

# Package: teal.widgets (via r-universe)

June 30, 2026

**Title** 'shiny' Widgets for 'teal' Applications

**Version** 0.6.1

**Date** 2026-06-30

**Description** Collection of 'shiny' widgets to support 'teal' applications. Enables the manipulation of application layout and plot or table settings.

**License** Apache License 2.0

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

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

**Depends** R (>= 3.6)

**Imports** bslib (>= 0.8.0), checkmate (>= 2.1.0), ggplot2 (>= 3.4.3), graphics, grDevices, gt (>= 1.3.0), gtsummary (>= 2.5.0), htmltools (>= 0.5.4), lifecycle (>= 0.2.0), methods, rtables (>= 0.6.12), rvest (>= 1.0.3), shiny (>= 1.8.1), shinyjs (>= 2.1.0), shinyWidgets (>= 0.5.1), styler (>= 1.2.0), xml2 (>= 1.3.0)

**Suggests** DT, knitr (>= 1.42), lattice (>= 0.18-4), magrittr (>= 1.5), png, rmarkdown (>= 2.23), shinytest2 (>= 0.2.0), shinyvalidate, testthat (>= 3.2.3), withr (>= 2.1.0)

**VignetteBuilder** knitr, rmarkdown

**Config/Needs/verdepcheck** rstudio/bslib, mllg/checkmate, tidyverse/ggplot2, rstudio/htmltools, r-lib/lifecycle, insightsengineering/rtables, rstudio/shiny, daattali/shinyjs, dreamRs/shinyWidgets, r-lib/styler, rstudio/DT, yihui/knitr, deepayan/lattice, tidyverse/magrittr, cran/png, tidyverse/rvest, rstudio/rmarkdown, rstudio/shinytest2, rstudio/shinyvalidate, r-lib/testthat, r-lib/withr

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

**Config/roxygen2/version** 8.0.0

**Config/testthat/edition** 3

**Encoding** UTF-8

**Language** en-US

**NeedsCompilation** no

**Author** Dawid Kaledkowski [aut], Pawel Rucki [aut], Mahmoud Hallal [aut], Nikolas Burkoff [aut], Maciej Nasinski [aut], Konrad Pagacz [aut], Junlue Zhao [aut], Dony Unardi [aut, cre], F. Hoffmann-La Roche AG [cph, fnd]

**Maintainer** Dony Unardi <unardid@gene.com>

**Repository** <https://cran.r-universe.dev>

**Date/Publication** 2026-06-30 13:40:02 UTC

**RemoteUrl** <https://github.com/cran/teal.widgets>

**RemoteRef** HEAD

**RemoteSha** c967fdc2233789b38be561519f1a38a35e440863

## Contents

basic_table_args . . . . .	3
clean_brushedPoints . . . . .	3
draggable_buckets . . . . .	4
get_dt_rows . . . . .	6
ggplot2_args . . . . .	7
nested_closeable_modal . . . . .	8
optionalSelectInput . . . . .	9
optionalSliderInput . . . . .	13
optionalSliderInputValMinMax . . . . .	14
panel_group . . . . .	15
panel_item . . . . .	16
parse_basic_table_args . . . . .	17
parse_ggplot2_args . . . . .	18
plot_with_settings . . . . .	19
resolve_basic_table_args . . . . .	23
resolve_ggplot2_args . . . . .	24
standard_layout . . . . .	26
table_with_settings . . . . .	27
verbatim_popup . . . . .	28
white_small_well . . . . .	30

**Index**

**31**

---

basic_table_args	<i>Builds a basic_table_args object</i>
------------------	---

---

### Description

This function has to be used to build an input for a `basic_table_args` argument. The `basic_table_args` argument should be a part of every module which contains any `rtables` object. Arguments are validated to match their `rtables` equivalents.

For more details see the vignette: `vignette("custom-basic-table-arguments", package = "teal.widgets")`.

### Usage

```
basic_table_args(...)
```

### Arguments

... arguments compatible with `rtables::basic_table()`.

### Value

(`basic_table_args`) object.

### See Also

- `resolve_basic_table_args()` to resolve multiple objects into one using pre-defined priorities.
- `parse_basic_table_args()` to parse resolved list into list of calls.

### Examples

```
basic_table_args(subtitles = "SUBTITLE")
```

---

clean_brushedPoints	<i>Clean brushed points</i>
---------------------	-----------------------------

---

### Description

Cleans and organizes output to account for NAs and remove empty rows. Wrapper around `shiny::brushedPoints`.

### Usage

```
clean_brushedPoints(data, brush)
```

**Arguments**

`data` (data.frame)  
A data.frame from which to select rows.

`brush` (list)  
The data from a brush e.g. `input$plot_brush`.

**Value**

A data.frame of selected rows.

**Examples**

```
brush <- list(
  mapping = list(
    x = "AGE",
    y = "BMRKR1"
  ),
  xmin = 30, xmax = 40,
  ymin = 0.7, ymax = 10,
  direction = "xy"
)

data <- data.frame(
  STUDYID = letters[1:20],
  USUBJID = LETTERS[1:20],
  AGE = sample(25:40, size = 20, replace = TRUE),
  BMRKR1 = runif(20, min = 0, max = 12)
)
nrow(clean_brushedPoints(data, brush))
data$AGE[1:10] <- NA
nrow(clean_brushedPoints(data, brush))
```

---

draggable\_buckets      *Draggable Buckets*

---

**Description**

A custom widget with draggable elements that can be put into buckets.

