

Package: shinyfilters (via r-universe)

October 18, 2026

Title Create 'shiny' Inputs from Vectors, 'data.frames', or any R Object

Version 0.3.1.9000

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

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

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

Imports cli, htmltools, methods, rlang, S7, shiny

Suggests bslib, data.table, dplyr, DT, knitr, magrittr, nycflights13, rmarkdown, roxygen2, shinyWidgets, testthat (>= 3.2.0), tibble, tidyselect, withr

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R (>= 3.5)

LazyData true

Config/pak/sysreqs cmake make libuv1-dev zlib1g-dev

Repository <https://joshwlivingston.r-universe.dev>

Date/Publication 2026-10-18 17:05:53 UTC

RemoteUrl <https://github.com/joshwlivingston/shinyfilters>

RemoteRef HEAD

RemoteSha 1f28f0def0fb3730af2186bd4d5c84628349ed65

Contents

apply_filters	2
args_filter_input	3
call_input_function	4
filterInput	5
get_filter_logical	7
get_input_ids	8
get_input_labels	9
get_input_values	10
nyc_flights	11
serverFilterInput	12
shinyfilters	14
shinyfilters-class	16
updateFilterInput	18
with_args	20
with_defaults	21
with_filters	22
with_ns	24
Index	25

apply_filters	<i>Apply Filters to an object</i>
---------------	-----------------------------------

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. A missing value in x matches a filter only when NA is one of its 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.
expanded	Logical; if TRUE, returns a named list of data.frames, each containing one column, its own, filtered according to the values of all <i>other</i> 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 <code>get_filter_logical()</code> .

Value

A filtered object, or a named list of filtered objects if `expanded = TRUE`. In a filtered data frame, a column whose class has no `[` method keeps its attributes, such as a class that chooses its input.

Examples

```
library(S7)
df <- data.frame(
  stringsAsFactors = FALSE,
  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)
```

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. Only applies when `x` is of class `Date` or `POSIXt`.

textbox (*character*). Logical. If FALSE (the default), `args_filter_input()` will provide the arguments for select inputs.

choices_asis (*character, factor, logical*). Logical. If TRUE, the choices provided to select inputs will not be modified. If FALSE (the default), duplicate values will be removed and the choices will be sorted. Only applies when `x` is of class `character`, `factor`, or `logical`.

server If TRUE, indicates that the choices will be provided server-side. In this case, arguments are not computed for `args_filter_input()`. Ignored in `args_update_filter_input()`.

args_unique An optional named list of arguments passed to `unique()`, called when `x` is a *character, factor, or logical*, `textbox = FALSE`, and `choices_asis = FALSE`.

args_sort An optional named list of arguments passed to `sort()`, which is called after `unique()`.

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

<code>x</code>	The object being passed to <code>filterInput()</code> .
<code>.f</code>	The input function to be called.
<code>...</code>	Arguments passed to either <code>args_filter_input()</code> 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

<code>x</code>	The object used to create the input.
<code>...</code>	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 `textAreaInput` (TRUE) or `textInput` (FALSE, default). Only applies when `textbox` is TRUE.
- radio** (*character, factor, list, logical*). Logical. Controls whether to use `radioButtons` (TRUE) or a dropdown input (FALSE, default). For character vectors, radio only applies if `textbox` is FALSE, the default.
- range** (*Date, POSIXt*). Logical. Controls whether to use `dateRangeInput` (TRUE) or `dateInput` (FALSE, default).
- selectize** (*character, factor, list, logical*). Logical. Controls whether to use `selectizeInput` (TRUE) or `selectInput` (FALSE, default). For character vectors, selectize only applies if `textbox` is FALSE, the default.
- slider** (*numeric*). Logical. Controls whether to use `sliderInput` (TRUE) or `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 `NS()`. Useful when using `filterInput()` on a data.frame inside a **shiny** module.

