Package ‘teal.transform’

February 17, 2024

Title Functions for Extracting and Merging Data in the ‘teal’ Framework

Version 0.5.0

Date 2024-02-16

Description A standardized user interface for column selection, that facilitates dataset merging in ‘teal’ framework.

License Apache License 2.0

URL https://insightsengineering.github.io/teal.transform/,
     https://github.com/insightsengineering/teal.transform/

BugReports https://github.com/insightsengineering/teal.transform/issues

Depends R (>= 3.6)

Imports checkmate (>= 2.1.0), dplyr (>= 1.1.0), lifecycle (>= 0.2.0),
       logger (>= 0.2.0), methods, rlang (>= 1.0.0), shiny (>= 1.6.0),
       shinyjs, shinyvalidate, stats, teal.data (>= 0.5.0),
       teal.logger (>= 0.1.1), teal.widgets (>= 0.4.0), tidyr (>=
       0.8.3), tidyselect

Suggests knitr (>= 1.42), rmarkdown (>= 2.19), teal.code (>= 0.5.0),
       testthat (>= 3.1.5)

VignetteBuilder knitr

RdMacros lifecycle

Config/Needs/verdepcheck mllg/checkmate, tidyverse/dplyr,
      r-lib/lifecycle, daroczig/logger, r-lib/rlang,
      rstudio/rmarkdown, rstudio/shiny, daattali/shinyjs,
      rstudio/shinyvalidate, insightsengineering/teal.data,
      insightsengineering/teal.logger,
      insightsengineering/teal.widgets, tidyverse/tidyrl,
      r-lib/tidyselect, yihui/knitr, insightsengineering/teal.code,
      r-lib/testthat

Config/Needs/website insightsengineering/nesttemplate

Encoding UTF-8
Language  en-US
LazyData  true
RoxygenNote  7.3.1
NeedsCompilation  no
Author  Dawid Kaledkowski [aut, cre],
       Pawel Rucki [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  2024-02-17 21:30:02 UTC

R topics documented:

add_no_selected_choices ........................................... 3
all_choices .............................................................. 3
check_no_multiple_selection ........................................... 4
choices_labeled ............................................................ 5
choices_selected ........................................................... 6
compose_and_enable_validators ......................................... 9
datanames_input .......................................................... 11
data_extract_multiple_srv ............................................. 12
data_extract_spec ......................................................... 15
data_extract_srv ......................................................... 17
data_extract_ui ........................................................... 21
filter_spec ............................................................... 23
format_data_extract ..................................................... 25
get_anl_relabel_call .................................................... 26
get_dataset_prefixed_col_names ....................................... 27
get_extract_datanames .................................................. 27
get_merge_call ............................................................ 28
get_relabel_call .......................................................... 29
is_single_dataset ......................................................... 29
list_extract_spec .......................................................... 30
merge_datasets ............................................................ 30
merge_expression_module ............................................... 33
merge_expression_srv ..................................................... 36
no_selected_as_NULL ....................................................... 40
resolve_delayed ........................................................... 40
select_spec ............................................................... 42
split_by_sep ............................................................... 45
value_choices ............................................................. 45
add_no_selected_choices

Add empty choice to choices selected

Description

[Stable]

Usage

add_no_selected_choices(x, multiple = FALSE)

Arguments

x (choices_selected) object.

multiple (logical(1)) whether multiple selections are allowed or not.

Value

choices_selected object with an empty option added to the choices.

all_choices

Bare constructor for all_choices object

Description

[Experimental]

An S3 structure representing the selection of all possible choices in a filter_spec, select_spec or choices_selected object.

Usage

all_choices()

Value

all_choices object.
Examples

# Both structures are semantically identical
filter_spec(
    vars = c("selected_variable"),
    choices = c("value1", "value2"),
    selected = c("value1", "value2")
)

filter_spec(
    vars = c("selected_variable"),
    choices = c("value1", "value2"),
    selected = all_choices()
)

choices_selected(choices = letters, selected = letters)
choices_selected(choices = letters, selected = all_choices())

check_no_multiple_selection

Checks that the extract_input specification does not allow multiple selection

Description

[Stable]

Usage

check_no_multiple_selection(extract_input)

Arguments

extract_input (list or NULL) a list of data_extract_spec

Details

Stops if condition not met.

Value

Raises an error when check fails, otherwise, it returns NULL, invisibly.
choices_labeled

Set "<choice>:<label>" type of names

Description

[Stable]

This is often useful for choices_selected() as it marks up the drop-down boxes for shiny::selectInput().

Usage

choices_labeled(choices, labels, subset = NULL, types = NULL)

## S3 method for class 'choices_labeled'
print(x, ...)

Arguments

choices      (character or factor or numeric or logical) vector.
labels       (character) vector containing labels to be applied to choices. If NA then "Label Missing" will be used.
subset       (character or factor or numeric or logical) vector that is a subset of choices. This is useful if only a few variables need to be named. If this argument is used, the returned vector will match its order.
types        (character) vector containing the types of the columns to be used for applying the appropriate icons to the choices_selected drop down box (e.g. "numeric").
x            an object used to select a method.
...         further arguments passed to or from other methods.

Details

If either choices or labels are factors, they are coerced to character. Duplicated elements from choices get removed.

Value

Named character vector.

Methods (by generic)

- print(choices_labeled): Print choices_labeled object
Examples

```r
library(shiny)
library(teal.data)

ADSL <- teal.transform::rADSL
ADTTE <- teal.transform::rADTTE

choices1 <- choices_labeled(names(ADSL), col_labels(ADSL, fill = FALSE))
choices2 <- choices_labeled(ADTTE$PARAMCD, ADTTE$PARAM)

# if only a subset of variables are needed, use subset argument
choices3 <- choices_labeled(
  names(ADSL),
  col_labels(ADSL, fill = FALSE),
  subset = c("ARMCD", "ARM")
)

ui <- fluidPage(
  selectInput("c1",
    label = "Choices from ADSL",
    choices = choices1,
    selected = choices1[1]
  ),
  selectInput("c2",
    label = "Choices from ADTTE",
    choices = choices2,
    selected = choices2[1]
  ),
  selectInput("c3",
    label = "Arm choices from ADSL",
    choices = choices3,
    selected = choices3[1]
  )
)
server <- function(input, output) {}

if (interactive()) {
  shinyApp(ui, server)
}
```

---

**choices_selected**

**Choices selected**

**Description**

[Stable]

Construct a single list containing available choices, the default selected value, and additional settings such as to order the choices with the selected elements appearing first or whether to block the user from making selections.

Can be used in UI input elements such as `teal.widgets::optionalSelectInput()`.
choices_selected

Usage

choices_selected(
  choices,
  selected = if (inherits(choices, "delayed_data")) NULL else choices[1],
  keep_order = FALSE,
  fixed = FALSE
)

is.choices_selected(x)

Arguments

choices (character) vector of possible choices or delayed_data object. See variable_choices() and value_choices().

selected (character) vector of preselected options, (all_choices) object or (delayed_data) object. If delayed_data object then choices must also be delayed_data object. If not supplied it will default to the first element of choices if choices is a vector, or NULL if choices is a delayed_data object.

keep_order (logical) In case of FALSE the selected variables will be on top of the drop-down field.

fixed (optional logical) Whether to block user to select choices.

x (choices_selected) object to check.