**Usage**

```
draggable_buckets(input_id, label, elements = character(), buckets)
```

**Arguments**

input_id	(character(1)) the HTML id of this widget
label	(character(1) or shiny.tag) the header of this widget
elements	(character) the elements to drag into buckets
buckets	(character or list) the names of the buckets the elements can be put in or a list of key-pair values where key is a name of a bucket and value is a character vector of elements in a bucket

**Details**

shinyvalidate validation can be used with this widget. See example below.

**Value**

the HTML code comprising an instance of this widget

**Examples**

```
library(shiny)

ui <- bslib::page_fluid(
  draggable_buckets("id", "Choices #1", c("a", "b"), c("bucket1", "bucket2")),
  draggable_buckets("id2", "Choices #2", letters, c("vowels", "consonants")),
  verbatimTextOutput("out"),
  verbatimTextOutput("out2")
)
server <- function(input, output) {
  iv <- shinyvalidate::InputValidator$new()
  iv$add_rule(
    "id",
    function(data) if (length(data[["bucket1"]]) == 0) "There should be stuff in bucket 1"
  )
  iv$enable()

  observeEvent(list(input$id, input$id2), {
    print(isolate(input$id))
    print(isolate(input$id2))
  })
  output$out <- renderPrint({
    iv$is_valid()
    input$id
  })
  output$out2 <- renderPrint(input$id2)
}
if (interactive()) shinyApp(ui, server)

# With default elements in the bucket
ui <- bslib::page_fluid(
  draggable_buckets("id", "Choices #1", c("a", "b"), list(bucket1 = character(), bucket2 = c("c"))),
  verbatimTextOutput("out")
)
```

```

)
server <- function(input, output) {
  observeEvent(input$id, {
    print(isolate(input$id))
  })
  output$out <- renderPrint(input$id)
}
if (interactive()) shinyApp(ui, server)

```

---

get\_dt\_rows

*Map lengthMenu property*


---

### Description

Maps the lengthMenu selected value property of DT::datatable to a shiny variable.

### Usage

```
get_dt_rows(dt_name, dt_rows)
```

### Arguments

dt_name	ns() of inputId of the DT::datatable
dt_rows	ns() of inputId of the variable that holds the current selected value of lengthMenu

### Value

(shiny::tagList) A shiny tagList.

### Examples

```

library(shiny)
library(DT)

ui <- function(id) {
  ns <- NS(id)
  tagList(
    get_dt_rows(ns("data_table"), ns("dt_rows")),
    textOutput(ns("rows")),
    DT::DTOutput(ns("data_table"))
  )
}

# use the input$dt_rows in the Shiny Server function
server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$data_table <- DT::renderDataTable(iris)
    # Change rows selected to see the first line on the UI change
    rows <- reactive{{

```

```

      paste0("Selected Rows ", input$dt_rows)
    })
    output$rows <- renderText(rows())
  })
}
if (interactive()) {
  shinyApp(
    ui = ui("my_table_module"),
    server = function(input, output, session) server("my_table_module")
  )
}

```

---

ggplot2\_args

*Creates ggplot2\_args object*


---

### Description

Constructor of ggplot2\_args class of objects. The ggplot2\_args argument should be a part of every module which contains any ggplot2 graphics. The function arguments are validated to match their ggplot2 equivalents.

For more details see the vignette: `vignette("custom-ggplot2-arguments", package = "teal.widgets")`.

### Usage

```
ggplot2_args(labs = list(), theme = list())
```

### Arguments

labs	(named list) where all fields have to match <code>ggplot2::labs()</code> arguments.
theme	(named list) where all fields have to match <code>ggplot2::theme()</code> arguments.

### Value

(ggplot2\_args) object.

### See Also

- [resolve\\_ggplot2\\_args\(\)](#) to resolve multiple objects into one using pre-defined priorities.
- [parse\\_ggplot2\\_args\(\)](#) to parse resolved list into list of calls.

### Examples

```

ggplot2_args(
  labs = list(title = "TITLE"),
  theme = list(title = ggplot2::element_text(size = 20))
)

```

---

 nested\_closeable\_modal

*Nested Closeable Modal Popup*


---

## Description

**[Deprecated]** Alternative to `shiny::modalDialog`. Create a nested modal popup that can be shown/hidden using jQuery and modal id, without disturbing the parent modal.

