Package 'shinyWidgets'

March 12, 2019

Title Custom Inputs Widgets for Shiny

Version 0.4.7

Description
Collection of custom input controls and user interface components for 'Shiny' applications. Give your applications a unique and colorful style !

URL https://github.com/dreamRs/shinyWidgets

BugReports https://github.com/dreamRs/shinyWidgets/issues

Depends R (>= 3.1.0)

Imports shiny (>= 0.14), htmltools, jsonlite, grDevices, scales

License GPL-3 | file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Suggests shinydashboard, viridisLite, RColorBrewer, testthat, covr, shinydashboardPlus, bs4Dash, argonR, argonDash

NeedsCompilation no

Author Victor Perrier [aut, cre],
Fanny Meyer [aut],
David Granjon [aut] (jQuery Knob shiny interface & sliders appearance),
Ian Fellows [ctb] (Methods for mutating vertical tabs & updateMultiInput),
Wil Davis [ctb] (numericRangeInput function),
SnapAppointments [cph] (bootstrap-select),
Mattia Larentis [ctb, cph] (Bootstrap Switch),
Emanuele Marchi [ctb, cph] (Bootstrap Switch),
Mark Otto [ctb] (Bootstrap library),
Jacob Thornton [ctb] (Bootstrap library),
Bootstrap contributors [ctb] (Bootstrap library),
Twitter, Inc [cph] (Bootstrap library),
Flatlogic [cph] (Awesome Bootstrap Checkbox),
mouse0270 [ctb, cph] (Material Design Switch),

Tristan Edwards [ctb, cph] (SweetAlert),
Fabian Lindfors [ctb, cph] (multi.js),
Anthony Terrien [ctb, cph] (jQuery Knob),
Daniel Eden [ctb, cph] (animate.css),
Ganapati V S [ctb, cph] (bttn.css),
Brian Grinstead [ctb, cph] (Spectrum),
Lokesh Rajendran [ctb, cph] (pretty-checkbox),
Leon Gersen [ctb, cph] (wnumb & noUiSlider),
Timofey Marochkin [ctb, cph] (air-datepicker),
Tobias Ahlin [ctb, cph] (CSS spin)

Maintainer Victor Perrier <victor.perrier@dreamrs.fr>

Repository CRAN

Date/Publication 2019-03-12 10:50:03 UTC

R topics documented:

`actionBttn` *Awesome action button*

Description

Like `actionButton` but awesome, via https://bttn.surge.sh/

Usage

```
actionBttn(inputId, label = NULL, icon = NULL, style = "unite",
  color = "default", size = "md", block = FALSE, no_outline = TRUE)
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`label`	The contents of the button, usually a text label.
`icon`	An optional icon to appear on the button.
`style`	Style of the button, to choose between `simple`, `bordered`, `minimal`, `stretch`, `jelly`, `gradient`, `fill`, `material-circle`, `material-flat`, `pill`, `float`, `unite`.
`color`	Color of the button : `default`, `primary`, `warning`, `danger`, `success`, `royal`.
`size`	Size of the button : `xs`,`sm`, `md`, `lg`.
`block`	Logical, full width button.
`no_outline`	Logical, don't show outline when navigating with keyboard/interact using mouse or touch.

See Also

downloadBttn

Examples

```
## Not run:
if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h2("Awesome action button"),
```

```
  tags$br(),
  actionBttn(
    inputId = "bttn1",
    label = "Go!",
    color = "primary",
    style = "bordered"
  ),
  tags$br(),
  verbatimTextOutput(outputId = "res_bttn1"),
  tags$br(),
  actionBttn(
    inputId = "bttn2",
    label = "Go!",
    color = "success",
    style = "material-flat",
    icon = icon("sliders"),
    block = TRUE
  ),
  tags$br(),
  verbatimTextOutput(outputId = "res_bttn2")
)

server <- function(input, output, session) {
  output$res_bttn1 <- renderPrint(input$bttn1)
  output$res_bttn2 <- renderPrint(input$bttn2)
}

shinyApp(ui = ui, server = server)

}

## End(Not run)
```

`actionGroupButtons` *Actions Buttons Group Inputs*

Description

Create a group of actions buttons.

Usage