Details

Please note that the order of selected will always follow the order of choices. The keep_order argument is set to false which will run the following code inside:

choices <- c(selected, setdiff(choices, selected))

In case you want to keep your specific order of choices, set keep_order to TRUE.

Value

choices_selected returns list of choices_selected, encapsulating the specified choices, selected, keep_order and fixed.

is.choices_selected returns TRUE if x inherits from a choices_selected object, FALSE otherwise.

Functions

- is.choices_selected(): Check if an object is a choices_selected class
Examples

library(shiny)
library(teal.widgets)

# all_choices example - semantically the same objects
choices_selected(choices = letters, selected = all_choices())
choices_selected(choices = letters, selected = letters)

choices_selected(
    choices = setNames(LETTERS[1:5], paste("Letter", LETTERS[1:5])),
    selected = "C"
)

ADSL <- teal.transform::rADSL
choices_selected(variable_choices(ADSL), "SEX")

# How to select nothing
# use an empty character
choices_selected(
    choices = c("", "A", "B", "C"),
    selected = ""
)

# How to allow the user to select nothing
# use an empty character
choices_selected(
    choices = c("A", "", "B", "C"),
    selected = "A"
)

# How to make Nothing the Xth choice
# just use keep_order
choices_selected(
    choices = c("A", "", "B", "C"),
    selected = "A",
    keep_order = TRUE
)

# How to give labels to selections
# by adding names - choices will be replaced by "name" in UI, not in code
choices_selected(
    choices = c("name for A" = "A", "Name for nothing" = "", "name for b" = "B", "name for C" = "C"),
    selected = "A"
)

# by using choices_labeled
# labels will be shown behind the choice
choices_selected(
    choices = choices_labeled(c("A", "", "B", "C"),
compose_and_enable_validators

Function to compose validators from data_extract_multiple_srv

---

**Description**

This function takes the output from data_extract_multiple_srv and collates the shinyvalidate::InputValidator returned into a single validator and enables this.

**Usage**

compose_and_enable_validators(iv, selector_list, validator_names = NULL)
compose_and_enable_validators

Arguments

- **iv** (shinyvalidate::InputValidator) A validator.
- **selector_list** (reactive named list of reactives). Typically this is the output from data_extract_multiple_srv. The validators in this list (specifically selector_list()[[validator_names]](iv)) will be added into iv.
- **validator_names** (character or NULL). If character then only validators in the elements of selector_list() whose name is in this list will be added. If NULL all validators will be added.

Value

(shinyvalidate::InputValidator) enabled iv with appropriate validators added into it.

Examples

```r
library(shiny)
library(shinyvalidate)
library(shinyjs)
library(teal.widgets)

iris_extract <- data_extract_spec(
  dataname = "iris",
  select = select_spec(
    label = "Select variable:",
    choices = variable_choices(iris, colnames(iris)),
    selected = "Sepal.Length",
    multiple = TRUE,
    fixed = FALSE
  )
)

data_list <- list(iris = reactive(iris))

ui <- fluidPage(
  useShinyjs(),
  standard_layout(
    output = verbatimTextOutput("out1"),
    encoding = tagList(
      data_extract_ui(
        id = "x_var",
        label = "Please select an X column",
        data_extract_spec = iris_extract
      ),
      data_extract_ui(
        id = "y_var",
        label = "Please select a Y column",
        data_extract_spec = iris_extract
      ),
      data_extract_ui(
        id = "col_var",
        label = "Please select a column",
        data_extract_spec = iris_extract
      )
    )
  )
)
server <- function(input, output, session) {
  exactly_2_validation <- function() {
    if (length(.) != 2) "Exactly 2 'Y' column variables must be chosen"
  }

  selector_list <- data_extract_multiple_srv(
    list(x_var = iris_extract, y_var = iris_extract, col_var = iris_extract),
    datasets = data_list,
    select_validation_rule = list(
      x_var = sv_required("Please select an X column"),
      y_var = compose_rules(
        sv_required("Exactly 2 'Y' column variables must be chosen"),
        exactly_2_validation()
      )
    )
  )

  iv_r <- reactive({
    iv <- InputValidator$new()
    compose_and_enable_validators(
      iv,
      selector_list,
      # if validator_names = NULL then all validators are used
      # to turn on only "x_var" then set this argument to "x_var"
      validator_names = NULL
    )
  })

  output$out1 <- renderPrint({
    if (iv_r$is_valid()) {
      ans <- lapply(selector_list(), function(x) {
        cat(format_data_extract(x()), "\n\n")
      })
    } else {
      "Check that you have made a valid selection"
    }
  })
}

if (interactive()) {
  shinyApp(ui, server)
}
**Description**

[Stable]

Creates `shiny::helpText()` with the names of available datasets for the current module.

**Usage**

datnames_input(data_extracts)

**Arguments**

data_extracts (list) of data extracts for single variable.

**Value**

`shiny.tag` defining help-text element that can be added to a UI element.

---

**data_extract_multiple_srv**

*Creates a named list of data_extract_srv output*

---

**Description**

[Experimental]

data_extract_multiple_srv loops over the list of data_extract given and runs data_extract_srv for each one returning a list of reactive objects.

**Usage**

data_extract_multiple_srv(data_extract, datasets, ...)

## S3 method for class 'reactive'
data_extract_multiple_srv(data_extract, datasets, ...)

## S3 method for class 'FilteredData'
data_extract_multiple_srv(data_extract, datasets, ...)

## S3 method for class 'list'
data_extract_multiple_srv(  
data_extract,  
datasets,  
join_keys = NULL,  
select_validation_rule = NULL,  
filter_validation_rule = NULL,  
dataset_validation_rule = if (is.null(select_validation_rule) &&  
is.null(filter_validation_rule)) {  
  NULL


Arguments

**data_extract**  (named list of data_extract_spec objects) the list data_extract_spec objects. The names of the elements in the list need to correspond to the ids passed to data_extract_ui. See example for details.

**datasets**  (FilteredData or list of reactive or non-reactive data.frame) object containing data either in the form of FilteredData or as a list of data.frame. When passing a list of non-reactive data.frame objects, they are converted to reactive data.frames internally. When passing a list of reactive or non-reactive data.frame objects, the argument join_keys is required also.

... An additional argument join_keys is required when datasets is a list of data.frame. It shall contain the keys per dataset in datasets.

**join_keys**  (join_keys or NULL) of join keys per dataset in datasets.

**select_validation_rule**  (NULL or function or named list of function) Should there be any shinyvalidate input validation of the select parts of the data_extract_ui. If all data_extract require the same validation function then this can be used directly (i.e. select_validation_rule = shinyvalidate::sv_required()). For more fine-grained control use a list: select_validation_rule = list(extract_1 = sv_required(), extract2 = ~ if (length(.x) > 2) "Error") If NULL then no validation will be added. See example for more details.

**filter_validation_rule**  (NULL or function or named list of function) Same as select_validation_rule but for the filter (values) part of the data_extract_ui.

**dataset_validation_rule**  (NULL or function or named list of function) Same as select_validation_rule but for the choose dataset part of the data_extract_ui

Value

reactive named list containing outputs from data_extract_srv(). Output list names are the same as data_extract input argument.

Examples

library(shiny)
library(shinyvalidate)
library(shinyjs)
library(teal.widgets)

iris_select <- data_extract_spec(
  dataname = "iris",
  select = select_spec(
    label = "Select variable:",
    choices = variable_choices(iris, colnames(iris)),
    selected = "Sepal.Length",
    multiple = TRUE,
    fixed = FALSE
  )
)

iris_filter <- data_extract_spec(
  dataname = "iris",
  filter = filter_spec(
    vars = "Species",
    choices = c("setosa", "versicolor", "virginica"),
    selected = "setosa",
    multiple = TRUE
  )
)

data_list <- list(iris = reactive(iris))

ui <- fluidPage(
  useShinyjs(),
  standard_layout(
    output = verbatimTextOutput("out1"),
    encoding = tagList(
      data_extract_ui(
        id = "x_var",
        label = "Please select an X column",
        data_extract_spec = iris_select
      ),
      data_extract_ui(
        id = "species_var",
        label = "Please select 2 Species",
        data_extract_spec = iris_filter
      )
    )
  )
)

server <- function(input, output, session) {
  exactly_2_validation <- function(msg) {
    if (length(.) != 2) msg
  }

  selector_list <- data_extract_multiple_srv(
    list(x_var = iris_select, species_var = iris_filter),
    exactly_2_validation
  )

  # The rest of the server code...
}
data_extract_spec