## Usage

```
nested_closeable_modal(id, ..., modal_args = list(easyClose = TRUE))
```

## Arguments

<code>id</code>	(character(1)) shiny module id for the component. Note that this id can be used to show/hide this modal with the appended jQuery methods show/hide.
<code>...</code>	(shiny.tag) shiny UI elements that will be displayed in the modal UI
<code>modal_args</code>	(list) optional list of arguments for the <code>shiny::modalDialog</code> function to customize the modal. Has <code>easyClose</code> set to <code>TRUE</code> as default

## Value

(shiny.tag) returns HTML for shiny module UI which can be nested into a modal popup

## Examples

```
library(shiny)
library(shinyjs)

ui <- bslib::page_fluid(
  useShinyjs(),
  actionButton("show_1", "$(\"#modal_1\").modal(\"show\")"),
  nested_closeable_modal(
    "modal_1",
    modal_args = list(
      size = "1",
      title = "First Modal",
      easyClose = TRUE,
      footer = NULL
    ),
    tags$div(
      "This modal can be closed by running", tags$code("$(\"#modal_1\").modal(\"hide\")"),
      "in the JS console!",
      tags$br(),
      "Note that the second modal is placed right within this modal",
      tags$br()
    )
  )
)
```

```

"Alternatively, calling the", tags$code("removeModal()"),
"will remove all the active modal popups",
tags$br(), tags$br(),
actionButton("show_2", "${\"#modal_2\"}.modal(\"show\")"),
actionButton("hide_1", "${\"#modal_1\"}.modal(\"hide\")"),
nested_closeable_modal(
  id = "modal_2",
  modal_args = list(
    size = "m",
    title = "Second Modal",
    footer = NULL,
    easyClose = TRUE
  ),
  tags$div(
    "This modal can be closed by running", tags$code("${\"#modal_1\"}.modal(\"hide\")"),
    "in the JS console!",
    "Note that removing the parent will remove the child.",
    "But, reopening will remember the open state of child",
    actionButton("hide_2", "${\"#modal_2\"}.modal(\"hide\")"),
    actionButton("hide_all", "${\"#modal_1\"}.modal(\"hide\")")
  )
)
)
)
)
)

server <- function(input, output) {
  observeEvent(input$show_1, {
    runjs("${\"#modal_1\"}.modal(\"show\")")
  })
  observeEvent(input$show_2, {
    runjs("${\"#modal_2\"}.modal(\"show\")")
  })
  observeEvent(c(input$hide_1, input$hide_all), {
    runjs("${\"#modal_1\"}.modal(\"hide\")")
  })
  observeEvent(input$hide_2, {
    runjs("${\"#modal_2\"}.modal(\"hide\")")
  })
}

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

```

**Description**

Wrapper for `shinyWidgets::pickerInput()` with additional features. When `fixed = TRUE` or when the number of choices is less or equal to 1 (see `fixed_on_single`), the `pickerInput` widget is hidden and non-interactive widget will be displayed instead. Toggle of HTML elements is just the visual effect to avoid displaying `pickerInput` widget when there is only one choice.

**Usage**

```
optionalSelectInput(
  inputId,
  label = NULL,
  choices = NULL,
  selected = NULL,
  multiple = FALSE,
  sep = NULL,
  options = list(),
  label_help = NULL,
  fixed = FALSE,
  fixed_on_single = FALSE,
  width = NULL
)

updateOptionalSelectInput(
  session,
  inputId,
  label = NULL,
  selected = NULL,
  choices = NULL
)
```

**Arguments**

<code>inputId</code>	The input slot that will be used to access the value.
<code>label</code>	Display label for the control, or <code>NULL</code> for no label.
<code>choices</code>	List of values to select from. If elements of the list are named then that name rather than the value is displayed to the user.
<code>selected</code>	The initially selected value (or multiple values if <code>multiple = TRUE</code> ). If not specified then defaults to the first value for single-select lists and no values for multiple select lists.
<code>multiple</code>	Is selection of multiple items allowed?
<code>sep</code>	( <code>character(1)</code> ) A separator string to split the choices or selected inputs into the values of the different columns.
<code>options</code>	List of options, see <code>pickerOptions</code> for all available options. To limit the number of selection possible, see example below.
<code>label_help</code>	( <code>shiny.tag</code> ) optional, e.g. an object returned by <code>shiny::helpText()</code> .

fixed	(logical(1)) optional, whether to block user to select choices.
fixed_on_single	(logical(1)) optional, whether to block user to select a choice when there is only one or less choice. When FALSE, user is still able to select or deselect the choice.
width	(character(1)) The width of the input passed to pickerInput e.g. 'auto', 'fit', '100px' or '75%'
session	(shiny.session)

**Value**

(shiny.tag) HTML tag with pickerInput widget and non-interactive element listing selected values.