```
actionGroupButtons(inputIds, labels, status = "default",
  size = "normal", direction = "horizontal", fullwidth = FALSE)
```

Arguments

`inputIds`	The `inputs` slot that will be used to access the value, one for each button.
`labels`	Labels for each buttons, must have same length as `inputIds`.

`status`	Add a class to the buttons, you can use Bootstrap status like 'info', 'primary', 'danger', 'warning' or 'success'. Or use an arbitrary strings to add a custom class, e.g. : with `status = 'myClass'`, buttons will have class `btn-myClass`.
`size`	Size of the buttons ('xs', 'sm', 'normal', 'lg').
`direction`	Horizontal or vertical.
`fullwidth`	If TRUE, fill the width of the parent div.

Value

An actions buttons group control that can be added to a UI definition.

Examples

```
## Not run:
if (interactive()) {
  library("shiny")
  library("shinyWidgets")

  ui <- fluidPage(
    br(),
    actionGroupButtons(
      inputIds = c("btn1", "btn2", "btn3"),
      labels = list("Action 1", "Action 2", tags$span(icon("gear"), "Action 3")),
      status = "primary"
    ),
    verbatimTextOutput(outputId = "res1"),
    verbatimTextOutput(outputId = "res2"),
    verbatimTextOutput(outputId = "res3")
  )

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

    output$res1 <- renderPrint(input$btn1)

    output$res2 <- renderPrint(input$btn2)

    output$res3 <- renderPrint(input$btn3)

  }

  shinyApp(ui = ui, server = server)
}

## End(Not run)
```

addSpinner *Display a spinner above an output when this one recalculate*

Description

Display a spinner above an output when this one recalculate

Usage

```
addSpinner(output, spin = "double-bounce", color = "#112446")
```

Arguments

output	An output element, typically the result of renderPlot.
spin	Style of the spinner, choice between : circle, bounce, folding-cube, rotating-plane, cube-grid, fading-circle, double-bounce, dots, cube.
color	Color for the spinner.

Value

a list of tags

Note

The spinner don't disappear from the page, it's only masked by the plot, so the plot must have a non-transparent background. For a more robust way to insert loaders, see package "shinycssloaders".

Examples

```
## Not run:
# wrap an output:
addSpinner(plotOutput("plot"))

# Complete demo:

if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h2("Exemple spinners"),
  actionButton(inputId = "refresh", label = "Refresh", width = "100%"),
  fluidRow(
    column(
      width = 5, offset = 1,
      addSpinner(plotOutput("plot1"), spin = "circle", color = "#E41A1C"),
      addSpinner(plotOutput("plot3"), spin = "bounce", color = "#377EB8"),
      addSpinner(plotOutput("plot5"), spin = "folding-cube", color = "#4DAF4A"),
```

```
      addSpinner(plotOutput("plot7"), spin = "rotating-plane", color = "#984EA3"),
      addSpinner(plotOutput("plot9"), spin = "cube-grid", color = "#FF7F00")
    ),
    column(
      width = 5,
      addSpinner(plotOutput("plot2"), spin = "fading-circle", color = "#FFFF33"),
      addSpinner(plotOutput("plot4"), spin = "double-bounce", color = "#A65628"),
      addSpinner(plotOutput("plot6"), spin = "dots", color = "#F781BF"),
      addSpinner(plotOutput("plot8"), spin = "cube", color = "#999999")
    )
  ),
  actionButton(inputId = "refresh2", label = "Refresh", width = "100%")
)

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

  dat <- reactive({
    input$refresh
    input$refresh2
    Sys.sleep(3)
    Sys.time()
  })

  lapply(
    X = seq_len(9),
    FUN = function(i) {
      output[[paste0("plot", i)]] <- renderPlot({
        dat()
        plot(sin, -pi, i*pi)
      })
    }
  )

}

shinyApp(ui, server)

}


## End(Not run)
```

`airDatepicker` *Air Date Picker Input*

Description

An alternative to `dateInput` to select single, multiple or date range. And two alias to select months or years.

Usage

