Package ‘shinyfilters’

December 18, 2025

Title Use 'shiny' to Filter Data

Version 0.2.0

Description Provides an interface to 'shiny' inputs used for filtering vectors, data.frames, and other objects. 'S7'-based implementation allows for seamless extensibility.

License MIT + file LICENSE

Imports methods, S7 (>= 0.2.0), shiny

Suggests bslib, DT, htmltools, knitr, rmarkdown, shinyWidgets, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

VignetteBuilder knitr

URL https://joshwlivingston.github.io/shinyfilters/, https://github.com/joshwlivingston/shinyfilters

BugReports https://github.com/joshwlivingston/shinyfilters/issues

NeedsCompilation no

Author Josh Livingston [cre, aut]

Maintainer Josh Livingston <joshwlivingston@gmail.com>

Repository CRAN

Date/Publication 2025-12-17 23:20:02 UTC

Contents

`apply_filters` *Apply Filters to an object*

Description

Applies a list of filters to an object, returning the filtered object.

Usage

```
apply_filters(
  x,
  filter_list,
  filter_combine_method = "and",
  expanded = FALSE,
  cols = NULL,
  ...
)
```

Arguments

`x`	An object to filter; typically a data.frame.
`filter_list`	A named list of filter values, used to filter the values in `x`. If `filter_list` is `NULL`, `x` is returned unmodified.
`filter_combine_method`	A string or function indicating how to combine multiple filters. If a string, it can be "and" (or "&") for logical AND, or "or" (or "|") for logical OR. If a function, it should take two logical vectors and return a combined logical vector.
`expanded`	Logical; if `TRUE`, returns a named list of data.frames, each containing one column, its own, filtered according to the values of all *other* filters.
`cols`	Optional character vector of column names to retain in the output when `x` is a data.frame. If `NULL` (the default), all columns are retained.
`...`	Additional arguments passed to `get_filter_logical()`.

Value

A filtered object, or a named list of filtered objects if `expanded = TRUE`.

Examples

```
library(S7)
df <- data.frame(
 category = rep(letters[1:3], each = 4),
 value = 1:12,
 date = as.Date('2024-01-01') + 0:11
)
filters <- list(
  category = c("a", "b"),
  value = c(3, 11)
)

# Apply filters with logical AND
apply_filters(df, filters, filter_combine_method = "and")

# Apply filters with logical OR
apply_filters(df, filters, filter_combine_method = "or")

# Get expanded filters
apply_filters(df, filters, expanded = TRUE)
```

`args_filter_input` *Derive Arguments for* **shiny** *Inputs*

Description

Provides the appropriate function arguments for the input function selected by `filterInput()` or `updateFilterInput()`.

Usage

```
args_filter_input(x, ...)

args_update_filter_input(x, ...)
```

Arguments

`x`	The object being passed to `filterInput()` or `updateFilterInput()`.
`...`	Additional arguments passed to the method. See details.

Details

The following arguments are supported in `...`:

`range`	*(Date, POSIXt)*. Logical. If `TRUE`, `args_filter_input()` will provide the arguments for range date inputs.
`textbox`	*(character)*. Logical. If `FALSE` (the default), `args_filter_input()` will provide the arguments for select in
`choices_asis`	*(character, factor, list, logical)*. Logical. If `TRUE`, the choices provided to select inputs will not be modified.

`server`	If `TRUE`, indicates that the choices will be provided server-side. In this case, arguments are not computed for

Value

A named list of arguments for a **shiny** input function

Examples

```
args_filter_input(iris$Petal.Length)
```

`call_input_function` *Prepare and Evaluate Input Function and Arguments*

Description

Internal function used to prepare input arguments using `args_filter_input()`, and gracefully pass them to provided input function.

Usage

```
call_filter_input(x, .f, ...)

call_update_filter_input(x, .f, ...)
```

Arguments

`x`	The object being passed to `filterInput()`.
`.f`	The input function to be called.
`...`	Arguments passed to either `args_filter_input()` or provided input function.

Details

`call_filter_input()` and `call_update_filter_input()` are used when customizing **shinyfilters**. For more, see `vignette("customizing-shinyfilters")`.