**Examples**

```
library(shiny)

ui_grid <- function(...) {
  bslib::page_fluid(
    bslib::layout_columns(
      col_widths = c(4, 4, 4),
      ...
    )
  )
}

ui <- ui_grid(
  wellPanel(
    optionalSelectInput(
      inputId = "c1",
      label = "Fixed choices",
      choices = LETTERS[1:5],
      selected = c("A", "B"),
      fixed = TRUE
    ),
    verbatimTextOutput(outputId = "c1_out")
  ),
  wellPanel(
    optionalSelectInput(
      inputId = "c2",
      label = "Single choice",
      choices = "A",
      selected = "A"
    ),
    verbatimTextOutput(outputId = "c2_out")
  ),
  wellPanel(
```

```

    optionalSelectInput(
      inputId = "c3",
      label = "NULL choices",
      choices = NULL
    ),
    verbatimTextOutput(outputId = "c3_out")
  ),
  wellPanel(
    optionalSelectInput(
      inputId = "c4",
      label = "Default",
      choices = LETTERS[1:5],
      selected = "A"
    ),
    verbatimTextOutput(outputId = "c4_out")
  ),
  wellPanel(
    optionalSelectInput(
      inputId = "c5",
      label = "Named vector",
      choices = c(`A` - value A` = "A", `B` - value B` = "B", `C` - value C` = "C"),
      selected = "A"
    ),
    verbatimTextOutput(outputId = "c5_out")
  ),
  wellPanel(
    selectInput(
      inputId = "c6_choices", label = "Update choices", choices = letters, multiple = TRUE
    ),
    optionalSelectInput(
      inputId = "c6",
      label = "Updated choices",
      choices = NULL,
      multiple = TRUE,
      fixed_on_single = TRUE
    ),
    verbatimTextOutput(outputId = "c6_out")
  )
)

server <- function(input, output, session) {
  observeEvent(input$c6_choices, ignoreNULL = FALSE, {
    updateOptionalSelectInput(
      session = session,
      inputId = "c6",
      choices = input$c6_choices,
      selected = input$c6_choices
    )
  })

  output$c1_out <- renderPrint(input$c1)
  output$c2_out <- renderPrint(input$c2)
  output$c3_out <- renderPrint(input$c3)
}

```

```
output$c4_out <- renderPrint(input$c4)
output$c5_out <- renderPrint(input$c5)
output$c6_out <- renderPrint(input$c6)
}

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

---

optionalSliderInput    *Optional Slider Input Widget*

---

### Description

Hidden input widgets are useful to have the `input[[inputId]]` variable available in the server function but no corresponding visual clutter from input widgets that provide only a single choice.

### Usage

```
optionalSliderInput(inputId, label, min, max, value, label_help = NULL, ...)
```

### Arguments

<code>inputId</code>	The input slot that will be used to access the value.
<code>label</code>	Display label for the control, or <code>NULL</code> for no label.
<code>min, max</code>	The minimum and maximum values (inclusive) that can be selected.
<code>value</code>	The initial value of the slider, either a number, a date (class <code>Date</code> ), or a date-time (class <code>POSIXt</code> ). A length one vector will create a regular slider; a length two vector will create a double-ended range slider. Must lie between <code>min</code> and <code>max</code> .
<code>label_help</code>	( <code>shiny.tag</code> ) optional object of class <code>shiny.tag</code> , e.g. an object returned by <code>shiny::helpText()</code>
<code>...</code>	optional arguments to <code>sliderInput</code>

### Details

if `min` or `max` are `NA` then the slider widget will be hidden

### Value

(`shiny.tag`) HTML tag with `sliderInput` widget.

**Examples**

```
ui <- bslib::page_fluid(
  shinyjs::useShinyjs(),
  optionalSliderInput("s", "shown", 0, 1, 0.2),
  optionalSliderInput("h", "hidden", 0, NA, 1),
)
if (interactive()) {
  shiny::shinyApp(ui, function(input, output) {})
}
```

---

optionalSliderInputValMinMax

*Optional Slider Input with minimal and maximal values*

---

**Description**

The `optionalSliderInput()` function needs three arguments to determine whether to hide the `sliderInput` widget or not. For teal modules we specify an optional slider input with one argument here called `value_min_max`.

**Usage**

```
optionalSliderInputValMinMax(
  inputId,
  label,
  value_min_max,
  label_help = NULL,
  ...
)
```

**Arguments**

<code>inputId</code>	The input slot that will be used to access the value.
<code>label</code>	Display label for the control, or <code>NULL</code> for no label.
<code>value_min_max</code>	( <code>numeric(1)</code> or <code>numeric(3)</code> ) If of length 1 then the value gets set to that number and the <code>sliderInput</code> will be hidden. Otherwise, if it is of length three the three elements will map to value, min and max of the <code>optionalSliderInput()</code> function.
<code>label_help</code>	( <code>shiny.tag</code> ) optional object of class <code>shiny.tag</code> , e.g. an object returned by <code>shiny::helpText()</code>
<code>...</code>	optional arguments to <code>sliderInput</code>

**Details**

For teal modules we parameterize an `optionalSliderInput` with one argument `value_min_max`

**Value**

(shiny.tag) HTML tag with range sliderInput widget.

**Examples**

```
ui <- bslib::page_fluid(
  shinyjs::useShinyjs(),
  optionalSliderInputValMinMax("a1", "b1", 1), # Hidden
  optionalSliderInputValMinMax("a2", "b2", c(3, 1, 5)) # Shown
)
if (interactive()) {
  shiny::shinyApp(ui, function(input, output) {})
}
```

---

panel\_group

*Panel group widget*

---

**Description****[Deprecated]**

Designed to group [panel\\_item](#) elements. Used to handle shiny inputs in the encoding panel.

**Usage**

```
panel_group(..., id = NULL)
```

**Arguments**

...	(shiny.tag) panels created by <a href="#">panel_group()</a>
id	optional, (character)

**Value**

(shiny.tag)

**Examples**

```
library(shiny)
panel_group(
  panel_item(
    title = "Display",
    collapsed = FALSE,
    checkboxGroupInput(
      "check",
      "Tables display",
      choices = LETTERS[1:3],
```

