(type = "text/javascript", src = file.path("shinyCohortBuilder", "scb.js"))
    ),
    tags$button(
      "Trigger action from UI", class = "btn btn-default",
      onclick = .trigger_action_js("uiaction", params = list(a = 1))
    ),
    actionButton("send", "Trigger action from server")
  )

  server <- function(input, output, session) {
    observeEvent(input$send, {
      .trigger_action(session, "serveraction", params = list(a = 2))
    })
    observeEvent(input$action, {
      print(input$action)
    })
  }

  shinyApp(ui, server)
}
```

---

updating-data-statistics

Render source data related statistics

Description

The function should assign rendering that displays data source statistics to the valid output. By default, the output is placed within .render_filters method.

Usage

```
.update_data_stats(source, ...)

## Default S3 method:
.update_data_stats(source, step_id, cohort, session, ...)

## S3 method for class 'tblist'
.update_data_stats(source, step_id, cohort, session, ...)
```

Arguments

source	Source object.
...	Extra arguments passed to a specific method.
step_id	Id if filtering step.
cohort	Cohort object.
session	Shiny session object.

Details

When rendering the output, a good practice is to use cached data statistics available with ‘cohort$get_cache(step_id)‘. This way, you omit running additional computations which results with performance improvement.

Value

No return value, used for side effect which assigning Cohort data statistics to the ‘output‘ object.

See Also

source-gui-layer

Examples

```
if (interactive()) {
  library(magrittr)
  library(shiny)
  library(cohortBuilder)
  library(shinyCohortBuilder)

  ui <- fluidPage(
    sliderInput("step_two_max", "Max step two copies", min = 6, max = 12, value = 8),
    uiOutput("2-stats_books")
  )

  server <- function(input, output, session) {
    coh <- cohort(
      set_source(as.tblist(librarian)),
      step(
        filter(
          "range", id = "copies", dataset = "books",
          variable = "copies", range = c(5, 12)
        )
      ),
      step(
        filter(
          "range", id = "copies", dataset = "books",
          variable = "copies", range = c(6, 8)
        )
      )
    ) %>% run()
```

```
    coh$attributes$stats <- c("pre", "post")
    observeEvent(input$step_two_max, {
      coh$update_filter("copies", step_id = 2, range = c(6, input$step_two_max))
      run(coh, min_step_id = "2")
     .update_data_stats(coh$get_source(), step_id = "2", cohort = coh, session = session)
    })
  }

  shinyApp(ui, server)
}
```

Index

∗ **datasets**
scb_chart_palette, 25
scb_icons, 26
scb_labels, 26
.available_filters_choices (available-filters-choices), 9
.cb_input, 3
.custom_attrition (rendering-custom-attrition), 20
.gui_filter (gui-filter-layer), 16
.keep_na_input (keep_na_input), 18
.pre_post_stats (pre_post_stats), 19
.pre_post_stats_text (pre_post_stats), 19
.render_filter, 4, *22*
.render_filters, *29*
.render_filters (rendering-filters), 22
.save_observer, 6
.sendOutput, 7
.step_attrition, *20*
.step_attrition (rendering-step-attrition), 24
.trigger_action (trigger-action), 27
.trigger_action_js (trigger-action), 27
.update_data_stats, *22*
.update_data_stats (updating-data-statistics), 29
.update_keep_na_input (keep_na_input), 18

autofilter, 8
available-filters-choices, 9

bs_theme, *14, 15*

cb_changed, 10
cb_server (cb_ui), 11
cb_ui, 11

demo_app, 13

gui, 15
gui-filter-layer, 16, *27*

keep_na_input, 18

pre_post_stats, 19

rendering-custom-attrition, 20
rendering-filters, 22
rendering-step-attrition, 24
runApp, *14*

scb_chart_palette, 25
scb_icons, 26
scb_labels, 26
shinyApp, *14*
shinyCohortBuilder-package, 3
source-gui-layer, *8*, *17*, *20*, *22*, *24*, 27, *30*

trigger-action, 27

updating-data-statistics, 29