```r
datasets = data_list,
select_validation_rule = list(
  x_var = sv_required("Please select an X column")
),
filter_validation_rule = list(
  species_var = compose_rules(
    sv_required("Exactly 2 Species must be chosen"),
    exactly_2_validation("Exactly 2 Species must be chosen")
  )
)
iv_r <- reactive({
  iv <- InputValidator$new()
  compose_and_enable_validators(
    iv,
    selector_list,
    validator_names = NULL
  )
})
output$out1 <- renderPrint({
  if (iv_r$is_valid()) {
    ans <- lapply(selector_list(), function(x) {
      cat(format_data_extract(x()), "\n"
    })
  } else {
    "Please fix errors in your selection"
  }
})
if (interactive()) {
  shinyApp(ui, server)
}
```

---

**data_extract_spec**  
*Data extract input for teal modules*

**Description**

[Stable]

The Data extract input can be used to filter and select columns from a data set. This function enables such an input in teal. Please use the constructor function `data_extract_spec` to set it up.

**Usage**

```r
data_extract_spec(dataname, select = NULL, filter = NULL, reshape = FALSE)
```
Arguments

dataname  (character) The name of the dataset to be extracted.
select (NULL or select_spec-S3 class or delayed_select_spec) Columns to be selected from the input dataset mentioned in dataname. The setup can be created using select_spec function.
filter (NULL or filter_spec or its respective delayed version) Setup of the filtering of key columns inside the dataset. This setup can be created using the filter_spec function. Please note that if both select and filter are set to NULL, then the result will be a filter spec UI with all variables as possible choices and a select spec with multiple set to TRUE.
reshape (logical) whether reshape long to wide. Note that it will be used only in case of long dataset with multiple keys selected in filter part.

Value

data_extract_spec object.

Module Development

tearl.transform uses this object to construct a UI element in a module.

Note

No checks based on columns can be done because the data is only referred to by name.

References

select_spec filter_spec

Examples

adtte_filters <- filter_spec(
  vars = c("PARAMCD", "CNSR"),
  sep = "-",
  choices = c("OS-1" = "OS-1", "OS-0" = "OS-0", "PFS-1" = "PFS-1"),
  selected = "OS-1",
  multiple = FALSE,
  label = "Choose endpoint and Censor"
)

data_extract_spec(
  dataname = "ADTTE",
  filter = adtte_filters,
  select = select_spec(
    choices = c("AVAL", "BMRKR1", "AGE"),
    selected = c("AVAL", "BMRKR1"),
    multiple = TRUE,
    fixed = FALSE,
    label = "Column"
  )
)
data_extract_srv

data_extract_spec(
  dataname = "ADSL",
  filter = NULL,
  select = select_spec(
    choices = c("AGE", "SEX", "USUBJID"),
    selected = c("SEX"),
    multiple = FALSE,
    fixed = FALSE
  )
)
data_extract_spec(
  dataname = "ADSL",
  filter = filter_spec(
    vars = variable_choices("ADSL", subset = c("AGE"))
  )
)
dynamic_filter <- filter_spec(
  vars = choices_selected(variable_choices("ADSL"), "COUNTRY"),
  multiple = TRUE
)
data_extract_spec(
  dataname = "ADSL",
  filter = dynamic_filter
)

data_extract_srv

**Extraction of the selector(s) details**

**Description**

[Stable]

Extracting details of the selection(s) in data_extract_ui elements.

**Usage**

data_extract_srv(id, datasets, data_extract_spec, ...)

## S3 method for class 'FilteredData'
data_extract_srv(id, datasets, data_extract_spec, ...)

## S3 method for class 'list'
data_extract_srv(
  id,
  datasets,
  data_extract_spec,
```r
join_keys = NULL,
select_validation_rule = NULL,
filter_validation_rule = NULL,
dataset_validation_rule = if (is.null(select_validation_rule) &&
    is.null(filter_validation_rule)) {
  NULL
} else {

  shinyvalidate::sv_required("Please select a dataset")
},
...
)

Arguments

id
An ID string that corresponds with the ID used to call the module’s UI function.

datasets
(filteredData or list of reactive or non-reactive data.frame) object containing data either in the form of filteredData or as a list of data.frame. When passing a list of non-reactive data.frame objects, they are converted to reactive data.frames internally. When passing a list of reactive or non-reactive data.frame objects, the argument join_keys is required also.

data_extract_spec
(data_extract_spec or a list of data_extract_spec) A list of data filter and select information constructed by data_extract_spec.

join_keys
(join_keys or NULL) of keys per dataset in datasets.

select_validation_rule
(NULL or function) Should there be any shinyvalidate input validation of the select parts of the data_extract_ui.

You can use a validation function directly (i.e. select_validation_rule = shinyvalidate::sv_required()) or for more fine-grained control use a function:

select_validation_rule = ~ if (length(.) > 2) "Error".

If NULL then no validation will be added. See example for more details.

filter_validation_rule
(NULL or function) Same as select_validation_rule but for the filter (values) part of the data_extract_ui.

dataset_validation_rule
(NULL or function) Same as select_validation_rule but for the choose dataset part of the data_extract_ui.

Value

A reactive list containing following fields:

• filters: A list with the information on the filters that are applied to the data set.
```
**data_extract_srv**

- **select**: The variables that are selected from the dataset.
- **always_selected**: The column names from the data set that should always be selected.
- **reshape**: Whether reshape long to wide should be applied or not.
- **dataname**: The name of the data set.
- **internal_id**: The id of the corresponding shiny input element.
- **keys**: The names of the columns that can be used to merge the data set.
- **iv**: A shinyvalidate::InputValidator containing validator for this data_extract.

**References**

**Examples**

```r
library(shiny)
library(shinyvalidate)
library(teal.data)
library(teal.widgets)

# Sample ADSL dataset
ADSL <- data.frame(
  STUDYID = "A",
  USUBJID = LETTERS[1:10],
  SEX = rep(c("F", "M"), 5),
  AGE = rpois(10, 30),
  BMRKR1 = rlnorm(10)
)

# Specification for data extraction
adsl_extract <- data_extract_spec(
  dataname = "ADSL",
  filter = filter_spec(vars = "SEX", choices = c("F", "M"), selected = "F"),
  select = select_spec(
    label = "Select variable:",
    choices = variable_choices(ADSL, c("AGE", "BMRKR1")),
    selected = "AGE",
    multiple = TRUE,
    fixed = FALSE
  )
)