Value

The result of calling the provided input function.

Examples

```
library(S7)
library(shiny)
# call_filter_input() is used inside filterInput() methods
method(filterInput, class_numeric) <- function(x, ...) {
  call_filter_input(x, sliderInput, ...)
}

# call_update_filter_input() is used inside updateFilterInput() methods
method(updateFilterInput, class_numeric) <- function(x, ...) {
  call_update_filter_input(x, updateSliderInput, ...)
}
```

`filterInput` *Create a* **shiny** *Input*

Description

Selects and creates a **shiny** input based the type of object `x` and other arguments.

Usage

```
filterInput(x, ...)
```

Arguments

`x`	The object used to create the input.
`...`	Arguments used for input selection or passed to the selected input. See details.

Details

The following arguments passed to `...` are supported:

`area`	*(character)*. Logical. Controls whether to use shiny::textAreaInput (`TRUE`) or shiny::textInput (`FALSE`, default). C
`range`	*(Date, POSIXt)*. Logical. Controls whether to use shiny::dateRangeInput (`TRUE`) or shiny::dateInput (`FALSE`, def
`selectize`	*(character, factor, list, logical)*. Logical. Controls whether to use shiny::selectizeInput (`TRUE`) or shiny::selectInp
`slider`	*(numeric)*. Logical. Controls whether to use shiny::sliderInput (`TRUE`) or shiny::numericInput (`FALSE`, default) .
`textbox`	*(character)*. Logical. Controls whether to use a text input (`TRUE`) or a dropdown input (`FALSE`, default).
`ns`	An optional namespace created by `shiny::NS()`. Useful when using `filterInput()` on a data.frame inside a s

Remaining arguments passed to `...` are passed to the `args_filter_input()` or the selected input function.

Value

One of the following **shiny** inputs is returned, based on the type of object passed to `x`, and other specified arguments. See `vignette("filter-input-catalog")` for the full list of examples.

Value	x	**Arguments**
shiny::dateInput	Date, POSIXt	*default*
shiny::dateRangeInput	Date, POSIXt	`range = TRUE`
shiny::numericInput	numeric	*default*
shiny::radioButtons	character, factor, list, logical	`radio = TRUE`
shiny::selectInput	character, factor, list, logical	*default*
shiny::selectizeInput	character, factor, list, logical	`selectize = TRUE`
shiny::sliderInput	numeric	`slider = TRUE`
shiny::textAreaInput	character	`textbox = TRUE, area = TRUE`
shiny::textInput	character	`textbox = TRUE`

Examples

```
library(shiny)

ui <- fluidPage(
 sidebarLayout(
 sidebarPanel(
 ##############################################
 # Create a filterInput() inside a shiny app:
 filterInput(
x = letters,
id = "letter",
label = "Pick a letter:"
 )
 ##############################################
 ),
 mainPanel(
 textOutput("selected_letter")
 )
 )
)

server <- function(input, output, session) {
 output$selected_letter <- renderText({
 paste("You selected:", input$letter)
 })
}

shinyApp(ui, server)
```

`get_filter_logical` *Compute a Filter Predicate*

Description

Computes a logical vector indicating which elements of `x` match the filter criteria specified by `val`.

Usage

```
get_filter_logical(x, val, ...)
```

Arguments

`x`	An object to filter; typically a data.frame.
`val`	The filter criteria.
`...`	Arguments passed to methods. See details.

Details

The following arguments are supported in `...`:

`column`	When `x` is a data.frame, `column` is the name of the column intended to be filtered.
`comparison`	When `x` is a numeric or Date and `val` is a length-**one** numeric or Date, `comparison` is the function used to com
`gte`	When `x` is a numeric or Date and `val` is a length-**two** numeric or Date, `gte` controls whether to use >= (`TRUE`, d
`lte`	When `x` is a numeric or Date and `val` is a length-**two** numeric or Date, `lte` controls whether to use <= (`TRUE`, d

Value

A logical vector indicating which elements of `x` match the filter criteria specified by `val`.

