

# Package ‘teal.slice’

December 2, 2025

**Type** Package

**Title** Filter Module for 'teal' Applications

**Version** 0.7.1

**Date** 2025-12-01

**Description** Data filtering module for 'teal' applications. Allows for interactive filtering of data stored in 'data.frame' and 'MultiAssayExperiment' objects. Also displays filtered and unfiltered observation counts.

**License** Apache License 2.0

**URL** <https://insightsengineering.github.io/teal.slice/>,  
<https://github.com/insightsengineering/teal.slice/>

**BugReports** <https://github.com/insightsengineering/teal.slice/issues>

**Depends** R (>= 4.0)

**Imports** bsicons (>= 0.1.2), bslib (>= 0.8.0), checkmate (>= 2.1.0), dplyr (>= 1.0.5), grDevices, htmltools (>= 0.5.4), jsonlite (>= 2.0.0), logger (>= 0.4.0), methods, plotly (>= 4.10.4), R6 (>= 2.2.0), rlang (>= 1.0.0), shiny (>= 1.8.1), shinycssloaders (>= 1.0.0), shinyjs (>= 2.1.0), shinyWidgets (>= 0.6.2), teal.data (>= 0.8.0), teal.logger (>= 0.4.0), teal.widgets (>= 0.5.0), utils

**Suggests** DT (>= 0.13), knitr (>= 1.42), MultiAssayExperiment, rmarkdown (>= 2.23), rvest (>= 1.0.0), shinytest2 (>= 0.4.1), SummarizedExperiment, testthat (>= 3.2.2), withr (>= 3.0.2)

**VignetteBuilder** knitr, rmarkdown

**Config/Needs/verdepcheck** rstudio/shiny, rstudio/bslib, mllg/checkmate, tidyverse/dplyr, rstudio/htmltools, jeroen/jsonlite, daroczig/logger, plotly/plotly, r-lib/R6, daattali/shinycssloaders, daattali/shinyjs, dreamRs/shinyWidgets, insightsengineering/teal.data, insightsengineering/teal.logger, insightsengineering/teal.widgets, rstudio/DT, yihui/knitr,

bioc::MultiAssayExperiment, bioc::SummarizedExperiment,  
rstudio/rmarkdown, r-lib/testthat, r-lib/withr,  
bioc::matrixStats

**Config/Needs/website** insightsengineering/nesttemplate

**Encoding** UTF-8

**Language** en-US

**RoxygenNote** 7.3.3

**NeedsCompilation** no

**Author** Dawid Kaledkowski [aut, cre] (ORCID:  
<<https://orcid.org/0000-0001-9533-457X>>),  
Pawel Rucki [aut],  
Aleksander Chlebowski [aut] (ORCID:  
<<https://orcid.org/0000-0001-5018-6294>>),  
Andre Verissimo [aut] (ORCID: <<https://orcid.org/0000-0002-2212-339X>>),  
Kartikeya Kirar [aut],  
Marcin Kosinski [aut],  
Chendi Liao [rev],  
Dony Unardi [rev],  
Andrew Bates [aut],  
Mahmoud Hallal [aut],  
Nikolas Burkoff [aut],  
Maciej Nasinski [aut],  
Konrad Pagacz [aut],  
Junlue Zhao [aut],  
F. Hoffmann-La Roche AG [cph, fnd]

**Maintainer** Dawid Kaledkowski <dawid.kaledkowski@roche.com>

**Repository** CRAN

**Date/Publication** 2025-12-02 07:50:02 UTC

Contents

FilterPanelAPI . . . . .	3
filter_state_api . . . . .	5
get_filter_expr . . . . .	7
init_filtered_data . . . . .	8
teal_slice . . . . .	9
teal_slices . . . . .	11
<b>Index</b>	<b>15</b>

---

FilterPanelAPI	<i>Class to encapsulate the API of the filter panel of a teal app</i>
----------------	---

---

## Description

An API class for managing filter states in a teal application's filter panel.

## Details

The purpose of this class is to encapsulate the API of the filter panel in a new class `FilterPanelAPI` so that it can be passed and used in the server call of any module instead of passing the whole `FilteredData` object.

This class is supported by methods to set, get, remove filter states in the filter panel API.