# Using reactive list of data.frames
data_list <- list(ADSL = reactive(ADSL))

join_keys <- join_keys(join_key("ADSL", "ADSL", c("STUDYID", "USUBJID")))

# App: data extraction with validation
ui <- fluidPage(
  standard_layout(
    output = verbatimTextOutput("out1"),
  )
)
```
encoding = tagList(
  data_extract_ui(
    id = "adsl_var",
    label = "ADSL selection",
    data_extract_spec = adsl_extract
  )
)

server <- function(input, output, session) {
  adsl_reactive_input <- data_extract_srv(
    id = "adsl_var",
    datasets = data_list,
    data_extract_spec = adsl_extract,
    join_keys = join_keys,
    select_validation_rule = sv_required("Please select a variable.")
  )

  iv_r <- reactive({
    iv <- InputValidator$new()
    iv$add_validator(adsl_reactive_input()$iv)
    iv$enable()
    iv
  })

  output$out1 <- renderPrint({
    if (iv_r()$is_valid()) {
      cat(format_data_extract(adsl_reactive_input()))
    } else {
      "Please fix errors in your selection"
    }
  })

  if (interactive()) {
    shinyApp(ui, server)
  }
}

# App: simplified data extraction
ui <- fluidPage(
  standard_layout(
    output = verbatimTextOutput("out1"),
    encoding = tagList(
      data_extract_ui(
        id = "adsl_var",
        label = "ADSL selection",
        data_extract_spec = adsl_extract
      )
    )
  )
)

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

```r
adsl_reactive_input <- data_extract_srv(
  id = "adsl_var",
  datasets = data_list,
  data_extract_spec = adsl_extract
)

output$out1 <- renderPrint(adsl_reactive_input())
}

if (interactive()) {
  shinyApp(ui, server)
}
```

data_extract_ui

**data extraction module user-interface**

**Description**

[Experimental]

**Usage**

```r
data_extract_ui(id, label, data_extract_spec, is_single_dataset = FALSE)
```

**Arguments**

- **id** (character) shiny input unique identifier.
- **label** (character) Label above the data extract input.
- **data_extract_spec** (list of data_extract_spec) This is the outcome of listing data_extract_spec() constructor calls.
- **is_single_dataset** (logical) FALSE to display the dataset widget.

**Details**

There are three inputs that will be rendered:

1. Dataset select Optional. If more than one data_extract_spec is handed over to the function, a shiny shiny::selectInput will be rendered. Else just the name of the dataset is given.
2. Filter Panel Optional. If the data_extract_spec contains a filter element a shiny shiny::selectInput will be rendered with the options to filter the dataset.
3. Select panel A shiny shiny::selectInput to select columns from the dataset to go into the analysis.
The output can be analyzed using `data_extract_srv(...)`. This functionality should be used in the encoding panel of your teal app. It will allow app-developers to specify a `data_extract_spec()` object. This object should be used to teal module variables being filtered data from CDISC datasets.

You can use this function in the same way as any `shiny::selectInputs` UI. The corresponding server module can be found in `data_extract_srv()`. 

**Value**

Shiny `shiny::selectInputs` that allow to define how to extract data from a specific dataset. The input elements will be returned inside a `shiny::div` container.

**Examples**

```r
library(shiny)
library(teal.widgets)

adtte_filters <- filter_spec(
  vars = c("PARAMCD", "CNSR"),
  sep = ",",
  choices = c("OS-1" = "OS-1", "OS-0" = "OS-0", "PFS-1" = "PFS-1"),
  selected = "OS-1",
  multiple = FALSE,
  label = "Choose endpoint and Censor"
)

response_spec <- data_extract_spec(
  dataname = "ADTTE",
  filter = adtte_filters,
  select = select_spec(
    choices = c("AVAL", "BMRKR1", "AGE"),
    selected = c("AVAL", "BMRKR1"),
    multiple = TRUE,
    fixed = FALSE,
    label = "Column"
  )
)

# Call to use inside your teal module UI function
standard_layout(
  output = tableOutput("table"),
  encoding = div(
    data_extract_ui(
      id = "regressor",
      label = "Regressor Variable",
      data_extract_spec = response_spec
    )
  )
)
```

filter_spec

Data extract filter specification

Description

[Stable]
It consists in choices and additionally the variable names for the choices.

Usage

filter_spec(
  vars,
  choices = NULL,
  selected = if (inherits(choices, "delayed_data")) NULL else choices[1],
  multiple = length(selected) > 1 || inherits(selected, "all_choices"),
  label = "Filter by",
  sep = attr(choices, "sep"),
  drop_keys = FALSE
)

Arguments

vars (character or delayed_data) object. Character vector giving the columns to be filtered. These should be key variables of the data set to be filtered. delayed_data objects can be created via variable_choices(), value_choices(), or choices_selected().

choices (character or numeric or logical or (delayed_data) object. Named character vector to define the choices of a shiny shiny::selectInput(). These choices will be used to filter the dataset. These shall be filter values of the vars input separated by the separator(sep). Please watch out that the filter values have to follow the order of the vars input.
In the following example we will show how to filter two columns:
vars = c("PARAMCD","AVISIT") and choices = c("CRP - BASELINE", "ALT - BASELINE") will lead to a filtering of (PARAMCD == "CRP" & AVISIT == "BASELINE") | (PARAMCD == "ALT" & AVISIT == "BASELINE").
The sep input has to be " - " in this case.
delayed_data objects can be created via variable_choices() or value_choices().

selected (character or numeric or logical or (delayed_data or all_choices) object. Named character vector to define the selected values of a shiny shiny::selectInput() (default values). This value will be displayed inside the shiny app upon start. The all_choices object indicates selecting all possible choices.

multiple (logical) Whether multiple values shall be allowed in the shiny shiny::selectInput().

label (optional character). Define a label on top of this specific shiny shiny::selectInput(). The default value is "Filter by".
sep (character) A separator string to split the choices or selected inputs into the values of the different columns.

drop_keys (optional logical) whether to drop filter column from the dataset keys, TRUE on default.

Details

The `filter_spec` is used inside teal apps to allow filtering datasets for their key variables. Imagine having an adverse events table. It has the columns `PARAMCD` and `CNSR`. `PARAMCD` contains the levels "OS", "PFS", "EFS". `CNSR` contains the levels "0" and "1". The first example should show how a `filter_spec` setup will influence the drop-down menu the app user will see.

Value

`filter_spec`-S3-class object or `delayed_filter_spec`-S3-class object.

Examples

```r
# for Adverse Events table
filter_spec(
  vars = c("PARAMCD", "CNSR"),
  sep = "-",
  choices = c("OS-1" = "OS-1", "OS-0" = "OS-0", "PFS-1" = "PFS-1"),
  selected = "OS-1",
  multiple = FALSE,
  label = "Choose endpoint and Censor"
)

# filtering a single variable
filter_spec(
  vars = c("PARAMCD"),
  sep = "-",
  choices = c("OS", "PFS", "EFS"),
  selected = "OS",
  multiple = FALSE,
  label = "Choose endpoint"
)

# filtering a single variable by multiple levels of the variable
filter_spec(
  vars = c("PARAMCD"),
  sep = "-",
  choices = c("OS", "PFS", "EFS"),
  selected = c("OS", "PFS"),
  multiple = TRUE,
  label = "Choose endpoint"
)

# delayed version
filter_spec(
  vars = variable_choices("ADSL", "SEX"),
  sep = "-",
)```
choices = value_choices("ADSL", "SEX", "SEX"),
selected = "F",
multiple = FALSE,
label = "Choose endpoint and Censor"
)
# using `choices_selected()`
filter_spec(
  vars = choices_selected(variable_choices("ADSL", subset = c("SEX", "AGE")), "SEX", fixed = FALSE),
multiple = TRUE
)
filter_spec(
  vars = choices_selected(variable_choices("ADSL"), "SEX", fixed = TRUE),
multiple = TRUE
)
# choose all choices
adsl_filter <- filter_spec(
  vars = choices_selected(variable_choices("ADSL"), "SEX", fixed = FALSE),
  choices = value_choices("ADSL", "SEX"),
  selected = all_choices()
)

format_data_extract  Formatting data extracts

Description

Returns a human-readable string representation of an extracted data_extract_spec object.