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	<code>x</code>	Arguments
<code>dateInput</code>	Date, POSIXt	<i>default</i>
<code>dateRangeInput</code>	Date, POSIXt	<code>range = TRUE</code>
<code>numericInput</code>	numeric	<i>default</i>
<code>radioButtons</code>	character, factor, list, logical	<code>radio = TRUE</code>
<code>selectInput</code>	character, factor, list, logical	<i>default</i>
<code>selectizeInput</code>	character, factor, list, logical	<code>selectize = TRUE</code>
<code>sliderInput</code>	numeric	<code>slider = TRUE</code>
<code>textAreaInput</code>	character	<code>textbox = TRUE, area = TRUE</code>
<code>textInput</code>	character	<code>textbox = TRUE</code>

Examples

```
library(shiny)

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      # Create a filterInput() inside a shiny app:
      filterInput(
        x = letters,
        inputId = "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	<i>Compute a Filter Predicate</i>
--------------------	-----------------------------------

Description

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

Usage

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

Arguments

<code>x</code>	An object to filter; typically a data.frame.
<code>val</code>	The filter criteria.
<code>...</code>	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 compare `x` with `val`. The default is `<=`.

gte When `x` is a numeric or Date and `val` is a length-**two** numeric or Date, `gte` controls whether to use `>=` (TRUE, default) or `>` (FALSE) on `val[[1]]`.

lte When `x` is a numeric or Date and `val` is a length-**two** numeric or Date, `lte` controls whether to use `<=` (TRUE, default) or `<` (FALSE) on `val[[2]]`.

When `val` is a character vector, such as the value of a select or radio input, and `x` isn't a character, factor, or logical vector, the elements of `x` are compared as text. A datetime `x` is compared as its date, whether `val` is text, a Date, or a datetime.

Value

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

Examples

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

get_input_values	<i>Get Multiple Values from a shiny Input Object</i>
------------------	---

Description

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

Usage

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

Arguments

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

Value

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

Examples

```
library(shiny)
df <- data.frame(
  stringsAsFactors = FALSE,
  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$output_all <- renderPrint({
    get_input_values(input, df)
  })
  output$output_subset <- renderPrint({
    get_input_values(input, c("name", "completed"))
  })
}
```

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

nyc_flights

Flights departing New York City

Description

A sample of 200 flights that departed New York City airports in 2013, with a mix of column types for trying out `filterInput()`.

Usage

```
nyc_flights
```

Format

A data frame with 200 rows and 7 columns:

date Date of departure.

carrier Two-letter airline code.

origin Origin airport: "EWR", "JFK", or "LGA".

dest Destination airport code.

dep_delay Departure delay in minutes. Negative values are early departures; NA for cancelled flights.

distance Distance between airports in miles.

delayed Whether the flight arrived more than 15 minutes late; NA when the arrival time is unknown.

Source

A random sample of flights from the nycflights13 package (<https://github.com/tidyverse/nycflights13>), released under CC0.

serverFilterInput	<i>Run the backend server for filterInput</i>
-------------------	---

Description

shinyfilters_server() runs an observer that makes your filters interdependent. serverFilterInput() is an alias of shinyfilters_server().

Usage

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

shinyfilters_server(
  x,
  session = getDefaultReactiveDomain(),
  filter_combine_method = "and",
  args_apply_filters = NULL,
  ...,
  input = deprecated()
)
```

Arguments

x	An object being filtered; typically the result of shinyfilters() .
session	The shiny session whose inputs are read and updated. Defaults to the current session. The inputs of a configuration with a namespace (see with_ns()) are found from any session of the app, so shinyfilters_server() can be called at the top level of the server or inside the module. The namespace is the full one, as in the UI: inside a module's server that is session\$ns(NULL), which differs from the module's id when modules are nested.
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() .
input	Deprecated: omit it, or use session. A shiny input object, or a reactive that resolves to a list of named values.

Details

Only the inputs that have no value are updated, and they are left without one. The inputs of a configuration made by `shinyfilters()` are updated as `updateFilterInput()` updates them, from the filtered data.

An input set by a function other than the **shiny** inputs `filterInput()` creates needs the function that updates it. A **shinyWidgets** input, such as `shinyWidgets::pickerInput`, comes with its own. A column's update sends its choices, or its minimum and maximum, and that function is given the ones it has an argument for. One with an argument for none of them, such as `shinyWidgets::updateNumericRangeInput`, leaves its input as it is, so the input doesn't follow the other filters. Any other input, such as `shiny::checkboxGroupInput`, needs its update function named with `.update_fn = fn` among the input's arguments, as described in `with_filters()`. Without it, `shinyfilters_server()` errors.