## Methods

### Public methods:

- `FilterPanelAPI$new()`
- `FilterPanelAPI$get_filter_state()`
- `FilterPanelAPI$set_filter_state()`
- `FilterPanelAPI$remove_filter_state()`
- `FilterPanelAPI$clear_filter_states()`
- `FilterPanelAPI$clone()`

**Method** `new()`: Initialize a `FilterPanelAPI` object.

*Usage:*

```
FilterPanelAPI$new(datasets)
```

*Arguments:*

`datasets` (`FilteredData`)

**Method** `get_filter_state()`: Gets the reactive values from the active `FilterState` objects of the `FilteredData` object.

Gets all active filters in the form of a nested list. The output list is a compatible input to `set_filter_state`.

*Usage:*

```
FilterPanelAPI$get_filter_state()
```

*Returns:* list with named elements corresponding to `FilteredDataset` objects with active filters.

**Method** `set_filter_state()`: Sets active filter states.

*Usage:*

```
FilterPanelAPI$set_filter_state(filter)
```

*Arguments:*

`filter` (`teal_slices`)

*Returns:* NULL, invisibly.

**Method** `remove_filter_state()`: Remove one or more FilterState of a FilteredDataset in the FilteredData object.

*Usage:*

```
FilterPanelAPI$remove_filter_state(filter)
```

*Arguments:*

`filter` (teal\_slices) specifying FilterState objects to remove; teal\_slices may contain only dataname and varname, other elements are ignored

*Returns:* NULL, invisibly.

**Method** `clear_filter_states()`: Remove all FilterStates of the FilteredData object.

*Usage:*

```
FilterPanelAPI$clear_filter_states(datanames)
```

*Arguments:*

`datanames` (character) datanames to remove their FilterStates; omit to remove all FilterStates in the FilteredData object

*Returns:* NULL, invisibly.

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
FilterPanelAPI$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
library(shiny)

fd <- init_filtered_data(list(iris = iris))
fpa <- FilterPanelAPI$new(fd)

# get the actual filter state --> empty named list
isolate(fpa$get_filter_state())

# set a filter state
set_filter_state(
  fpa,
  teal_slices(
    teal_slice(dataname = "iris", varname = "Species", selected = "setosa", keep_na = TRUE)
  )
)

# get the actual filter state --> named list with filters
isolate(fpa$get_filter_state())

# remove all_filter_states
```

```
fpa$clear_filter_states()

# get the actual filter state --> empty named list
isolate(fpa$get_filter_state())
```

---

filter_state_api	<i>Managing FilteredData states</i>
------------------	-------------------------------------

---

## Description

Set, get and remove filter states of FilteredData object.

## Usage

```
set_filter_state(datasets, filter)

get_filter_state(datasets)

remove_filter_state(datasets, filter)

clear_filter_states(datasets, force = FALSE)
```

## Arguments

datasets	(FilteredData) object to store filter state and filtered datasets, shared across modules see <a href="#">FilteredData</a> for details
filter	(teal_slices) specify filters in place on app start-up
force	(logical(1)) flag specifying whether to include anchored filter states.

## Value

- set\_\*, remove\_\* and clear\_filter\_state return NULL invisibly
- get\_filter\_state returns a named teal\_slices object containing a teal\_slice for every existing FilterState

## See Also

[teal\\_slice](#)

## Examples

```

datasets <- init_filtered_data(list(iris = iris, mtcars = mtcars))
fs <- teal_slices(
  teal_slice(dataname = "iris", varname = "Species", selected = c("setosa", "versicolor")),
  teal_slice(dataname = "iris", varname = "Sepal.Length", selected = c(5.1, 6.4)),
  teal_slice(dataname = "mtcars", varname = "gear", selected = c(4, 5)),
  teal_slice(dataname = "mtcars", varname = "carb", selected = c(4, 10))
)

# set initial filter state
set_filter_state(datasets, filter = fs)

# get filter state
get_filter_state(datasets)

# modify filter state
set_filter_state(
  datasets,
  teal_slices(
    teal_slice(dataname = "iris", varname = "Species", selected = "setosa", keep_na = TRUE)
  )
)