Usage

format_data_extract(data_extract)

Arguments

data_extract  list the list output of data_extract_srv.

Details

This function formats the output of data_extract_srv. See the example for more information.

Value

character(1) representation of the data_extract object.
get_anl_relabel_call  

Gets the relabel call

Description  

[Stable]

Usage

get_anl_relabel_call(columns_source, datasets, anl_name = "ANL")

Arguments

columns_source  (named list) where names are column names, values are labels + additional attribute `dataname`  

datasets  (named list of reactive or non-reactive data.frame) object containing data as a list of data.frame. When passing a list of non-reactive data.frame objects, they are converted to reactive data.frame objects internally.
get_dataset_prefixed_col_names

anl_name (character(1)) Name of the analysis dataset.

Value
(calls) to relabel dataset and assign to anl_name.

---

get_dataset_prefixed_col_names

Returns non-key column names from data

Description

[Stable]

Usage

get_dataset_prefixed_col_names(data)

Arguments

data (data.frame) Data with attribute filter_and_columns. This can only be created by data_extract_srv(), which returns a shiny shiny::reactive().

Value

A named character vector with the non-key columns of the data.

References

data_extract_srv()

---

get_extract_datanames

Gets names of the datasets from a list of data_extract_spec objects

Description

[Stable]

Fetches dataname slot per data_extract_spec from a list of data_extract_spec.

Usage

get_extract_datanames(data_extracts)

Arguments

data_extracts (data_extract_spec(1)) object or a list (of lists) of data_extract_spec.
get_merge_call

Description

[Stable]

Creates list of calls depending on selector(s) and type of the merge. The merge order is the same as in selectors passed to the function.

Usage

```
get_merge_call(
  selector_list,
  join_keys = teal.data::join_keys(),
  dplyr_call_data = get_dplyr_call_data(selector_list, join_keys = join_keys),
  merge_function = "dplyr::full_join",
  anl_name = "ANL"
)
```

Arguments

- **selector_list** (reactive) output from `data_extract_multiple_srv()` or a reactive named list of outputs from `data_extract_srv()`. When using a reactive named list, the names must be identical to the shiny ids of the respective `data_extract_ui()`.
- **join_keys** (join_keys) nested list of keys used for joining.
- **dplyr_call_data** (list) simplified selectors with aggregated set of filters.
- **merge_function** (character(1) or reactive) A character string of a function that accepts the arguments x, y and by to perform the merging of datasets.
- **anl_name** (character(1)) Name of the analysis dataset.

Value

List with merge call elements.
**get_relabel_call**

Create relabel call from named character

**Description**

[Stable]

Function creates relabel call from named character.

**Usage**

```
get_relabel_call(labels)
```

**Arguments**

- `labels` (named character) where name is name is function argument name and value is a function argument value.

**Value**

call object with relabel step.

**Examples**

```
get_relabel_call(
  labels = c(
    x = as.name("ANL"),
    AGE = "Age",
    AVAL = "Continuous variable"
  )
)
```

```
get_relabel_call(
  labels = c(
    AGE = "Age",
    AVAL = "Continuous variable"
  )
)
```

---

**is_single_dataset**

Verify uniform dataset source across data extract specification

**Description**

[Stable]

Checks if the input data_extract_spec objects all come from the same dataset.
Usage

is_single_dataset(...)

Arguments

... either data_extract_spec objects or lists of data_extract_spec objects that do not contain NULL.

Value

TRUE if all data_extract_spec objects come from the same dataset, FALSE otherwise.

list_extract_spec Make sure that the extract specification is in list format

Description

[Stable]

Usage

list_extract_spec(x, allow_null = FALSE)

Arguments

x (data_extract_spec or list) of data_extract_spec elements.
allow_null (logical) whether x can be NULL.

Value

x as a list if it is not already.

merge_datasets Merge the datasets on the keys

Description

[Experimental]

Combines/merges multiple datasets with specified keys attribute.
merge_datasets

Usage

merge_datasets(
    selector_list,
    datasets,
    join_keys,
    merge_function = "dplyr::full_join",
    anl_name = "ANL"
)

Arguments

selector_list (reactive) output from `data_extract_multiple_srv()` or a reactive named list of outputs from `data_extractsrv()`. When using a reactive named list, the names must be identical to the shiny ids of the respective `data_extract_ui()`.

datasets (named list of reactive or non-reactive data.frame) object containing data as a list of data.frame. When passing a list of non-reactive data.frame objects, they are converted to reactive data.frame objects internally.

join_keys (join_keys) of variables used as join keys for each of the datasets in datasets. This will be used to extract the keys of every dataset.

merge_function (character(1) or reactive) A character string of a function that accepts the arguments x, y and by to perform the merging of datasets.

anl_name (character(1)) Name of the analysis dataset.

Details

Internally this function uses calls to allow reproducibility.

This function is often used inside a teal module server function with the selectors being the output of `data_extract_srv` or `data_extract_multiple_srv`.

# inside teal module server function

response <- data_extract_srv(
    id = "response",
    data_extract_spec = response_spec,
    datasets = datasets
)

regressor <- data_extract_srv(
    id = "regressor",
    data_extract_spec = regressor_spec,
    datasets = datasets
)

merged_data <- merge_datasets(list(regressor(), response()))

Value

merged_dataset list containing:

- expr (list of call) code needed to replicate merged dataset;
• **columns_source** (list) of column names selected for particular selector; Each list element contains named character vector where:
  - Values are the names of the columns in the ANL. In case if the same column name is selected in more than one selector it gets prefixed by the id of the selector. For example if two data_extract have id x, y, then their duplicated selected variable (for example AGE) is prefixed to be x.AGE and y.AGE;
  - Names of the vector denote names of the variables in the input dataset;
  - attr("dataname") to indicate which dataset variable is merged from;
  - attr("always selected") to denote the names of the variables which need to be always selected;
• **keys** (list) the keys of the merged dataset;
• **filter_info** (list) The information given by the user. This information defines the filters that are applied on the data. Additionally it defines the variables that are selected from the data sets.