Value

A reactiveValues list with two elements: `filtered`, the filtered data, and `input_values`, 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, radioButton, ...)
}
method(updateFilterInput, must_use_radio) <- function(x, ...) {
  call_update_filter_input(x, updateRadioButtons, ...)
}

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

df_shared <- data.frame(
  stringsAsFactors = FALSE,
  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 <- 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 reactiveValues list
    serverFilterInput(df_shared, 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)

```

shinyfilters

*Configure the Filters for a Data Frame***Description**

shinyfilters() creates a set of interdependent filters for use in a **shiny** application.

Usage

```

shinyfilters(
  .data,
  ...,
  area = FALSE,
  radio = FALSE,

```

```

    range = TRUE,
    selectize = TRUE,
    multiple = TRUE,
    slider = TRUE,
    textbox = FALSE,
    ns = NULL
  )

```

Arguments

<code>.data</code>	A <code>data.frame</code> .
<code>...</code>	Named arguments passed to the selected input.
<code>area</code>	(<i>character</i>). Logical. Controls whether to use <code>textAreaInput</code> (TRUE) or <code>textInput</code> (FALSE). Only applies when <code>textbox</code> is TRUE.
<code>radio</code>	(<i>character, factor, list, logical</i>). Logical. Controls whether to use <code>radioButtons</code> (TRUE) or a dropdown input (FALSE). For character vectors, <code>radio</code> only applies if <code>textbox</code> is FALSE. TRUE turns off the <code>selectize</code> default; setting both to TRUE is an error.
<code>range</code>	(<i>Date, POSIXt</i>). Logical. Controls whether to use <code>dateRangeInput</code> (TRUE) or <code>dateInput</code> (FALSE).
<code>selectize</code>	(<i>character, factor, list, logical</i>). Logical. Controls whether to use <code>selectizeInput</code> (TRUE) or <code>selectInput</code> (FALSE). For character vectors, <code>selectize</code> only applies if <code>textbox</code> is FALSE.
<code>multiple</code>	Passed to <code>selectInput</code> or <code>selectizeInput</code> .
<code>slider</code>	(<i>numeric</i>). Logical. Controls whether to use <code>sliderInput</code> (TRUE) or <code>numericInput</code> (FALSE).
<code>textbox</code>	(<i>character</i>). Logical. Controls whether to use a text input (TRUE) or a dropdown input (FALSE).
<code>ns</code>	An optional namespace created by <code>NS()</code> or an object to be coerced to a namespace. Useful when using <code>shinyfilters()</code> inside a shiny module.

Value

A `shinyfilters` object.

See Also

`with_filters()`, `with_ns()`

Examples

```

shinyfilters(nyc_flights)

# Use sliders for every numeric column
shinyfilters(nyc_flights, slider = TRUE)

```

shinyfilters-class *'shinyfilters' Objects*

Description

A shinyfilters object holds a data.frame and all of the data's filters. To create one, call `shinyfilters()`. To modify one, use `with_filters()`, `with_args()`, `with_defaults()`, or `with_ns()`.

Usage

The object behaves much like the data.frame that defines it.

```
filters <- shinyfilters(nyc_flights)
```

Names:

`names(filters)` provides the names of the data.frame.

Shape:

`dim(filters)`, `length(filters)`, `nrow(filters)`, and `ncol(filters)` describe the dimensions.

Data:

`as.data.frame(filters)` returns the data.

Subsetting:

`filters$col` creates the input for one filter.

`filters[[cols]]` creates the inputs for all filters in cols.

`filters[cols]` subsets a shinyfilters object, returning only filters matching cols.

shinyfilters supports `<tidy-select>` in both `[]` and `[[`

In a shiny app:

A shinyfilters object can be placed directly inside a ui:

```
library(bslib)
ui <- page_sidebar(sidebar = filters)
```

To enable interdependent filters, call `shinyfilters_server` in the server:

```
server <- function(...) {
  shinyfilters_server(filters)
}
```

Creating filters:

Sometimes, you will have to create the filters as you place them in the app. The following functions create filters:

- `[]`
- `$`
- `filterInput`

```
ui <- function(request) {
  page_sidebar(sidebar = filterInput(filters))
}
```

Currently, it is known that you must create the filters when:

1. Bookmarking is enabled
2. When placing a filter inside [bslib::accordion](#).

In these instances, shinyfilters will intercept and throw an informative error.