Examples

```
df <- data.frame(
  category = rep(letters[1:3], each = 4),
  value = 1:12,
  date = Sys.Date() + 0:11
)

# Filter character column
get_filter_logical(df, c("a", "b"), column = "category")

# Filter numeric column with single value
get_filter_logical(df, 5, column = "value", comparison = `<=`)

# Filter numeric column with range
get_filter_logical(df, c(3, 8), column = "value", gte = TRUE, lte = FALSE)
```

`get_input_ids` *Retrieve the Ids of Input Objects*

Description

Returns the (unnamespaced) ids of the inputs for the provided object.

Usage

```
get_input_ids(x, ...)
```

Arguments

`x`	An object for which to retrieve input ids; typically a data.frame.
`...`	Passed onto methods.

Value

A character vector of input ids.

Examples

```
df <- data.frame(
  name = c("Alice", "Bob"),
  age = c(25, 30),
  completed = c(TRUE, FALSE)
)

get_input_ids(df)
```

`get_input_labels` *Retrieve the Labels of Input Objects*

Description

Returns the labels of the **shiny** inputs for the provided object.

Usage

```
get_input_labels(x, ...)
```

Arguments

`x`	An object for which to retrieve input labels; typically a data.frame.
`...`	Passed onto methods.

Value

A character vector of input labels

Examples

```
df <- data.frame(
  name = c("Alice", "Bob"),
  age = c(25, 30),
  completed = c(TRUE, FALSE)
)

get_input_labels(df)
```

`get_input_values` *Get Multiple Values from a* **shiny** *Input Object*

Description

Retrieves multiple input values from a **shiny** `input` object based on the names provided in `x`.

Usage

```
get_input_values(input, x, ...)
```

Arguments

`input`	A **shiny** `input` object, i.e., the `input` argument to the shiny server.
`x`	A character vector of input names, or a data.frame whose column names are converted to input names via `get_input_ids()`.
`...`	Passed onto methods.

Value

A named list of input values corresponding to the names in `x`.

Examples

```
library(shiny)
df <- data.frame(
  name = c("Alice", "Bob"),
  age = c(25, 30),
  completed = c(TRUE, FALSE)
)
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      filterInput(df)
    ),
    mainPanel(
      verbatimTextOutput("output_all"),
      verbatimTextOutput("output_subset")
    )
  )
)
server <- function(input, output, session) {
  output$output <- renderPrint({
    get_input_values(input, df)
  })
  output$output_subset <- renderPrint({
    get_input_values(input, c("name", "completed"))
  })
}
shinyApp(ui, server)
```

`serverFilterInput` *Run the backend server for filterInput*

Description

Run the backend server for filterInput

Usage

```
serverFilterInput(
  x,
  input,
  filter_combine_method = "and",
  args_apply_filters = NULL,
  ...
)
```

Arguments

`x`	An object being filtered; typically a data.frame.
`input`	A **shiny** `input` object, or a reactive that resolves to a list of named values.
`filter_combine_method`	A string or function indicating how to combine multiple filters. If a string, it can be "and" (or "&") for logical AND, or "or" (or "|") for logical OR. If a function, it should take two logical vectors and return a combined logical vector.
`args_apply_filters`	A named list of additional arguments passed to `apply_filters()`.
`...`	Additional arguments passed to `updateFilterInput()`.

Value

A reactiveValues list with a single element, `input_values`, which contains the current filter input values as a named list.