```

        selected = LETTERS[1]
      ),
      radioButtons(
        "radio",
        label = "Plot type",
        choices = letters[1:2],
        selected = letters[1]
      )
    ),
    panel_item(
      title = "Pre-processing",
      radioButtons(
        "filtering",
        "What to filter",
        choices = LETTERS[1:4],
        selected = LETTERS[1]
      ),
      radioButtons(
        "na_action",
        "NA action",
        choices = letters[1:3],
        selected = letters[1]
      )
    )
  )
)

```

---

panel\_item

*Panel item widget*

---

## Description

### [Deprecated]

Designed to be grouped using [panel\\_group](#) element. Used to handle shiny inputs in the encoding panel.

## Usage

```
panel_item(title, ..., collapsed = TRUE, input_id = NULL)
```

## Arguments

title	(character) title of panel
...	content of panel
collapsed	(logical) optional, whether to initially collapse panel

`input_id` (character) optional name of the panel item element. If supplied, this will register a shiny input variable that indicates whether the panel item is open or collapsed and is accessed with `input$input_id`.

**Value**

(shiny.tag)

**Examples**

```
library(shiny)
panel_item(
  title = "Display",
  collapsed = FALSE,
  checkboxGroupInput(
    "check",
    "Tables display",
    choices = LETTERS[1:3],
    selected = LETTERS[1]
  ),
  radioButtons(
    "radio",
    label = "Plot type",
    choices = letters[1:2],
    selected = letters[1]
  )
)
```

---

`parse_basic_table_args`

*Parses basic\_table\_args object into the basic\_table expression*

---

**Description**

A function to parse expression from the `basic_table_args` object.

**Usage**

```
parse_basic_table_args(basic_table_args = teal.widgets::basic_table_args())
```

**Arguments**

`basic_table_args`

(`basic_table_args`)

This argument could be a result of the [resolve\\_basic\\_table\\_args\(\)](#).

**Value**

(language) the `rtables::basic_table()` filled with additional arguments.

**Examples**

```
parse_basic_table_args(
  resolve_basic_table_args(
    user_table = basic_table_args(title = "TITLE"),
    user_default = basic_table_args(title = "DEFAULT_TITLE", subtitles = "SUBTITLE")
  )
)
```

---

`parse_ggplot2_args`      *Parse ggplot2\_args object into the ggplot2 expression*

---

**Description**

A function to parse expression from the `ggplot2_args` object.

**Usage**

```
parse_ggplot2_args(
  ggplot2_args = teal.widgets::ggplot2_args(),
  ggtheme = c("default", "gray", "bw", "linedraw", "light", "dark", "minimal", "classic",
             "void", "test")
)
```

**Arguments**

`ggplot2_args`      (`ggplot2_args`)  
 This argument could be a result of the [resolve\\_ggplot2\\_args\(\)](#).

`ggtheme`            (`character(1)`)  
 name of the `ggplot2` theme to be applied, e.g. "dark".

**Value**

(list) of up to three elements of class `language`: "labs", "ggtheme" and "theme".

**Examples**

```
parse_ggplot2_args(
  resolve_ggplot2_args(ggplot2_args(
    labs = list(title = "TITLE"),
    theme = list(title = ggplot2::element_text(size = 20))
  ))
)

parse_ggplot2_args(
```

```

resolve_ggplot2_args(
  ggplot2_args(
    labs = list(title = "TITLE"),
    theme = list(title = ggplot2::element_text(size = 20))
  )
),
ggtheme = "gray"
)

```

---

plot\_with\_settings      *Plot-with-settings module*

---

## Description

Universal module for plots with settings for height, width, and download.

## Usage

```

plot_with_settings_ui(id)

plot_with_settings_srv(
  id,
  plot_r,
  height = c(600, 200, 2000),
  width = NULL,
  show_hide_signal = reactive(TRUE),
  brushing = FALSE,
  clicking = FALSE,
  dblclicking = FALSE,
  hovering = FALSE,
  graph_align = "left"
)

```

## Arguments

id	(character(1)) shiny module id.
plot_r	(reactive or function) reactive expression or a simple function to draw a plot. A simple function is needed e.g. for base plots like plot(1) as the output can not be caught when downloading. Take into account that simple functions are less efficient than reactive, as not catching the result.
height	(numeric) optional vector with three elements c(VAL, MIN, MAX), where VAL is the starting value of the slider in the main and expanded plot display.
width	(numeric) optional vector with three elements c(VAL, MIN, MAX), where VAL is the starting value of the slider in the main and expanded plot display; NULL for default display.

show_hide_signal	optional, (reactive logical) a mechanism to allow modules which call this module to show/hide the plot_with_settings UI
brushing	(logical) optional mechanism to enable / disable brushing on the main plot. All the brushing data is stored as a reactive object in the "brush" element of returned list. See the example for details.
clicking	(logical) a mechanism to enable / disable clicking on data points on the main plot. All the clicking data is stored as a reactive object in the "click" element of returned list. See the example for details.
dblclicking	(logical) optional mechanism to enable / disable double-clicking on data points on the main plot. All the double clicking data is stored as a reactive object in the the "dblclick" element of returned list. See the example for details.
hovering	(logical(1)) optional mechanism to enable / disable hovering over data points on the main plot. All the hovering data is stored as a reactive object in the "hover" element of returned list. See the example for details.
graph_align	(character(1)) optional, one of "left" (default), "center", "right" or "justify". The alignment of the graph on the main page.

### Details

By default the plot is rendered with 72 dpi. In order to change this, to for example 96 set `options(teal.plot_dpi = 96)`. The minimum allowed dpi value is 24 and it must be a whole number. If an invalid value is set then the default value is used and a warning is outputted to the console.

### Value

A shiny module.

### Examples