**Examples**

```r
library(shiny)
library(teal.data)

X <- data.frame(A = c(1, 1:3), B = 2:5, D = 1:4, E = letters[1:4], G = letters[6:9])
Y <- data.frame(A = c(1, 1, 2), B = 2:4, C = c(4, 4:5), E = letters[4:6], G = letters[1:3])
join_keys <- join_keys(join_key("X", "Y", c("A", "B")))
selector_list <- list(
  list(
    dataname = "X",
    filters = NULL,
    select = "E",
    keys = c("A", "B"),
    reshape = FALSE,
    internal_id = "x"
  ),
  list(
    dataname = "Y",
    filters = NULL,
    select = "G",
    keys = c("A", "C"),
    reshape = FALSE,
    internal_id = "y"
  )
)
data_list <- list(X = reactive(X), Y = reactive(Y))

merged_datasets <- isolate(
  merge_datasets(
    selector_list = selector_list,
    datasets = data_list,
    join_keys = join_keys
  )
)
```
**merge_expression_module**

Merge expression module

**Description**

[Experimental]

Convenient wrapper to combine `data_extract_multiple_srv()` and `merge_expression_srv()` when no additional processing is required. Compare the example below with that found in `merge_expression_srv()`.

**Usage**

```r
merge_expression_module(
  datasets,
  join_keys = NULL,
  data_extract,
  merge_function = "dplyr::full_join",
  anl_name = "ANL",
  id = "merge_id"
)
```

```
## S3 method for class 'reactive'
merge_expression_module(
  datasets,
  join_keys = NULL,
  data_extract,
  merge_function = "dplyr::full_join",
  anl_name = "ANL",
  id = "merge_id"
)
```

```
## S3 method for class 'list'
merge_expression_module(
  datasets,
  join_keys = NULL,
  data_extract,
  merge_function = "dplyr::full_join",
  anl_name = "ANL",
  id = "merge_id"
)
```
merge_expression_module

Arguments

datasets (named list of reactive or non-reactive data.frame) object containing data as a list of data.frame. When passing a list of non-reactive data.frame objects, they are converted to reactive data.frame objects internally.

join_keys (join_keys) of variables used as join keys for each of the datasets in datasets. This will be used to extract the keys of every dataset.

data_extract (named list of data_extract_spec).

merge_function (character(1)) A character string of a function that accepts the arguments x, y and by to perform the merging of datasets.

anl_name (character(1)) Name of the analysis dataset.

id An ID string that corresponds with the ID used to call the module’s UI function.

Value

Reactive expression with output from merge_expression_srv().

See Also

merge_expression_srv()

Examples

library(shiny)
library(teal.data)
library(teal.widgets)

ADSL <- data.frame(
    STUDYID = "A",
    USUBJID = LETTERS[1:10],
    SEX = rep(c("F", "M"), 5),
    AGE = rpois(10, 30),
    BMRRK1 = rlnorm(10)
)

ADLB <- expand.grid(
    STUDYID = "A",
    USUBJID = LETTERS[1:10],
    PARAMCD = c("ALT", "CRP", "IGA"),
    AVISIT = c("SCREENING", "BASELINE", "WEEK 1 DAY 8", "WEEK 2 DAY 15")
)

ADLB$AVAL <- rlnorm(120)
ADLB$CHG <- rnorm(120)

data_list <- list(
    ADSL = reactive(ADSL),
    ADLB = reactive(ADLB)
)

join_keys <- join_keys(
    join_key("ADSL", "ADSL", c("STUDYID", "USUBJID")),
    join_key("ADLB", "ADLB", c("STUDYID", "USUBJID")),
    join_key("ADLBA", "ADLBA", c("STUDYID", "USUBJID")),
    join_key("ADLBB", "ADLBB", c("STUDYID", "USUBJID"))
)
merge_expression_module

```r
join_key("ADSL", "ADLB", c("STUDYID", "USUBJID")),
join_key("ADLB", "ADLB", c("STUDYID", "USUBJID", "PARAMCD", "AVISIT"))
)

adsl_extract <- data_extract_spec(
  dataname = "ADSL",
  select = select_spec(
    label = "Select variable:",
    choices = c("AGE", "BMRKR1"),
    selected = "AGE",
    multiple = TRUE,
    fixed = FALSE
  )
)

adlb_extract <- data_extract_spec(
  dataname = "ADLB",
  filter = filter_spec(vars = "PARAMCD", choices = c("ALT", "CRP", "IGA"), selected = "ALT"),
  select = select_spec(
    label = "Select variable:",
    choices = c("AVAL", "CHG"),
    selected = "AVAL",
    multiple = TRUE,
    fixed = FALSE
  )
)

ui <- fluidPage(
  standard_layout(
    output = div(
     verbatimTextOutput("expr"),
dataTableOutput("data")
    ),
    encoding = tagList(
      data_extract_ui("adsl_var", label = "ADSL selection", adsl_extract),
data_extract_ui("adlb_var", label = "ADLB selection", adlb_extract)
    )
  )
)

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

data_q <- eval_code(
  data_q,
  "ADSL <- data.frame(
    STUDYID = 'A',
    USUBJID = LETTERS[1:10],
    SEX = rep(c('F', 'M'), 5),
    AGE = rpois(10, 30),
    BMRKR1 = rlnorm(10)
  )"
)
}
```
```r
data_q <- eval_code(
  data_q,
  "ADLB <- expand.grid(
    STUDYID = 'A',
    USUBJID = LETTERS[1:10],
    PARAMCD = c('ALT', 'CRP', 'IGA'),
    AVISIT = c('SCREENING', 'BASELINE', 'WEEK 1 DAY 8', 'WEEK 2 DAY 15'),
    AVAL = rlnorm(120),
    CHG = rlnorm(120)
  )"
)

merged_data <- merge_expression_module(
  data_extract = list(adsl_var = adsl_extract, adlb_var = adlb_extract),
  datasets = data_list,
  join_keys = join_keys,
  merge_function = "dplyr::left_join"
)

code_merge <- reactive({
  for (exp in merged_data()$expr) data_q <- eval_code(data_q, exp)
  data_q
})

output$expr <- renderText(paste(merged_data()$expr, collapse = "\n"))
output$data <- renderDataTable(code_merge()[["ANL"]])

if (interactive()) {
  shinyApp(ui, server)
}
```

merge_expression_srv  

**Data merge module server**

**Description**

*[Experimental]*

**Usage**

```r
merge_expression_srv(
  id = "merge_id",
  selector_list,
  datasets,
  join_keys,
  merge_function = "dplyr::full_join",
  anl_name = "ANL"
)
```
merge_expression_srv

## S3 method for class 'reactive'
merge_expression_srv(
  id = "merge_id",
  selector_list,
  datasets,
  join_keys,
  merge_function = "dplyr::full_join",
  anl_name = "ANL"
)

## S3 method for class 'list'
merge_expression_srv(
  id = "merge_id",
  selector_list,
  datasets,
  join_keys,
  merge_function = "dplyr::full_join",
  anl_name = "ANL"
)

Arguments

- **id**
  An ID string that corresponds with the ID used to call the module's UI function.

- **selector_list**
  (reactive) output from data_extract_multiple_srv() or a reactive named list of outputs from data_extract_srv(). When using a reactive named list, the names must be identical to the shiny ids of the respective data_extract_ui().

- **datasets**
  (named list of reactive or non-reactive data.frame) object containing data as a list of data.frame. When passing a list of non-reactive data.frame objects, they are converted to reactive data.frame objects internally.

- **join_keys**
  (join_keys) of variables used as join keys for each of the datasets in datasets. This will be used to extract the keys of every dataset.

- **merge_function**
  (character(1) or reactive) A character string of a function that accepts the arguments x, y and by to perform the merging of datasets.