Examples

```
library(bslib)
library(DT)
library(S7)
library(shiny)

must_use_radio <- new_S3_class(
 class = "must_use_radio",
 constructor = function(.data) .data
)
method(filterInput, must_use_radio) <- function(x, ...) {
 call_filter_input(x, shiny::radioButtons, ...)
}
method(updateFilterInput, must_use_radio) <- function(x, ...) {
 call_update_filter_input(x, shiny::updateRadioButtons, ...)
}

use_radio <- function(x) {
 structure(x, class = unique(c("must_use_radio", class(x))))
}

df_shared <- data.frame(
 x = letters,
 y = use_radio(sample(c("red", "green", "blue"), 26, replace = TRUE)),
 z = round(runif(26, 0, 3.5), 2),
 q = sample(Sys.Date() - 0:7, 26, replace = TRUE)
)

filters_ui <- function(id) {
 ns <- shiny::NS(id)
 filterInput(
 x = df_shared,
 range = TRUE,
 selectize = TRUE,
 slider = TRUE,
 multiple = TRUE,
 ns = ns
 )
}

filters_server <- function(id) {
 moduleServer(id, function(input, output, session) {
  # serverFilterInput() returns a shiny::observe() expressionc
  serverFilterInput(df_shared, input = input, range = TRUE)
  })
}
```

```
ui <- page_sidebar(
  sidebar = sidebar(filters_ui("demo")),
  DTOutput("df_full"),
  verbatimTextOutput("input_values"),
  DTOutput("df_filt")
)

server <- function(input, output, session) {
 res <- filters_server("demo")
 output$df_full <- renderDT(datatable(df_shared))
 output$input_values <- renderPrint(res$input_values)
 output$df_filt <- renderDT(datatable(apply_filters(
  df_shared,
  res$input_values
  )))
}

shinyApp(ui, server)
```

`updateFilterInput` *Create a* **shiny** *Input*

Description

Updates a **shiny** input based the type of object `x` and other arguments.

Usage

```
updateFilterInput(x, ...)
```

Arguments

`x`	The object used to create the input.
`...`	Arguments used for input selection or passed to the selected input update function. See details.

Details

The following arguments passed to `...` are supported:

`area`	*(character)*. Logical. Controls whether to use shiny::updateTextAreaInput (`TRUE`) or shiny::updateTextInput (`FA`
`range`	*(Date, POSIXt)*. Logical. Controls whether to use shiny::updateDateRangeInput (`TRUE`) or shiny::updateDateInp
`selectize`	*(character, factor, list, logical)*. Logical. Controls whether to use shiny::updateSelectizeInput (`TRUE`) or shiny::up
`slider`	*(numeric)*. Logical. Controls whether to use shiny::updateSliderInput (`TRUE`) or shiny::updateNumericInput (`FAL`
`textbox`	*(character)*. Logical. Controls whether to update a text input (`TRUE`) or a dropdown input (`FALSE`, default).

Remaining arguments passed to `...` are passed to `args_update_filter_input()` or the selected input update function.

Value

The result of the following **shiny** input updates is returned, based on the type of object passed to `x`, and other specified arguments.

Value	`x`	**Arguments**
shiny::updateDateInput	Date, POSIXt	*default*
shiny::updateDateRangeInput	Date, POSIXt	`range = TRUE`
shiny::updateNumericInput	numeric	*default*
shiny::updateRadioButtons	character, factor, list, logical	`radio = TRUE`
shiny::updateSelectInput	character, factor, list, logical	*default*
shiny::updateSelectizeInput	character, factor, list, logical	`selectize = TRUE`
shiny::updateSliderInput	numeric	`slider = TRUE`
shiny::updateTextAreaInput	character	`textbox = TRUE, area = TRUE`
shiny::updateTextInput	character	`textbox = TRUE`

Examples

```
library(shiny)

fruits <- list(
"a" = c("apples", "avocados"),
"b" = c("bananas", "blueberries"),
"c" = c("cherries", "cantaloupe")
)

ui <- fluidPage(
sidebarLayout(
sidebarPanel(
filterInput(
x = letters[1:3],
inputId = "letter",
label = "Pick a letter:",
       multiple = TRUE
),
filterInput(
x = fruits,
inputId = "fruits",
label = "Pick a fruit:"
)
),
   mainPanel()
)
)

server <- function(input, output, session) {
shiny::observe({
fruits_filtered <- fruits
if (!is.null(input$letter) && length(input$letter) != 0L) {
fruits_filtered <- fruits[input$letter]
}
#########################################################
# 2. Call updateFilterInput() inside the shiny server:
updateFilterInput(x = fruits_filtered, inputId = "fruits")
#########################################################
})
}
shinyApp(ui, server)
```

Index