```
accordion(shinyfilters(nyc_flights))
```

```
#> Error:
```

```
#> ! <shinyfilters> objects cannot be used in some shiny functions.
```

```
#> i Use `[`, `$`, or `filterInput()`` to render the filters directly.
```

Printing

Printing shows the input each column receives:

```
shinyfilters(nyc_flights)
#> <shinyfilters> * 7 filters
#>
#> Filters
#>   date      <date>  dateRangeInput
#>   carrier   <chr>   selectizeInput
#>   origin    <fct>   selectizeInput
#>   dest       <chr>   selectizeInput
#>   dep_delay <dbl>   sliderInput
#>   distance  <dbl>   sliderInput
#>   delayed   <lgl>   selectizeInput
```

Any modifications will be displayed:

```
shinyfilters(nyc_flights, range = FALSE) |>
  with_filters(date = "date") |>
  with_ns("sidebar-mod")
#> <shinyfilters> * 7 filters * namespace "sidebar-mod"
#>
#> Filters
#> *   date      <date>  dateInput
#>   carrier   <chr>   selectizeInput
#>   origin    <fct>   selectizeInput
#>   dest       <chr>   selectizeInput
#>   dep_delay <dbl>   sliderInput
#>   distance  <dbl>   sliderInput
#>   delayed   <lgl>   selectizeInput
#>
#> Default Overrides
#>   range = FALSE
#>
#> * Filter set by `with_filters()``
```

updateFilterInput	<i>Update a shiny Input</i>
-------------------	------------------------------------

Description

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

Usage

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

Arguments

<code>x</code>	The object used to create the input, or a configuration made by shinyfilters() .
<code>...</code>	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 [updateTextAreaInput](#) (TRUE) or [updateTextInput](#) (FALSE, default). Only applies when `textbox` is TRUE.

radio (*character, factor, list, logical*). Logical. Controls whether to use [updateRadioButtons](#) (TRUE) or a dropdown input update function (FALSE, default). For character vectors, radio only applies if `textbox` is FALSE, the default.

range (*Date, POSIXt*). Logical. Controls whether to use [updateDateRangeInput](#) (TRUE) or [updateDateInput](#) (FALSE, default).

selectize (*character, factor, list, logical*). Logical. Controls whether to use [updateSelectizeInput](#) (TRUE) or [updateSelectInput](#) (FALSE, default). For character vectors, selectize only applies if `textbox` is FALSE, the default.

slider (*numeric*). Logical. Controls whether to use [updateSliderInput](#) (TRUE) or [updateNumericInput](#) (FALSE, default).

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.

When `x` is a configuration made by [shinyfilters\(\)](#), each column's input is updated with the function and arguments that match the input [filterInput\(\)](#) creates for it: the configuration's defaults, and the input and arguments [with_filters\(\)](#) or [with_args\(\)](#) set for it, computed from the configuration's data. The arguments that set an input's value (`value`, `selected`, `start`, and `end`) are left out, and the configuration's namespace is applied to the ids.

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
updateDateInput	Date, POSIXt	<i>default</i>
updateDateRangeInput	Date, POSIXt	<code>range = TRUE</code>
updateNumericInput	numeric	<i>default</i>
updateRadioButtons	character, factor, list, logical	<code>radio = TRUE</code>
updateSelectInput	character, factor, list, logical	<i>default</i>
updateSelectizeInput	character, factor, list, logical	<code>selectize = TRUE</code>
updateSliderInput	numeric	<code>slider = TRUE</code>
updateTextAreaInput	character	<code>textbox = TRUE, area = TRUE</code>
updateTextInput	character	<code>textbox = TRUE</code>

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) {
  observe({
    fruits_filtered <- fruits
    if (!is.null(input$letter) && length(input$letter) != 0L) {
      fruits_filtered <- fruits[input$letter]
    }
  })
}
```

```
# Call updateFilterInput() inside the shiny server:
updateFilterInput(x = fruits_filtered, inputId = "fruits")
})
}
shinyApp(ui, server)
```

with_args

*Set the Arguments of Inputs***Description**

with_args() sets arguments for the inputs [filterInput\(\)](#) creates for one or more columns of a configuration made by [shinyfilters\(\)](#), such as a slider's value, step, or width. Each column keeps its input.

Usage

```
with_args(.filters, ...)
```

Arguments

.filters	A configuration created by shinyfilters() .
...	Any number of formulas. • cols ~ list(arg = value, ...): the columns on the left, their arguments on the right. Arguments replace the ones shinyfilters passes for the column, such as label, choices, min, max, and value. inputId can't be set: an input's id is always its column's name. .update_fn = fn names the function that updates the input instead, as described in with_filters(). A value can use .x, the column the input is created for. It is evaluated when the input is created, and a call in it that takes na.rm, such as range(.x), gets na.rm = TRUE unless it sets na.rm itself. Other values are evaluated right away, and can use the columns too, as they are at that point.