# remove specific filters
remove_filter_state(
  datasets,
  teal_slices(
    teal_slice(dataname = "iris", varname = "Species"),
    teal_slice(dataname = "mtcars", varname = "gear"),
    teal_slice(dataname = "mtcars", varname = "carb")
  )
)

# remove all states
clear_filter_states(datasets)

if (requireNamespace("MultiAssayExperiment", quietly = TRUE)) {
  # Requires MultiAssayExperiment from Bioconductor
  data(miniACC, package = "MultiAssayExperiment")

  datasets <- init_filtered_data(list(mae = miniACC))
  fs <- teal_slices(
    teal_slice(
      dataname = "mae", varname = "years_to_birth", selected = c(30, 50),
      keep_na = TRUE, keep_inf = FALSE
    ),
    teal_slice(
      dataname = "mae", varname = "vital_status", selected = "1",
      keep_na = FALSE
    ),
    teal_slice(

```

```

        dataname = "mae", varname = "gender", selected = "female",
        keep_na = TRUE
    ),
    teal_slice(
        dataname = "mae", varname = "ARRAY_TYPE", selected = "",
        keep_na = TRUE, experiment = "RPPAArray", arg = "subset"
    )
)

# set initial filter state
set_filter_state(datasets, filter = fs)

# get filter state
get_filter_state(datasets)

# modify filter state
set_filter_state(
    datasets,
    teal_slices(
        teal_slice(dataname = "mae", varname = "years_to_birth", selected = c(40, 60))
    )
)

# remove specific filters
remove_filter_state(
    datasets,
    teal_slices(
        teal_slice(dataname = "mae", varname = "years_to_birth"),
        teal_slice(dataname = "mae", varname = "vital_status")
    )
)

# remove all states
clear_filter_states(datasets)
}

```

---

get_filter_expr	<i>Gets filter expression for multiple datanames taking into account its order.</i>
-----------------	---

---

## Description

To be used in Show R Code button.

## Usage

```
get_filter_expr(datasets, datanames = datasets$datanames())
```

**Arguments**

datasets            (FilteredData)  
datanames           (character) vector of dataset names

**Value**

A character string containing all subset expressions.

---

init_filtered_data	<i>Initialize</i> FilteredData
--------------------	--------------------------------

---

**Description**

Function creates a FilteredData object.

**Usage**

```
init_filtered_data(x, join_keys = teal.data::join_keys())
```

**Arguments**

x                    (named list) of datasets.  
join\_keys            (join\_keys) see [teal.data::join\\_keys\(\)](#).

**Value**

Object of class FilteredData.

**Examples**

```
datasets <- init_filtered_data(list(iris = iris, mtcars = mtcars))  
datasets
```



---

teal_slice	<i>Specify single filter</i>
------------	------------------------------

---

## Description

Create a teal\_slice object that holds complete information on filtering one variable. Check out [teal\\_slice-utilities](#) functions for working with teal\_slice object.

## Usage