- **anl_name**
  (character(1)) Name of the analysis dataset.

Details

When additional processing of the data_extract_list input is required, merge_expression_srv() can be combined with data_extract_multiple_srv() or data_extract_srv() to influence the selector_list input. Compare the example below with that found in merge_expression_module().

Value

Reactive expression with output from merge_expression_srv().

See Also

merge_expression_module()
Examples

```r
library(shiny)
library(teal.data)
library(teal.widgets)

ADSL <- data.frame(
    STUDYID = "A",
    USUBJID = LETTERS[1:10],
    SEX = rep(c("F", "M"), 5),
    AGE = rpois(10, 30),
    BMRKR1 = rlnorm(10)
)

ADLB <- expand.grid(
    STUDYID = "A",
    USUBJID = LETTERS[1:10],
    PARAMCD = c("ALT", "CRP", "IGA"),
    AVISIT = c("SCREENING", "BASELINE", "WEEK 1 DAY 8", "WEEK 2 DAY 15")
)

ADLB$AVAL <- rlnorm(120)
ADLB$CHG <- rlnorm(120)

data_list <- list(
    ADSL = reactive(ADSL),
    ADLB = reactive(ADLB)
)

join_keys <- join_keys(
    join_key("ADSL", "ADSL", c("STUDYID", "USUBJID")),
    join_key("ADSL", "ADLB", c("STUDYID", "USUBJID")),
    join_key("ADLB", "ADLB", c("STUDYID", "USUBJID", "PARAMCD", "AVISIT"))
)

adsl_extract <- data_extract_spec(
    dataname = "ADSL",
    select = select_spec(
        label = "Select variable:",
        choices = c("AGE", "BMRKR1"),
        selected = "AGE",
        multiple = TRUE,
        fixed = FALSE
    )
)

adlb_extract <- data_extract_spec(
    dataname = "ADLB",
    filter = filter_spec(vars = "PARAMCD", choices = c("ALT", "CRP", "IGA"), selected = "ALT"),
    select = select_spec(
        label = "Select variable:",
        choices = c("AVAL", "CHG"),
        selected = "AVAL",
        multiple = TRUE,
        fixed = FALSE
    )
)
```
ui <- fluidPage(
  standard_layout(
    output = div(
      verbatimTextOutput("expr"),
      dataTableOutput("data")
    ),
    encoding = tagList(
      data_extract_ui("adsl_var", label = "ADSL selection", adsl_extract),
      data_extract_ui("adlb_var", label = "ADLB selection", adlb_extract)
    )
  )
)

server <- function(input, output, session) {
  data_q <- qenv()
  data_q <- eval_code(
    data_q,
    "ADSL <- data.frame(
      STUDYID = 'A',
      USUBJID = LETTERS[1:10],
      SEX = rep(c('F', 'M'), 5),
      AGE = rpois(10, 30),
      BMRKR1 = rlnorm(10)
    )"
  )
  data_q <- eval_code(
    data_q,
    "ADLB <- expand.grid(
      STUDYID = 'A',
      USUBJID = LETTERS[1:10],
      PARAMCD = c('ALT', 'CRP', 'IGA'),
      AVISIT = c('SCREENING', 'BASELINE', 'WEEK 1 DAY 8', 'WEEK 2 DAY 15'),
      AVAL = rlnorm(120),
      CHG = rlnorm(120)
    )"
  )
  selector_list <- data_extract_multiple_srv(
    list(adsl_var = adsl_extract, adlb_var = adlb_extract),
    datasets = data_list
  )
  merged_data <- merge_expression_srv(
    selector_list = selector_list,
    datasets = data_list,
    join_keys = join_keys,
    merge_function = "dplyr::left_join"
  )
}
code_merge <- reactive(
  for (exp in merged_data()$expr) data_q <- eval_code(data_q, exp)
  data_q
)

output$expr <- renderText(paste(merged_data()$expr, collapse = "\n"))
output$data <- renderDataTable(code_merge()["ANL"])

if (interactive()) {
  shinyApp(ui, server)
}

no_selected_as_NULL

Check select choices for no choice made

Description

[Stable]

Usage

no_selected_as_NULL(x)

Arguments

x (character) Word that shall be checked for NULL, empty, "--no-selection".

Value

The word or NULL.

resolve_delayed

Resolve delayed inputs by evaluating the code within the provided datasets

Description

[Stable]
Usage

```r
resolve_delayed(x, datasets, keys)
```

### S3 method for class 'FilteredData'
```r
resolve_delayed(
  x,
  datasets,
  keys = sapply(datasets$datanames(), datasets$get_keys, simplify = FALSE)
)
```

### S3 method for class 'list'
```r
resolve_delayed(x, datasets, keys = NULL)
```

Arguments

- **x** (delayed_data, list) to resolve.
- **datasets** (FilteredData or named list) to use as a reference to resolve `x`.
- **keys** (named list) with primary keys for each dataset from `datasets`. `names(keys)` should match `names(datasets)`.

Value

Resolved object.

Methods (by class)

- `resolve_delayed(FilteredData)`: Default values for `keys` parameters is extracted from `datasets`.
- `resolve_delayed(list)`: Generic method when `datasets` argument is a named list.

Examples

```r
library(shiny)
ADSL <- teal.transform::rADSL
isolate({
  data_list <- list(ADSL = reactive(ADSL))

  # value_choices example
  v1 <- value_choices("ADSL", "SEX", "SEX")
  v1
  resolve_delayed(v1, data_list)

  # variable_choices example
  v2 <- variable_choices("ADSL", c("BMRKR1", "BMRKR2"))
  v2
  resolve_delayed(v2, data_list)

  # data_extract_spec example
  adsl_filter <- filter_spec(
```
select_spec

Column selection input specification

Description

[Stable]

select_spec is used inside teal to create a shiny::selectInput() that will select columns from a dataset.

Usage

select_spec(
  choices,
```r
selected = if (inherits(choices, "delayed_data")) NULL else choices[1],
multiple = length(selected) > 1 || inherits(selected, "all_choices"),
fixed = FALSE,
always_selected = NULL,
ordered = FALSE,
label = "Select"
)

selectspec.delayed_data(
  choices,
  selected = NULL,
  multiple = length(selected) > 1,
  fixed = FALSE,
  always_selected = NULL,
  ordered = FALSE,
  label = NULL
)

selectspec.default(
  choices,
  selected = choices[1],
  multiple = length(selected) > 1,
  fixed = FALSE,
  always_selected = NULL,
  ordered = FALSE,
  label = NULL
)
```

**Arguments**

- **choices** (character or delayed_data) object. Named character vector to define the choices of a shiny `shiny::selectInput()` These have to be columns in the dataset defined in the `data_extract_spec()` where this is called. delayed_data objects can be created via `variable_choices()` or `value_choices()`.

- **selected** (optional character or NULL or all_choices or delayed_data). Named character vector to define the selected values of a shiny `shiny::selectInput()`. Passing an all_choices() object indicates selecting all possible choices. Defaults to the first value of choices or NULL for delayed data loading.

- **multiple** (logical) Whether multiple values shall be allowed in the shiny `shiny::selectInput()`.

- **fixed** (optional logical). `data_extract_spec()` specific feature to hide the choices selected in case they are not needed. Setting fixed to TRUE will not allow the user to select columns. It will then lead to a selection of columns in the dataset that is defined by the developer of the app.

- **always_selected** (character) Additional column names from the data set that should always be selected.