```
airDatepickerInput(inputId, label = NULL, value = NULL,
  multiple = FALSE, range = FALSE, timepicker = FALSE,
  separator = " - ", placeholder = NULL, dateFormat = "yyyy-mm-dd",
  minDate = NULL, maxDate = NULL, disabledDates = NULL,
  view = c("days", "months", "years"), minView = c("days", "months",
  "years"), monthsField = c("monthsShort", "months"),
  clearButton = FALSE, todayButton = FALSE, autoClose = FALSE,
  timepickerOpts = timepickerOptions(), position = NULL,
  update_on = c("change", "close"), addon = c("right", "left", "none"),
  language = "en", inline = FALSE, width = NULL)

timepickerOptions(dateTimeSeparator = NULL, timeFormat = NULL,
  minHours = NULL, maxHours = NULL, minMinutes = NULL,
  maxMinutes = NULL, hoursStep = NULL, minutesStep = NULL)

airMonthpickerInput(inputId, label = NULL, value = NULL, ...)

airYearpickerInput(inputId, label = NULL, value = NULL, ...)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
value	Initial value(s), dates as character string are accepted in yyyy-mm-dd format, or Date/POSIXct object. Can be a single value or several values.
multiple	Select multiple dates.
range	Select a date range.
timepicker	Add a timepicker below calendar to select time.
separator	Separator between dates when several are selected, default to " - ".
placeholder	A character string giving the user a hint as to what can be entered into the control.
dateFormat	Format to use to display date(s), default to "yyyy-mm-dd"
minDate	The minimum allowed date. Either a Date object, or a string in yyyy-mm-dd format.
maxDate	The maximum allowed date. Either a Date object, or a string in yyyy-mm-dd format.
disabledDates	A vector of dates to disable, e.g. won't be able to select one of dates passed.
view	Starting view, one of 'days' (default), 'months' or 'years'.
minView	Minimal view, one of 'days' (default), 'months' or 'years'.
monthsField	Names for the months when view is 'months', use 'monthsShort' for abbreviations or 'months' for full names.
clearButton	If TRUE, then button "Clear" will be visible.

todayButton	If TRUE, then button "Today" will be visible.
autoClose	If TRUE, then after date selection, datepicker will be closed.
timepickerOpts	Options for timepicker, see timepickerOptions.
position	Where calendar should appear, a two word string like 'bottom left' (default), or 'top right', 'left top'.
update_on	When to send selected value to server: on 'change' or when calendar is 'close'd.
addon	Display a calendar icon to 'right' or the 'left' of the widget, or 'none'. This icon act likes an actionButton, you can retrieve value server-side with input$<inputId>_button.
language	Language to use, can be one of 'cs', 'da', 'de', 'en', 'es', 'fi', 'fr', 'hu', 'nl', 'pl', 'pt-BR', \code'pt', 'ro', 'ru', 'sk', 'zh'.
inline	If TRUE, datepicker will always be visible.
width	The width of the input, e.g. '400px', or '100%'.
dateTimeSeparator	Separator between date and time, default to " ".
timeFormat	Desirable time format. You can use h (hours), hh (hours with leading zero), i (minutes), ii (minutes with leading zero), aa (day period - 'am' or 'pm'), AA (day period capitalized)
minHours	Minimal hours value, must be between 0 and 23. You will not be able to choose value lower than this.
maxHours	Maximum hours value, must be between 0 and 23. You will not be able to choose value higher than this.
minMinutes	Minimal minutes value, must be between 0 and 59. You will not be able to choose value lower than this.
maxMinutes	Maximum minutes value, must be between 0 and 59. You will not be able to choose value higher than this.
hoursStep	Hours step in slider.
minutesStep	Minutes step in slider.
...	Arguments passed to airDatepickerInput.

Value

a Date object or a POSIXct in UTC timezone.

Note

This widget prevents dateInput from working, don't use both !

See Also

See updateAirDateInput for updating slider value server-side. And demoAirDatepicker for examples.

Examples

```
## Not run:

if (interactive()) {

# examples of different options to select dates:
demoAirDatepicker("datepicker")

# select month(s)
demoAirDatepicker("months")

# select year(s)
demoAirDatepicker("years")

# select date and time
demoAirDatepicker("timepicker")

# You can select multiple dates :
library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  airDatepickerInput(
    inputId = "multiple",
    label = "Select multiple dates:",
    placeholder = "You can pick 5 dates",
    multiple = 5, clearButton = TRUE
  ),
  verbatimTextOutput("res")
)

server <- function(input, output, session) {
  output$res <- renderPrint(input$multiple)
}

shinyApp(ui, server)

}


## End(Not run)
```