```
teal_slice(
  dataname,
  varname,
  id,
  expr,
  choices = NULL,
  selected = NULL,
  keep_na = NULL,
  keep_inf = NULL,
  fixed = FALSE,
  anchored = FALSE,
  multiple = TRUE,
  title = NULL,
  ...
)
```

## Arguments

dataname	(character(1)) name of data set
varname	(character(1)) name of variable
id	(character(1)) identifier of the filter. Must be specified when expr is set. When varname is specified then id is set to "{dataname} {varname}" by default.
expr	(character(1)) string providing a logical expression. Must be a valid R expression which can be evaluated in the context of the data set. For a data.frame var == "x" is sufficient, but MultiAssayExperiment::subsetByColData requires dataname prefix, e.g. data\$var == "x".
choices	(vector) optional, specifies allowed choices; When specified it should be a subset of values in variable denoted by varname; Type and size depends on variable type. Factors are coerced to character.
selected	(vector) optional, specifies selected values from choices; Type and size depends on variable type. Factors are coerced to character.
keep_na	(logical(1)) optional flag specifying whether to keep missing values
keep_inf	(logical(1)) optional flag specifying whether to keep infinite values
fixed	(logical(1)) flag specifying whether to fix this filter state (forbid setting state)

anchored	(logical(1)) flag specifying whether to lock this filter state (forbid removing and inactivating)
multiple	(logical(1)) optional flag specifying whether more than one value can be selected; only applicable to ChoicesFilterState and LogicalFilterState
title	(character(1)) optional title of the filter. Ignored when varname is set.
...	additional arguments which can be handled by extensions of teal.slice classes.

## Details

teal\_slice object fully describes filter state and can be used to create, modify, and delete a filter state. A teal\_slice contains a number of common fields (all named arguments of teal\_slice), some of which are mandatory, but only dataname and either varname or expr must be specified, while the others have default values.

Setting any of the other values to NULL means that those properties will not be modified (when setting an existing state) or that they will be determined by data (when creating new a new one). Entire object is FilterState class member and can be accessed with FilterState\$get\_state().

A teal\_slice can come in two flavors:

1. teal\_slice\_var - this describes a typical interactive filter that refers to a single variable, managed by the FilterState class. This class is created when varname is specified. The object retains all fields specified in the call. id can be created by default and need not be specified.
2. teal\_slice\_expr - this describes a filter state that refers to an expression, which can potentially include multiple variables, managed by the FilterStateExpr class. This class is created when expr is specified. dataname and anchored are retained, fixed is set to TRUE, id becomes mandatory, title remains optional, while other arguments are disregarded.

A teal\_slice can be passed FilterState/FilterStateExpr constructors to instantiate an object. It can also be passed to FilterState\$set\_state to modify the state. However, once a FilterState is created, only the mutable features can be set with a teal\_slice: selected, keep\_na and keep\_inf.

Special consideration is given to two fields: fixed and anchored. These are always immutable logical flags that default to FALSE. In a FilterState instantiated with fixed = TRUE the features selected, keep\_na, keep\_inf cannot be changed. Note that a FilterStateExpr is always considered to have fixed = TRUE. A FilterState instantiated with anchored = TRUE cannot be removed.

## Value

A teal.slice object. Depending on whether varname or expr was specified, the resulting teal\_slice also receives class teal\_slice\_var or teal\_slice\_expr, respectively.

## Filters in SumarizedExperiment and MultiAssayExperiment objects

To establish a filter on a column in a data.frame, dataname and varname are sufficient. MultiAssayExperiment objects can be filtered either on their colData slot (which contains subject information) or on their experiments, which are stored in the experimentList slot. For filters referring to colData no extra arguments are needed. If a filter state is created for an experiment, that experiment name must be specified in the experiment argument. Furthermore, to specify filter for an SumarizedExperiment one must also set arg ("subset" or "select", arguments in the subset() function for SumarizedExperiment) in order to determine whether the filter refers to the SE's rowData or colData.

**Note**

Date time objects of POSIX\*t classes are printed as strings after converting to UTC timezone.

**See Also**

[teal\\_slices](#), [is.teal\\_slice](#), [as.teal\\_slice](#), [as.list.teal\\_slice](#), [print.teal\\_slice](#), [format.teal\\_slice](#)

**Examples**

```
x1 <- teal_slice(
  dataname = "data",
  id = "Female adults",
  expr = "SEX == 'F' & AGE >= 18",
  title = "Female adults"
)
x2 <- teal_slice(
  dataname = "data",
  varname = "var",
  choices = c("F", "M", "U"),
  selected = "F",
  keep_na = TRUE,
  keep_inf = TRUE,
  fixed = FALSE,
  anchored = FALSE,
  multiple = TRUE,
  id = "Gender",
  extra_arg = "extra"
)

is.teal_slice(x1)
as.list(x1)
as.teal_slice(list(dataname = "a", varname = "var"))
format(x1)
format(x1, show_all = TRUE, trim_lines = FALSE)
print(x1)
print(x1, show_all = TRUE, trim_lines = FALSE)
```

---

teal\_slices

---

*Complete filter specification*


---

**Description**

Create teal\_slices object to package multiple filters and additional settings. Check out [teal\\_slices-utilities](#) functions for working with teal\_slices object.

**Usage**