- **ordered** (logical(1)) Flags whether selection order should be tracked.
select_spec

label (optional character). Define a label on top of this specific shiny `shiny::selectInput()`. The default value is "Select".

Value

A select_spec-S3 class object or delayed_select_spec-S3-class object. It contains all input values.

If select_spec, then the function double checks the choices and selected inputs.

Examples

```r
# Selection with just one column allowed
select_spec(
  choices = c("AVAL", "BMRKR1", "AGE"),
  selected = c("AVAL"),
  multiple = FALSE,
  fixed = FALSE,
  label = "Column"
)

# Selection with just multiple columns allowed
select_spec(
  choices = c("AVAL", "BMRKR1", "AGE"),
  selected = c("AVAL", "BMRKR1"),
  multiple = TRUE,
  fixed = FALSE,
  label = "Columns"
)

# Selection without user access
select_spec(
  choices = c("AVAL", "BMRKR1"),
  selected = c("AVAL", "BMRKR1"),
  multiple = TRUE,
  fixed = TRUE,
  label = "Columns"
)

# Delayed version
select_spec(
  label = "Select variable:",
  choices = variable_choices("ADSL", c("BMRKR1", "BMRKR2")),
  selected = "BMRKR1",
  multiple = FALSE,
  fixed = FALSE
)

# all_choices passed to selected
select_spec(
  label = "Select variable:",
  choices = variable_choices("ADSL", c("BMRKR1", "BMRKR2")),
  selected = all_choices()
)
split_by_sep

)  

# Both below objects are semantically the same  
select_spec(choices = variable_choices("ADSL"), selected = variable_choices("ADSL"))  
select_spec(choices = variable_choices("ADSL"), selected = all_choices())

---

**split_by_sep**  
**Split by separator (matched exactly)**

**Description**  
[Stable]

**Usage**  
split_by_sep(x, sep)

**Arguments**

x  
(character) Character vector, each element of which is to be split. Other inputs, including a factor return themselves.

sep  
(character) separator to use for splitting.

**Value**

List of character vectors split by sep. Self if x is not a character.

---

**value_choices**  
**Value labeling and filtering based on variable relationship**

**Description**  
[Stable]

Wrapper on choices_labeled to label variable values basing on other variable values.

**Usage**

value_choices(data, var_choices, var_label = NULL, subset = NULL, sep = " - ")

## S3 method for class 'character'
value_choices(data, var_choices, var_label = NULL, subset = NULL, sep = " - ")

## S3 method for class 'data.frame'
value_choices(data, var_choices, var_label = NULL, subset = NULL, sep = " - ")
variable_choices

Variable label extraction and custom selection from data

Description

[Stable]

Wrapper on choices_labeled to label variables basing on existing labels in data.

Arguments

data (data.frame, character) If data.frame, then data to extract labels from. If character, then name of the dataset to extract data from once available.

var_choices (character or NULL) vector with choices column names.

var_label (character) vector with labels column names.

subset (character or function) If character, vector with values to subset. If function, then this function is used to determine the possible columns (e.g. all factor columns). In this case, the function must take only single argument "data" and return a character vector.
See examples for more details.

sep (character) separator used in case of multiple column names.

Value

named character vector or delayed_data object.

Examples

ADR$ <- teal.transform::rADRS
value_choices(ADR$, "PARAMCD", "PARAM", subset = c("BESRspi", "INVEt"))
value_choices(ADR$, c("PARAMCD", "ARMCD"), c("PARAM", "ARM"))
value_choices(ADR$, c("PARAMCD", "ARMCD"), c("PARAM", "ARM"),
  subset = c("BESRspi - ARM A", "INVEt - ARM A", "OVRINd - ARM A")
)
value_choices(ADR$, c("PARAMCD", "ARMCD"), c("PARAM", "ARM"), sep = " --- ")

# delayed version
value_choices("ADR$, c("PARAMCD", "ARMCD"), c("PARAM", "ARM"))

# functional subset
value_choices(ADR$, "PARAMCD", "PARAM", subset = function(data) {
  levels(data$PARAMCD)[1:2]
})
variable_choices

Usage

variable_choices(data, subset = NULL, fill = FALSE, key = NULL)

# S3 method for class 'character'
variable_choices(data, subset = NULL, fill = FALSE, key = NULL)

# S3 method for class 'data.frame'
variable_choices(data, subset = NULL, fill = TRUE, key = NULL)

Arguments

data (data.frame or character) If data.frame, then data to extract labels from. If character, then name of the dataset to extract data from once available.

subset (character or function) If character, then a vector of column names. If function, then this function is used to determine the possible columns (e.g. all factor columns). In this case, the function must take only single argument "data" and return a character vector.
See examples for more details.

fill (logical(1)) if TRUE, the function will return variable names for columns with non-existent labels; otherwise will return NA for them.

key (character) vector with names of the variables, which are part of the primary key of the data argument.
This is an optional argument, which allows to identify variables associated with the primary key and display the appropriate icon for them in the `teal.widgets::optionalSelectInput()` widget.

Value

Named character vector with additional attributes or delayed_data object.

Examples

library(teal.data)

ADRS <- teal.transform::rADRS
variable_choices(ADRS)
variable_choices(ADRS, subset = c("PARAM", "PARAMCD"))
variable_choices(ADRS, subset = c("", "PARAM", "PARAMCD"))
variable_choices(
  ADRS,
  subset = c("", "PARAM", "PARAMCD"),
  key = default_cdisc_join_keys["ADRS", "ADRS"]
)

# delayed version
variable_choices("ADRS", subset = c("SUBJID", "STUDYID"))

# functional subset (with delayed data) - return only factor variables
variable_choices("ADRS", subset = function(data) {
  # code here
})
idx <- vapply(data, is.factor, logical(1))
names(data)[idx]
add_no_selected_choices, 3
all_choices, 3
check_no_multiple_selection, 4
choices_labeled, 5, 45, 46
choices_selected, 5, 6
choices_selected(), 3, 23
compose_and_enable_validators, 9
data_extract_multiple_srv, 12
data_extract_multiple_srv(), 28, 31, 37
data_extract_spec, 15, 15, 18, 21
data_extract_spec(), 21, 22, 43
data_extract_srv, 17, 19, 25
data_extract_srv(), 13, 22, 27, 28, 31, 37
data_extract_ui, 17, 21
data_extract_ui(), 28, 31, 37
datanames_input, 11
filter_spec, 16, 23
format_data_extract, 25
get_anl_relabel_call, 26
get_dataset_prefixed_col_names, 27
get_extract_datanames, 27
get_merge_call, 28
get_relabel_call, 29
is.choices_selected(choices_selected), 6
is_single_dataset, 29
list_extract_spec, 30
merge_datasets, 30
merge_expression_module, 33
merge_expression_module(), 37
merge_expression_srv, 36
merge_expression_srv(), 33, 34, 37
no_selected_as_NULL, 40
print.choices_labeled
(choices_labeled), 5
resolve_delayed, 40
select_spec, 16, 42
shiny::div, 22
shiny::helpText(), 12
shiny::reactive(), 27
shiny::selectInput, 21, 22
shiny::selectInput(), 5, 23, 42–44
split_by_sep, 45
teal.widgets::optionalSelectInput(), 6, 47
value_choices, 45
value_choices(), 7, 23, 43
variable_choices, 46
variable_choices(), 7, 23, 43