---

animateOptions *Animate options*

Description

Animate options

Usage

```
animateOptions(enter = "fadeInDown", exit = "fadeOutUp",
  duration = 1)
```

Arguments

enter	Animation name on appearance
exit	Animation name on disappearance
duration	Duration of the animation

Value

a list

See Also

animations

Examples

```
## Not run:
## Only run examples in interactive R sessions
if (interactive()) {

dropdown(
 "Your contents goes here ! You can pass several elements",
 circle = TRUE, status = "danger", icon = icon("gear"), width = "300px",
 animate = animateOptions(enter = "fadeInDown", exit = "fadeOutUp", duration = 3)
)

}

## End(Not run)
```

---

animations *Animation names*

Description

List of all animations by categories

Usage

```
animations
```

Format

A list of lists

Source

https://github.com/daneden/animate.css/blob/master/animate-config.json

---

appendVerticalTab *Mutate Vertical Tabset Panel*

Description

Mutate Vertical Tabset Panel

Usage

```
appendVerticalTab(inputId, tab,
  session = shiny::getDefaultReactiveDomain())

removeVerticalTab(inputId, index,
  session = shiny::getDefaultReactiveDomain())

reorderVerticalTabs(inputId, newOrder,
  session = shiny::getDefaultReactiveDomain())
```

Arguments

inputId	The id of the verticalTabsetPanel object.
tab	The verticalTab to append.
session	The session object passed to function given to shinyServer.
index	The index of the the tab to remove.
newOrder	The new index order.

Examples

```
if (interactive()) {
library(shiny)
library(shinyWidgets)

ui <- fluidPage(

  verticalTabsetPanel(
    verticalTabPanel("blaa","foo"),
    verticalTabPanel("yarp","bar"),
    id="hippi"
  )
)

server <- function(input, output, session) {
  appendVerticalTab("hippi", verticalTabPanel("bipi","long"))
  removeVerticalTab("hippi", 1)
```

```
  appendVerticalTab("hippi", verticalTabPanel("howdy","fair"))
  reorderVerticalTabs("hippi", c(3,2,1))
}

# Run the application
shinyApp(ui = ui, server = server)
}
```

`awesomeCheckbox` *Awesome Checkbox Input Control*

Description

Create a Font Awesome Bootstrap checkbox that can be used to specify logical values.

Usage

```
awesomeCheckbox(inputId, label, value = FALSE, status = "primary",
  width = NULL)
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`label`	Input label.
`value`	Initial value (TRUE or FALSE).
`status`	Color of the buttons, a valid Bootstrap status : default, primary, info, success, warning, danger.
`width`	The width of the input

Value

A checkbox control that can be added to a UI definition.

See Also

`updateAwesomeCheckbox`

Examples

```
## Not run:
## Only run examples in interactive R sessions
if (interactive()) {

ui <- fluidPage(
 awesomeCheckbox(inputId = "somevalue",
                 label = "A single checkbox",
                 value = TRUE,
                 status = "danger"),
```

```
 verbatimTextOutput("value")
)
server <- function(input, output) {
  output$value <- renderText({ input$somevalue })
}
shinyApp(ui, server)
}

## End(Not run)
```

---

awesomeCheckboxGroup *Awesome Checkbox Group Input Control*

Description

Create a Font Awesome Bootstrap checkbox that can be used to specify logical values.

Usage

```
awesomeCheckboxGroup(inputId, label, choices, selected = NULL,
  inline = FALSE, status = "primary", width = NULL)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Input label.
choices	List of values to show checkboxes for.
selected	The values that should be initially selected, if any.
inline	If TRUE, render the choices inline (i.e. horizontally)
status	Color of the buttons
width	The width of the input

Value

A checkbox control that can be added to a UI definition.

See Also

updateAwesomeCheckboxGroup

Examples

```
## Not run:
if (interactive()) {


ui <- fluidPage(
  br(),
  awesomeCheckboxGroup(
    inputId = "id1", label = "Make a choice:",
    choices = c("graphics", "ggplot2")
  ),
  verbatimTextOutput(outputId = "res1"),
  br(),
  awesomeCheckboxGroup(
    inputId = "id2", label = "Make a choice:",
    choices = c("base", "dplyr", "data.table"),
    inline = TRUE, status = "danger"
  ),
  verbatimTextOutput(outputId = "res2")
)

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

  output$res1 <- renderPrint({
    input$id1
  })

  output$res2 <- renderPrint({
    input$id2
  })

}

shinyApp(ui = ui, server = server)


}

## End(Not run)
```

`awesomeRadio` *Awesome Radio Buttons Input Control*

Description

Create a set of prettier radio buttons used to select an item from a list.

