Package 'teal.transform'

February 12, 2025

Title Functions for Extracting and Merging Data in the 'teal' Framework

Version 0.6.0

Date 2025-02-06

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 (>= 0.1.3), stats, teal.data (>= 0.7.0), teal.logger (>= 0.3.1), teal.widgets (>= 0.4.3), tidyr (>= 1.0.0), tidyselect, utils

Suggests knitr (>= 1.42), rmarkdown (>= 2.23), roxy.shinylive (>= 1.0.0), testthat (>= 3.1.5), withr (>= 2.0.0)

VignetteBuilder knitr, rmarkdown

RdMacros lifecycle

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

Config/Needs/website insightsengineering/nesttemplate

Encoding UTF-8

Language en-US

RoxygenNote 7.3.2

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 2025-02-12 06:00:02 UTC

Contents

---

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.

---

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

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

ADSL <- rADSL
ADTTE <- 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()`.

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, (delayed_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	(logical) optional, 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)

ADSL <- teal.data::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"),
    c("name for A", "nothing", "name for B", "name for C")
  ),
  selected = "A"
)

# Passing a `delayed_data` object to `selected`
choices_selected(
  choices = variable_choices("ADSL"),
  selected = variable_choices("ADSL", subset = c("STUDYID"))
```

```
)

# Passing `delayed_choices` object - semantically identical objects:
choices_selected(choices = letters, selected = letters)
choices_selected(choices = letters, selected = all_choices())

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

# functional form (subsetting for factor variables only) of choices_selected
# with delayed data loading
choices_selected(variable_choices("ADSL", subset = function(data) {
  idx <- vapply(data, is.factor, logical(1))
  names(data)[idx]
}))

cs <- choices_selected(
  choices = c("A", "B", "C"),
  selected = "A"
)

ui <- fluidPage(
  optionalSelectInput(
    inputId = "id",
    choices = cs$choices,
    selected = cs$selected
  )
)

server <- function(input, output, session) {}
if (interactive()) {
  shinyApp(ui, server)
}
```

---

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::InputValidators
returned into a single validator and enables this.

Usage

```
compose_and_enable_validators(iv, selector_list, validator_names = NULL)
```

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

```
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 color column",
          data_extract_spec = iris_extract
        )
      )
    )
  )

  server <- function(input, output, session) {
    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"),
          function(x) if (length(x) != 2) "Exactly 2 'Y' column variables must be chosen"
        )
      )
    )
    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)
  }
```

---

datanames_input   *Help text with available datasets input*

Description

[Stable]

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

Usage

```
datanames_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
```

```
 } else {

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

Arguments

data_extract   (named list of data_extract_spec objects) the list data_extract_spec ob-
                    jects. 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(.) > 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) {
  selector_list <- data_extract_multiple_srv(
    list(x_var = iris_select, species_var = iris_filter),
    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"),
        function(x) if (length(x) != 2) "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\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

```
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 se-
                  lected 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

`teal.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_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,
  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](#).
...	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 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.
- 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

[data_extract_srv](#)

Examples

```
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
      )
    )
```

```r
  )
)
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) {
  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	teal *data extraction module user-interface*

Description

[Experimental]

Usage

```
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 anal-
   ysis.

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 module` 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

```
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 = tags$div(
    data_extract_ui(
      id = "regressor",
      label = "Regressor Variable",
      data_extract_spec = response_spec
    )
  )
)
```

---

delayed_choices | *Bare constructor for* delayed_choices *object*

Description

[Experimental]

Special S3 structures that delay selection of possible choices in a filter_spec, select_spec or choices_selected object.

Usage

```
all_choices()

first_choice()

last_choice()

first_choices(n)

last_choices(n)
```

Arguments

n   positive (`integer`-like) number of first/last items to subset to

Value

Object of class `delayed_data`, `delayed_choices`, which is a function that returns the appropriate subset of its argument. `all_choices`, `first_choices`, and `last_choices` structures also have an additional class for internal use.

Examples