```
# Example using a reactive as input to plot_r
library(shiny)
library(ggplot2)

ui <- bslib::page_fluid(
  plot_with_settings_ui(
    id = "plot_with_settings"
  )
)

server <- function(input, output, session) {
  plot_r <- reactive({
    ggplot(faithful, aes(x = .data$waiting, y = .data$eruptions)) +
```

```

    geom_point()
  })

  plot_with_settings_srv(
    id = "plot_with_settings",
    plot_r = plot_r,
    height = c(400, 100, 1200),
    width = c(500, 250, 750)
  )
}

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

# Example using a function as input to plot_r
library(lattice)

ui <- bslib::page_fluid(
  radioButtons("download_option", "Select the Option", list("ggplot", "trellis", "grob", "base")),
  plot_with_settings_ui(
    id = "plot_with_settings"
  ),
  sliderInput("nums", "Value", 1, 10, 1)
)

server <- function(input, output, session) {
  plot_r <- function() {
    numbers <- seq_len(input$nums)
    if (input$download_option == "ggplot") {
      ggplot(data.frame(n = numbers), aes(.data$n)) +
        geom_bar()
    } else if (input$download_option == "trellis") {
      densityplot(numbers)
    } else if (input$download_option == "grob") {
      tr_plot <- densityplot(numbers)
      ggplotGrob(
        ggplot(data.frame(n = numbers), aes(.data$n)) +
          geom_bar()
      )
    } else if (input$download_option == "base") {
      plot(numbers)
    }
  }
}

plot_with_settings_srv(
  id = "plot_with_settings",
  plot_r = plot_r,
  height = c(400, 100, 1200),
  width = c(500, 250, 750)
)
}

```

```

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

# Example with brushing/hovering/clicking/double-clicking
ui <- bslib::page_fluid(
  plot_with_settings_ui(
    id = "plot_with_settings"
  ),
  fluidRow(
    column(4, tags$h3("Brush"), verbatimTextOutput("brushing_data")),
    column(4, tags$h3("Click"), verbatimTextOutput("clicking_data")),
    column(4, tags$h3("DbClick"), verbatimTextOutput("dblclicking_data")),
    column(4, tags$h3("Hover"), verbatimTextOutput("hovering_data"))
  )
)

server <- function(input, output, session) {
  plot_r <- reactive({
    ggplot(faithful, aes(x = .data$waiting, y = .data$eruptions)) +
      geom_point()
  })

  plot_data <- plot_with_settings_srv(
    id = "plot_with_settings",
    plot_r = plot_r,
    height = c(400, 100, 1200),
    brushing = TRUE,
    clicking = TRUE,
    dblclicking = TRUE,
    hovering = TRUE
  )

  output$brushing_data <- renderPrint(plot_data$brush())
  output$clicking_data <- renderPrint(plot_data$click())
  output$dblclicking_data <- renderPrint(plot_data$dblclick())
  output$hovering_data <- renderPrint(plot_data$hover())
}

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

# Example which allows module to be hidden/shown
library("shinyjs")

ui <- bslib::page_fluid(
  useShinyjs(),
  actionButton("button", "Show/Hide"),
  plot_with_settings_ui(
    id = "plot_with_settings"
  )
)

```

```

server <- function(input, output, session) {
  plot_r <- plot_r <- reactive(
    ggplot(faithful, aes(x = .data$waiting, y = .data$eruptions)) +
    geom_point()
  )

  show_hide_signal_rv <- reactiveVal(TRUE)

  observeEvent(input$button, show_hide_signal_rv(!show_hide_signal_rv()))

  plot_with_settings_srv(
    id = "plot_with_settings",
    plot_r = plot_r,
    height = c(400, 100, 1200),
    width = c(500, 250, 750),
    show_hide_signal = reactive(show_hide_signal_rv())
  )
}

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

```

---

resolve\_basic\_table\_args

*Resolves and reduces multiple basic\_table\_args objects*

---

### Description

Resolving and reducing multiple `basic_table_args` objects. This function is intended to utilize user provided settings, defaults provided by the module creator and also `teal` option. See [Details](#), below, to understand the logic.

### Usage

```

resolve_basic_table_args(
  user_table = basic_table_args(),
  user_default = basic_table_args(),
  module_table = basic_table_args(),
  app_default = getOption("teal.basic_table_args", basic_table_args())
)

```

### Arguments

`user_table` (basic\_table\_args)  
 end user setup for `rtables::basic_table()` of a specific table. Created with the `basic_table_args()` function. The `NULL` value is supported.

user_default	(basic_table_args) end user default setup for <code>rtables::basic_table()</code> of a specific table. Created with the <code>basic_table_args()</code> function. The NULL value is supported.
module_table	(ggplot2_args) module creator setup for <code>rtables::basic_table()</code> of a specific table. Created with the <code>basic_table_args()</code> function. The NULL value is supported.
app_default	(basic_table_args) Application level setting. Can be NULL.

**Details**

The function picks the first non NULL value for each argument, checking in the following order:

1. `basic_table_args` argument provided by the end user. Per table (`user_table`) and then default (`user_default`) setup.
2. `app_default` global R variable, `teal.basic_table_args`.
3. `module_table` which is a module creator setup.

**Value**

`basic_table_args` object.

**See Also**

`parse_basic_table_args()` to parse resolved list into list of calls.

**Examples**