Usage

```
awesomeRadio(inputId, label, choices, selected = NULL, inline = FALSE,
  status = "primary", checkbox = FALSE, width = NULL)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
choices	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)
selected	The initially selected value (if not specified then defaults to the first value).
inline	If TRUE, render the choices inline (i.e. horizontally).
status	Color of the buttons, a valid Bootstrap status : default, primary, info, success, warning, danger.
checkbox	Logical, render radio like checkboxes (with a square shape).
width	The width of the input, e.g. 400px, or 100%.

Value

A set of radio buttons that can be added to a UI definition.

See Also

updateAwesomeRadio

Examples

```
## Not run:

## Only run examples in interactive R sessions
if (interactive()) {

ui <- fluidPage(
  br(),
  awesomeRadio(
    inputId = "id1", label = "Make a choice:",
    choices = c("graphics", "ggplot2")
  ),
  verbatimTextOutput(outputId = "res1"),
  br(),
  awesomeRadio(
    inputId = "id2", label = "Make a choice:",
    choices = c("base", "dplyr", "data.table"),
    inline = TRUE, status = "danger"
  ),
  verbatimTextOutput(outputId = "res2")
)
```

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

  output$res1 <- renderPrint({
    input$id1
  })

  output$res2 <- renderPrint({
    input$id2
  })

}

shinyApp(ui = ui, server = server)

}

## End(Not run)
```

---

checkboxGroupButtons *Buttons Group checkbox Input Control*

Description

Create buttons grouped that act like checkboxes.

Usage

```
checkboxGroupButtons(inputId, label = NULL, choices = NULL,
  selected = NULL, status = "default", size = "normal",
  direction = "horizontal", justified = FALSE, individual = FALSE,
  checkIcon = list(), width = NULL, choiceNames = NULL,
  choiceValues = NULL)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Input label.
choices	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)
selected	The initially selected value.
status	Add a class to the buttons, you can use Bootstrap status like 'info', 'primary', 'danger', 'warning' or 'success'. Or use an arbitrary strings to add a custom class, e.g. : with `status = 'myClass'`, buttons will have class `btn-myClass`.
size	Size of the buttons ('xs', 'sm', 'normal', 'lg')
direction	Horizontal or vertical.
justified	If TRUE, fill the width of the parent div.

individual — If TRUE, buttons are separated.

checkIcon — A list, if no empty must contain at least one element named 'yes' corresponding to an icon to display if the button is checked.

width — The width of the input, e.g. '400px', or '100%'.

choiceNames, choiceValues — Same as in checkboxGroupInput. List of names and values, respectively, that are displayed to the user in the app and correspond to the each choice (for this reason, choiceNames and choiceValues must have the same length).

Value

A buttons group control that can be added to a UI definition.

See Also

updateCheckboxGroupButtons

Examples

```
## Not run:
## Only run examples in interactive R sessions
if (interactive()) {

ui <- fluidPage(
  checkboxGroupButtons(inputId = "somevalue",
                       label = "Make a choice: ",
                       choices = c("A", "B", "C")),
  verbatimTextOutput("value")
)
server <- function(input, output) {
  output$value <- renderText({ input$somevalue })
}
shinyApp(ui, server)
}

## End(Not run)
```

---

chooseSliderSkin *Theme selector for* sliderInput

Description

Customize the appearance of the original shiny's sliderInput

Usage