Details

Arguments stay with their column. Setting one again replaces it, and an input chosen later with [with_filters\(\)](#) keeps the ones it has an argument for.

Value

The updated configuration.

See Also

[with_filters\(\)](#) to choose inputs, [with_defaults\(\)](#) to set arguments for every column.

Examples

```
filters <- shinyfilters(nyc_flights)

# A range slider: `.x` is the column the input is created for
filters <- with_args(filters, dep_delay ~ list(value = range(.x)))
filters$dep_delay

# Several arguments
with_args(filters, dep_delay ~ list(value = range(.x), step = 5))

# The same arguments for several columns
with_args(filters, c(dep_delay, distance) ~ list(value = range(.x)))

# Different arguments for different columns
with_args(
  filters,
  dep_delay ~ list(step = 5),
  origin ~ list(label = "Airport")
)
```

with_defaults

Change the Default Arguments of a Configuration

Description

`with_defaults()` adds, replaces, or removes the arguments that a configuration made by `shinyfilters()` passes to `filterInput()` for every column.

Usage

```
with_defaults(.filters, ...)
```

Arguments

<code>.filters</code>	A configuration created by <code>shinyfilters()</code> .
<code>...</code>	Named arguments passed to <code>filterInput()</code> for every column, such as <code>textbox = TRUE</code> or <code>width = "200px"</code> . An argument set to <code>NULL</code> , or an input flag such as <code>slider</code> set to <code>FALSE</code> , <code>filterInput()</code> 's default, is removed. <code>selectize = FALSE</code> is kept: <code>shiny::selectInput()</code> takes it too. Arguments not named here keep their current values.

Value

The updated configuration.

See Also

`shinyfilters()`, `with_filters()`, `with_ns()`

Examples

```
filters <- shinyfilters(nyc_flights, width = "200px")

# Turn a default off
filters <- with_defaults(filters, slider = FALSE)
filters

# Replace one
with_defaults(filters, width = "100%")

# Remove one
with_defaults(filters, width = NULL)
```

with_filters

Choose the Input for Columns

Description

`with_filters()` sets the input that `filterInput()` creates for one or more columns of a configuration made by `shinyfilters()`, and adds or replaces columns computed from the others. When a column's input is chosen more than once, the last one wins; the arguments set for the column stay with it.

Usage

```
with_filters(.filters, ...)
```

Arguments

- | | |
|----------|--|
| .filters | A configuration created by <code>shinyfilters()</code> . |
| ... | <p>Either two unnamed arguments, or any number of named arguments, or formulas:</p> <ul style="list-style-type: none"> • <code>with_filters(.filters, col = input, ...)</code>: each name is a column. • <code>with_filters(.filters, col = expression, ...)</code>: adds or replaces a column, computed from the other columns. A replaced column keeps its input. A function or a single string is always read as an input, and <code>input ~ list(...)</code> as an input with its arguments; any other value is the column's data. A column takes precedence over a variable of the same name. • <code>with_filters(.filters, cols ~ input, ...)</code>: a two-sided formula selects columns on its left, like <code>cols</code> above, and names one input for all of them on its right. It can be mixed with named columns. • <code>with_filters(.filters, cols ~ list(arg = value), ...)</code>: sets arguments of the inputs the columns have, as <code>with_args()</code> does. |

Each input is either a keyword or the **shiny** input function it stands for, such as `shiny::radioButtons()` for "radio". The keywords are "area", "date", "numeric", "radio", "range", "select", "selectize", "slider", and "textbox".

"date", "numeric", and "select" are the inputs columns have by default: use them to opt a column out of a default argument such as `slider = TRUE`. "radio", "select", and "selectize" work with any column: a number or a date uses its sorted unique values as choices, and a datetime its dates.

Other functions are called like `call_filter_input()`: they receive the arguments `args_filter_input()` returns for the column's type, plus any other arguments they accept.

An input takes its arguments with it, wherever an input goes. Write `input ~ list(...)`, such as `col = "slider" ~ list(value = range(.x))` or `cols ~ "slider" ~ list(value = range(.x), step = 5)`. It is read as written and never run; `with_args()` describes the arguments.