```
resolve_basic_table_args(
  user_table = basic_table_args(title = "TITLE"),
  user_default = basic_table_args(title = "DEFAULT_TITLE", subtitles = "SUBTITLE")
)
```

---

`resolve_ggplot2_args` *Resolving and reducing multiple ggplot2\_args objects*

---

**Description**

Resolving and reducing multiple `ggplot2_args` objects. This function is intended to utilize user provided settings, defaults provided by the module creator and also `teal` option. See Details, below, to understand the logic.

**Usage**

```
resolve_ggplot2_args(
  user_plot = ggplot2_args(),
  user_default = ggplot2_args(),
  module_plot = ggplot2_args(),
  app_default = getOption("teal.ggplot2_args", ggplot2_args())
)
```

## Arguments

<code>user_plot</code>	<code>(ggplot2_args)</code> end user setup for theme and labs in the specific plot. Created with the <code>ggplot2_args()</code> function. The NULL value is supported.
<code>user_default</code>	<code>(ggplot2_args)</code> end user setup for module default theme and labs. Created with the <code>ggplot2_args()</code> function. The NULL value is supported.
<code>module_plot</code>	<code>(ggplot2_args)</code> module creator setup for theme and labs in the specific plot. Created with the <code>ggplot2_args()</code> function. The NULL value is supported.
<code>app_default</code>	<code>(ggplot2_args)</code> Application level setting. Can be NULL.

## Details

The function picks the first non NULL value for each argument, checking in the following order:

1. `ggplot2_args` argument provided by the end user. Per plot (`user_plot`) and then default (`user_default`) setup.
2. `app_default` global R variable, `teal.ggplot2_args`.
3. `module_plot` which is a module creator setup.

## Value

`ggplot2_args` object.

## See Also

[parse\\_ggplot2\\_args\(\)](#) to parse resolved list into list of calls.

## Examples

```
resolve_ggplot2_args(  
  user_plot = ggplot2_args(  
    labs = list(title = "TITLE"),  
    theme = list(title = ggplot2::element_text(size = 20))  
  ),  
  user_default = ggplot2_args(  
    labs = list(x = "XLAB")  
  )  
)
```

---

standard_layout	<i>Standard UI layout</i>
-----------------	---------------------------

---

**Description**

Create a standard UI layout with output on the right and an encoding panel on the left. This is the layout used by the teal modules.

**Usage**

```
standard_layout(
  output,
  encoding = NULL,
  forms = NULL,
  pre_output = NULL,
  post_output = NULL
)
```

**Arguments**

output	(shiny.tag) object with the output element (table, plot, listing) such as for example returned by <code>shiny::plotOutput()</code> .
encoding	(shiny.tag) object containing the encoding elements. If this element is NULL then no encoding side panel on the right is created.
forms	(tagList) for example <code>shiny::actionButton()</code> that are placed below the encodings panel
pre_output	(shiny.tag) optional, with text placed before the output to put the output into context. For example a title.
post_output	(shiny.tag) optional, with text placed after the output to put the output into context. For example the <code>shiny::helpText()</code> elements are useful.

**Value**

an object of class shiny.tag with the UI code.

**Examples**

```
library(shiny)
standard_layout(
  output = white_small_well(tags$h3("Tests")),
  encoding = tags$div(
    tags$label("Encodings", class = "text-primary"),
    panel_item(
      "Tests",
```

```

    optionalSelectInput(
      "tests",
      "Tests:",
      choices = c(
        "Shapiro-Wilk",
        "Kolmogorov-Smirnov (one-sample)"
      ),
      selected = "Shapiro-Wilk"
    )
  ),
  forms = tagList(
    verbatim_popup_ui("warning", "Show Warnings"),
    verbatim_popup_ui("rcode", "Show R code")
  )
)

```

---

table_with_settings	table_with_settings <i>module</i>
---------------------	-----------------------------------

---

### Description

Module designed to create a shiny table output based on table objects. Supports rtables objects (ElementaryTable or TableTree), gtsummary objects, or gt objects.

### Usage

```
table_with_settings_ui(id, ...)
```

```
table_with_settings_srv(id, table_r, show_hide_signal = reactive(TRUE))
```

### Arguments

id	An ID string that corresponds with the ID used to call the module's UI function.
...	(character) Useful for providing additional HTML classes for the output tag.
table_r	(reactive) reactive expression that yields a table object. Supported types: <ul style="list-style-type: none"> <li>• rtables objects (ElementaryTable or TableTree)</li> <li>• gtsummary objects</li> <li>• gt objects (gt_tbl)</li> </ul>
show_hide_signal	(reactive logical) optional mechanism to allow modules which call this module to show/hide the table_with_settings UI.

**Value**

A shiny module.

**Examples**