```
chooseSliderSkin(skin = c("Shiny", "Flat", "Modern", "Nice", "Simple",
  "HTML5", "Round", "Square"), color = NULL)
```

Arguments

skin	The skin to apply. Choose among 5 different flavors, namely 'Shiny', 'Flat', 'Modern', 'Nice', 'Simple', 'HTML5', 'Round' and 'Square'.
color	A color to apply to all sliders. Works with following skins: 'Shiny', 'Flat', 'Modern', 'HTML5'. For 'Flat' a CSS filter is applied, desired color maybe a little offset.

Note

It is not currently possible to apply multiple themes at the same time.

See Also

See setSliderColor to update the color of your sliderInput.

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)

# With Modern design

ui <- fluidPage(
  chooseSliderSkin("Modern"),
  sliderInput("obs", "Customized single slider:",
              min = 0, max = 100, value = 50
  ),
  sliderInput("obs2", "Customized range slider:",
              min = 0, max = 100, value = c(40, 80)
  ),
  plotOutput("distPlot")
)

server <- function(input, output) {

  output$distPlot <- renderPlot({
    hist(rnorm(input$obs))
  })

}

shinyApp(ui, server)



# Use Flat design & a custom color
```

```
ui <- fluidPage(
  chooseSliderSkin("Flat", color = "#112446"),
  sliderInput("obs", "Customized single slider:",
              min = 0, max = 100, value = 50
  ),
  sliderInput("obs2", "Customized range slider:",
              min = 0, max = 100, value = c(40, 80)
  ),
  sliderInput("obs3", "An other slider:",
              min = 0, max = 100, value = 50
  ),
  plotOutput("distPlot")
)

server <- function(input, output) {

  output$distPlot <- renderPlot({
    hist(rnorm(input$obs))
  })
}

shinyApp(ui, server)

}



## End(Not run)
```

`circleButton` *Circle Action button*

Description

Create a rounded action button.

Usage

```
circleButton(inputId, icon = NULL, status = "default",
  size = "default", ...)
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`icon`	An icon to appear on the button.
`status`	Color of the button.
`size`	Size of the button : default, lg, sm, xs.
`...`	Named attributes to be applied to the button.

closeSweetAlert *Close Sweet Alert*

Description

Close Sweet Alert

Usage

```
closeSweetAlert(session)
```

Arguments

session	The session object passed to function given to shinyServer.

colorSelectorInput *Color Selector Input*

Description

Choose between a restrictive set of colors.

Usage

```
colorSelectorInput(inputId, label, choices, selected = NULL,
  mode = c("radio", "checkbox"), display_label = FALSE, ncol = 10)

colorSelectorExample()

colorSelectorDrop(inputId, label, choices, selected = NULL,
  display_label = FALSE, ncol = 10, circle = TRUE, size = "sm",
  up = FALSE, width = NULL)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
choices	A list of colors, can be a list of named list, see example.
selected	Default selected color, if NULL the first color for mode = 'radio' and none for mode = 'checkbox'
mode	'radio' for only one choice, 'checkbox' for selecting multiple values.
display_label	Display list's names after palette of color.
ncol	If choices is not a list but a vector, go to line after n elements.
circle	Logical, use a circle or a square button

size	Size of the button : default, lg, sm, xs.
up	Logical. Display the dropdown menu above.
width	Width of the dropdown menu content.

Functions

- colorSelectorExample: Examples of use for colorSelectorInput
- colorSelectorDrop: Display a colorSelector in a dropdown button

Examples

```
if (interactive()) {

# Full example
colorSelectorExample()

# Simple example
ui <- fluidPage(
  colorSelectorInput(
    inputId = "mycolor1", label = "Pick a color :",
    choices = c("steelblue", "cornflowerblue",
                "firebrick", "palegoldenrod",
                "forestgreen")
  ),
  verbatimTextOutput("result1")
)

server <- function(input, output, session) {
  output$result1 <- renderPrint({
    input$mycolor1
  })
}

shinyApp(ui = ui, server = server)

}
```

---

confirmSweetAlert *Launch a confirmation dialog*

Description

Launch a popup to ask confirmation to the user

Usage

```
confirmSweetAlert(session, inputId, title = NULL, text = NULL,
  type = NULL, danger_mode = FALSE, btn_labels = c("Cancel",
  "Confirm"), closeOnClickOutside = FALSE, html = FALSE)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The input slot that will be used to access the value.
title	Title of the alert.
text	Text of the alert, can contains HTML tags.
type	Type of the alert : info, success, warning or error.
danger_mode	Logical, activate danger mode (focus on cancel button).
btn_labels	Labels for buttons.
closeOnClickOutside	Decide whether the user should be able to dismiss the modal by clicking outside of it, or not.
html	Does text contains HTML tags ?

See Also

sendSweetAlert, inputSweetAlert

Examples