```
teal_slices(
  ...,
  exclude_varnames = NULL,
  include_varnames = NULL,
  count_type = NULL,
  allow_add = TRUE
)
```

**Arguments**

...	any number of teal_slice objects.
include_varnames, exclude_varnames	(named lists of character) where list names match names of data sets and vector elements match variable names in respective data sets; specify which variables are allowed to be filtered; see Details.
count_type	<p><i>This is a new feature. Do kindly share your opinions on <a href="#">teal.slice's GitHub repository</a>.</i></p> <p>(character(1)) string specifying how observations are tallied by these filter states. Possible options:</p> <ul style="list-style-type: none"> <li>• "none" (default) to have counts of single FilterState to show unfiltered number only.</li> <li>• "all" to have counts of single FilterState to show number of observation in filtered and unfiltered dataset. Note, that issues were reported when using this option with MultiAssayExperiment. Please make sure that adding new filters doesn't fail on target platform before deploying for production.</li> </ul>
allow_add	(logical(1)) logical flag specifying whether the user will be able to add new filters

**Details**

teal\_slices() collates multiple teal\_slice objects into a teal\_slices object, a complete filter specification. This is used by all classes above FilterState as well as filter\_panel\_api wrapper functions. teal\_slices has attributes that modify the behavior of the filter panel, which are resolved by different classes.

include\_varnames and exclude\_varnames determine which variables can have filters assigned. The former enumerates allowed variables, the latter enumerates forbidden values. Since these could be mutually exclusive, it is impossible to set both allowed and forbidden variables for one data set in one teal\_slices.

**Value**

teal\_slices, which is an unnamed list of teal\_slice objects.

**See Also**

- [teal\\_slice](#) for creating constituent elements of teal\_slices
- `teal::slices_store` for robust utilities for saving and loading teal\_slices in JSON format
- [is.teal\\_slices](#), [as.teal\\_slices](#), [as.list.teal\\_slices](#), [\[.teal\\_slices](#)], [c.teal\\_slices](#), [print.teal\\_slices](#), [format.teal\\_slices](#)

**Examples**

```

filter_1 <- teal_slice(
  dataname = "dataname1",
  varname = "varname1",
  choices = letters,
  selected = "b",
  keep_na = TRUE,
  fixed = FALSE,
  extra1 = "extraone"
)
filter_2 <- teal_slice(
  dataname = "dataname1",
  varname = "varname2",
  choices = 1:10,
  keep_na = TRUE,
  selected = 2,
  fixed = TRUE,
  anchored = FALSE,
  extra2 = "extratwo"
)
filter_3 <- teal_slice(
  dataname = "dataname2",
  varname = "varname3",
  choices = 1:10 / 10,
  keep_na = TRUE,
  selected = 0.2,
  fixed = TRUE,
  anchored = FALSE,
  extra1 = "extraone",
  extra2 = "extratwo"
)

all_filters <- teal_slices(
  filter_1,
  filter_2,
  filter_3,
  exclude_varnames = list(
    "dataname1" = "varname2"
  )
)

is.teal_slices(all_filters)
all_filters[1:2]
c(all_filters[1], all_filters[2])

```

```
print(all_filters)
print(all_filters, trim_lines = FALSE)
```

# Index

`as.list.teal_slice`, [11](#)  
`as.list.teal_slices`, [13](#)  
`as.teal_slice`, [11](#)  
`as.teal_slices`, [13](#)  
  
`c.teal_slices`, [13](#)  
`clear_filter_states` (`filter_state_api`),  
[5](#)  
  
`filter_state_api`, [5](#)  
`FilteredData`, [5](#)  
`FilterPanelAPI`, [3](#)  
`format.teal_slice`, [11](#)  
`format.teal_slices`, [13](#)  
  
`get_filter_expr`, [7](#)  
`get_filter_state` (`filter_state_api`), [5](#)  
  
`init_filtered_data`, [8](#)  
`is.teal_slice`, [11](#)  
`is.teal_slices`, [13](#)  
  
`print.teal_slice`, [11](#)  
`print.teal_slices`, [13](#)  
  
`remove_filter_state` (`filter_state_api`),  
[5](#)  
  
`set_filter_state` (`filter_state_api`), [5](#)  
`subset()`, [10](#)  
  
`teal.data::join_keys()`, [8](#)  
`teal_slice`, [5](#), [9](#), [13](#)  
`teal_slices`, [11](#), [11](#)