```
# These pairs of structures represent semantically identical specifications:
choices_selected(choices = letters, selected = letters)
choices_selected(choices = letters, selected = all_choices())

choices_selected(choices = letters, selected = letters[1])
choices_selected(choices = letters, selected = first_choice())

choices_selected(choices = letters, selected = letters[length(letters)])
choices_selected(choices = letters, selected = last_choice())

choices_selected(choices = letters, selected = head(letters, 4))
choices_selected(choices = letters, selected = first_choices(4))

choices_selected(choices = letters, selected = tail(letters, 4))
choices_selected(choices = letters, selected = last_choices(4))

filter_spec(
  vars = c("selected_variable"),
  choices = c("value1", "value2", "value3"),
  selected = "value3"
)
filter_spec(
  vars = c("selected_variable"),
  choices = c("value1", "value2", "value3"),
  selected = last_choice()
)
```

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, "multiple_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 delayed_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. delayed_choices objects resolve selection when choices become available.
multiple	(logical) Whether multiple values shall be allowed in the shiny shiny::selectInput().
label	(character) optional, defines 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	(logical) optional, 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

```
# 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.

Examples

```
library(shiny)

simple_des <- data_extract_spec(
  dataname = "iris",
  filter = filter_spec(vars = "Petal.Length", choices = c("1.4", "1.5")),
  select = select_spec(choices = c("Petal.Length", "Species"))
)

ui <- fluidPage(
  data_extract_ui(
    id = "extract",
    label = "data extract ui",
    data_extract_spec = simple_des,
    is_single_dataset = TRUE
  ),
  verbatimTextOutput("formatted_extract")
)
server <- function(input, output, session) {
  extracted_input <- data_extract_srv(
    id = "extract",
    datasets = list(iris = iris),
    data_extract_spec = simple_des
  )
  output$formatted_extract <- renderPrint({
    cat(format_data_extract(extracted_input()))
  })
}

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

---

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

Value

(`call`) 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.

Value

character vector with the unique dataname set.

---

get_merge_call   *Get merge call from a list of selectors*

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.

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_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

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 = "reponse",
  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

```
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
```

```
  )
)

paste(merged_datasets$expr)
```

---

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

```
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"
)
```

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),
  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 <- rnorm(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 = tags$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)
        )"
    )

    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

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

```
## 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

```
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 = tags$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

```
resolve_delayed(x, datasets, keys)

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

## S3 method for class 'list'
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

```
library(shiny)

ADSL <- teal.data::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(
  vars = variable_choices("ADSL", "SEX"),
  sep = "-",
  choices = value_choices("ADSL", "SEX", "SEX"),
  selected = "F",
  multiple = FALSE,
  label = "Choose endpoint and Censor"
)

adsl_select <- select_spec(
  label = "Select variable:",
  choices = variable_choices("ADSL", c("BMRKR1", "BMRKR2")),
  selected = "BMRKR1",
  multiple = FALSE,
  fixed = FALSE
)

adsl_de <- data_extract_spec(
  dataname = "ADSL",
  select = adsl_select,
  filter = adsl_filter
)

resolve_delayed(adsl_filter, datasets = data_list)
resolve_delayed(adsl_select, datasets = data_list)
resolve_delayed(adsl_de, datasets = data_list)

# nested list (arm_ref_comp)
arm_ref_comp <- list(
  ARMCD = list(
    ref = variable_choices("ADSL"),
    comp = variable_choices("ADSL")
  )
)

resolve_delayed(arm_ref_comp, datasets = data_list)
})
```

---

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

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

select_spec.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	(character or NULL or delayed_choices or delayed_data) optional named character vector to define the selected values of a shiny [shiny::selectInput()](). Passing a delayed_choices object defers selection until data is available. 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	(logical) optional [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.

label   (character) optional, defines 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

```
# 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
)
```

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

# 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 = " - ")
```

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`, `delayed_variable_choices`) Choice of 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

```
ADRS <- teal.data::rADRS
value_choices(ADRS, "PARAMCD", "PARAM", subset = c("BESRSPI", "INVET"))
value_choices(ADRS, c("PARAMCD", "ARMCD"), c("PARAM", "ARM"))
value_choices(ADRS, c("PARAMCD", "ARMCD"), c("PARAM", "ARM"),
  subset = c("BESRSPI - ARM A", "INVET - ARM A", "OVRINV - ARM A")
)
value_choices(ADRS, c("PARAMCD", "ARMCD"), c("PARAM", "ARM"), sep = " --- ")

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

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

`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.

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 <- 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("USUBJID", "STUDYID"))

# functional subset (with delayed data) - return only factor variables
variable_choices("ADRS", subset = function(data) {
  idx <- vapply(data, is.factor, logical(1))
```

```
  names(data)[idx]
})
```

Index