```
library(shiny)
library(rtables)
library(gtsummary)
library(gt)
library(magrittr)

ui <- bslib::page_fluid(
  table_with_settings_ui(id = "rtables_table"),
  table_with_settings_ui(id = "gtsummary_table"),
  table_with_settings_ui(id = "gt_table")
)

server <- function(input, output, session) {
  table_r_rtables <- reactive({
    l <- basic_table() %>%
      split_cols_by("ARM") %>%
      analyze(c("SEX", "AGE"))
    build_table(l, DM)
  })

  table_r_gtsummary <- reactive({
    gtsummary::tbl_summary(mtcars)
  })

  table_r_gt <- reactive({
    mtcars %>%
      gt::gt() %>%
      gt::tab_header(title = "Motor Trend Car Road Tests")
  })

  table_with_settings_srv(id = "rtables_table", table_r = table_r_rtables)
  table_with_settings_srv(id = "gtsummary_table", table_r = table_r_gtsummary)
  table_with_settings_srv(id = "gt_table", table_r = table_r_gt)
}

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

**Description**

This module consists of a button that once clicked pops up a modal window with verbatim-styled text.

**Usage**

```
verbatim_popup_ui(id, button_label, type = c("button", "link"), ...)
```

```
verbatim_popup_srv(
  id,
  verbatim_content,
  title,
  style = FALSE,
  disabled = shiny::reactiveVal(FALSE)
)
```

**Arguments**

id	(character(1)) the shiny id
button_label	(character(1)) the text printed on the button
type	(character(1)) specifying whether to use [shiny::actionButton()] or [shiny::actionLink()].
...	additional arguments to [shiny::actionButton()](or [shiny::actionLink()]).
verbatim_content	(character, expression, condition or reactive(1) holding any of the above) the content to show in the popup modal window
title	(character(1)) the title of the modal window
style	(logical(1)) whether to style the verbatim_content using styler::style_text. If verbatim_content is a condition or reactive holding condition then this argument is ignored
disabled	(reactive(1)) the shiny reactive value holding a logical. The popup button is disabled when the flag is TRUE and enabled otherwise.

**Value**

the UI function returns a shiny.tag.list object

**Examples**

```
library(shiny)

ui <- bslib::page_fluid(verbatim_popup_ui("my_id", button_label = "Open popup"))
srv <- function(input, output) {
  verbatim_popup_srv(
    "my_id",
    "if (TRUE) { print('Popups are the best') }",
    title = "My custom title",
    style = TRUE
  )
}
```

```
}  
if (interactive()) shinyApp(ui, srv)
```

---

white_small_well	<i>Small well class for HTML</i>
------------------	----------------------------------

---

### Description

Adds Small Well class and overflow-x property to HTML output element.

### Usage

```
white_small_well(...)
```

### Arguments

... other arguments to pass to tag object's div attributes.

### Details

white\_small\_well is intended to be used with [shiny::uiOutput\(\)](#). The overflow-x property is set to auto so that a scroll bar is added when the content overflows at the left and right edges of the output window. For example, this is useful for displaying wide tables.

### Value

An HTML output element with class Small Well and overflow-x property

### Examples

```
white_small_well(shiny::htmlOutput("summary"))
```

# Index

`basic_table_args`, 3  
`basic_table_args()`, 23, 24

`clean_brushedPoints`, 3

`draggable_buckets`, 4

`get_dt_rows`, 6  
`ggplot2::labs()`, 7  
`ggplot2::theme()`, 7  
`ggplot2_args`, 7  
`ggplot2_args()`, 25

`nested_closeable_modal`, 8

`optionalSelectInput`, 9  
`optionalSliderInput`, 13  
`optionalSliderInput()`, 14  
`optionalSliderInputValMinMax`, 14

`panel_group`, 15, 16  
`panel_group()`, 15  
`panel_item`, 15, 16  
`parse_basic_table_args`, 17  
`parse_basic_table_args()`, 3, 24  
`parse_ggplot2_args`, 18  
`parse_ggplot2_args()`, 7, 25  
`pickerOptions`, 10  
`plot_with_settings`, 19  
`plot_with_settings_srv`  
    (`plot_with_settings`), 19  
`plot_with_settings_ui`  
    (`plot_with_settings`), 19

`resolve_basic_table_args`, 23  
`resolve_basic_table_args()`, 3, 17  
`resolve_ggplot2_args`, 24  
`resolve_ggplot2_args()`, 7, 18  
`rtables::basic_table()`, 3, 23, 24

`shiny::actionButton()`, 26

`shiny::helpText()`, 10, 13, 14, 26  
`shiny::plotOutput()`, 26  
`shiny::uiOutput()`, 30  
`shinyWidgets::pickerInput()`, 10  
`standard_layout`, 26

`table_with_settings`, 27  
`table_with_settings_srv`  
    (`table_with_settings`), 27  
`table_with_settings_ui`  
    (`table_with_settings`), 27

`updateOptionalSelectInput`  
    (`optionalSelectInput`), 9

`verbatim_popup`, 28  
`verbatim_popup_srv` (`verbatim_popup`), 28  
`verbatim_popup_ui` (`verbatim_popup`), 28

`white_small_well`, 30