`shinyfilters_server()` needs the function that updates an input that isn't one `filterInput()` creates. A **shinyWidgets** input comes with its own, such as `updatePickerInput()` for `pickerInput()`. For any other, name it with `.update_fn = fn` among the arguments, such as `col = checkboxGroupInput ~ list(.update_fn = updateCheckboxGroupInput)`.

Arguments stay with the column: setting it again adds to them, replacing those of the same name, and an input chosen later keeps the ones it has an argument for. They are matched by name only, so set an argument again if its value doesn't suit the new input. `.update_fn` stays with the input instead: a new input drops it.

Value

The updated configuration.

See Also

`shinyfilters()`, `with_args()`

Examples

```
filters <- shinyfilters(nyc_flights)
filters <- with_filters(filters, origin = "radio", carrier = "selectize")
filters

# Choose one input for several columns
filters <- with_filters(filters, dep_delay:origin ~ "slider")

# Give an input its arguments. `.x` is the column the input is for.
with_filters(filters, dep_delay ~ "slider" ~ list(value = range(.x)))

# Add a column computed from the others
with_filters(filters, delay_sq = dep_delay^2)

# Give one column the input it has by default
filters <- shinyfilters(nyc_flights, slider = TRUE)
with_filters(filters, distance = "numeric")
```

`with_ns`*Change the Namespace of a Configuration*

Description

`with_ns()` adds, replaces, or removes the namespace that a configuration made by `shinyfilters()` applies to its input ids. Use it to build a configuration once and reuse it in several modules.

Usage

```
with_ns(.filters, ns)
```

Arguments

<code>.filters</code>	A configuration created by <code>shinyfilters()</code> .
<code>ns</code>	The namespace: a string, used as the id passed to <code>shiny::NS()</code> ; a namespace created by <code>shiny::NS()</code> ; or NULL to remove the namespace. <code>shinyfilters_server()</code> finds the inputs by it, so it is the full namespace of their ids.

Value

The updated configuration.

See Also

`shinyfilters()`

Examples

```
filters <- shinyfilters(nyc_flights)

# Add a namespace
filters <- with_ns(filters, "flights")
filters

# Replace it, here with a namespace created by NS()
with_ns(filters, shiny::NS("departures"))

# Remove it
with_ns(filters, NULL)
```

Index

* datasets

- nyc_flights, 11
- apply_filters, 2
- apply_filters(), 12
- args_filter_input, 3
- args_filter_input(), 4–6, 23
- args_update_filter_input
 - (args_filter_input), 3
- args_update_filter_input(), 18
- bslib::accordion, 17
- call_filter_input
 - (call_input_function), 4
- call_filter_input(), 23
- call_input_function, 4
- call_update_filter_input
 - (call_input_function), 4
- dateInput, 6, 15
- dateRangeInput, 6, 15
- filterInput, 5, 16
- filterInput(), 3–5, 11, 13, 18, 20–23
- get_filter_logical, 7
- get_filter_logical(), 3
- get_input_ids, 8
- get_input_ids(), 10
- get_input_labels, 9
- get_input_values, 10
- NS(), 6, 15
- numericInput, 6, 15
- nyc_flights, 11
- radioButtons, 6, 15
- selectInput, 6, 15
- selectizeInput, 6, 15

- serverFilterInput, 12
- shiny::NS(), 24
- shiny::radioButtons(), 22
- shiny::selectInput(), 21
- shinyfilters, 14, 15
- shinyfilters(), 12, 13, 16, 18, 20–24
- shinyfilters-class, 16
- shinyfilters_server, 16
- shinyfilters_server
 - (serverFilterInput), 12
- shinyfilters_server(), 23, 24
- sliderInput, 6, 15
- sort(), 4
- textAreaInput, 6, 15
- textInput, 6, 15
- unique(), 4
- updateDateInput, 18, 19
- updateDateRangeInput, 18, 19
- updateFilterInput, 18
- updateFilterInput(), 3, 4, 12, 13
- updateNumericInput, 18, 19
- updateRadioButtons, 18, 19
- updateSelectInput, 18, 19
- updateSelectizeInput, 18, 19
- updateSliderInput, 18, 19
- updateTextAreaInput, 18, 19
- updateTextInput, 18, 19
- with_args, 20
- with_args(), 16, 18, 22, 23
- with_defaults, 21
- with_defaults(), 16, 20
- with_filters, 22
- with_filters(), 13, 15, 16, 18, 20, 21
- with_ns, 24
- with_ns(), 12, 15, 16, 21