```
## Not run:

if (interactive()) {

library("shiny")
library("shinyWidgets")


ui <- fluidPage(
  tags$h1("Confirm sweet alert"),
  actionButton(
    inputId = "launch",
    label = "Launch confirmation dialog"
  ),
  verbatimTextOutput(outputId = "res"),
  uiOutput(outputId = "count")
)

server <- function(input, output, session) {
  # Launch sweet alert confirmation
  observeEvent(input$launch, {
    confirmSweetAlert(
      session = session,
      inputId = "myconfirmation",
      type = "warning",
      title = "Want to confirm ?",
      danger_mode = TRUE
    )
  })
```

```
  # raw output
  output$res <- renderPrint(input$myconfirmation)

  # count click
  true <- reactiveVal(0)
  false <- reactiveVal(0)
  observeEvent(input$myconfirmation, {
    if (isTRUE(input$myconfirmation)) {
      x <- true() + 1
      true(x)
    } else {
      x <- false() + 1
      false(x)
    }
  }, ignoreNULL = TRUE)
  output$count <- renderUI({
    tags$span(
      "Confirm:", tags$b(true()),
      tags$br(),
      "Cancel:", tags$b(false())
    )
  })
}

shinyApp(ui, server)




# other options :

ui <- fluidPage(
  tags$h1("Confirm sweet alert"),
  actionButton(
    inputId = "launch1",
    label = "Launch confirmation dialog (with danger mode)"
  ),
  verbatimTextOutput(outputId = "res1"),
  tags$br(),
  actionButton(
    inputId = "launch2",
    label = "Launch confirmation dialog (with normal mode)"
  ),
  verbatimTextOutput(outputId = "res2"),
  tags$br(),
  actionButton(
    inputId = "launch3",
    label = "Launch confirmation dialog (with HTML)"
  ),
  verbatimTextOutput(outputId = "res3")
)
```

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

  observeEvent(input$launch1, {
    confirmSweetAlert(
      session = session,
      inputId = "myconfirmation1",
      type = "warning",
      title = "Want to confirm ?",
      danger_mode = TRUE
    )
  })
  output$res1 <- renderPrint(input$myconfirmation1)

  observeEvent(input$launch2, {
    confirmSweetAlert(
      session = session,
      inputId = "myconfirmation2",
      type = "warning",
      title = "Are you sure ??",
      btn_labels = c("Nope", "Yep"),
      danger_mode = FALSE
    )
  })
  output$res2 <- renderPrint(input$myconfirmation2)

  observeEvent(input$launch3, {
    confirmSweetAlert(
      session = session,
      inputId = "myconfirmation3",
      title = NULL,
      text = tags$b(
        icon("file"),
        "Do you really want to delete this file ?",
        style = "color: #FA5858;"
      ),
      btn_labels = c("Cancel", "Delete file"),
      danger_mode = TRUE, html = TRUE
    )
  })
  output$res3 <- renderPrint(input$myconfirmation3)

}

shinyApp(ui = ui, server = server)

}


## End(Not run)
```

---

demoAirDatepicker *Some examples on how to use airDatepickerInput*

Description

Some examples on how to use airDatepickerInput

Usage

```
demoAirDatepicker(example = "datepicker")
```

Arguments

example	Name of the example : "datepicker", "timepicker", "months", "years", "update".

Examples

```
## Not run:

if (interactive()) {

demoAirDatepicker("datepicker")

}


## End(Not run)
```

---

demoNoUiSlider *Some examples on how to use noUiSliderInput*

Description

Some examples on how to use noUiSliderInput

Usage

```
demoNoUiSlider(example = "color")
```

Arguments

example	Name of the example : "color", "update", "behaviour", "more", "format".

Examples

```
## Not run:

if (interactive()) {

demoNoUiSlider("color")

}
```

```
## End(Not run)
```

---

demoNumericRange *An example showing how numericRangeInput works*

Description

An example showing how numericRangeInput works

Usage

```
demoNumericRange()
```

Examples

```
## Not run:

if (interactive()) {

demoNumericRange()

}


## End(Not run)
```

---

downloadBttn *Create a download* actionBttn

Description

Create a download button with actionBttn.

Usage

```
downloadBttn(outputId, label = "Download", style = "unite",
  color = "default", size = "md", block = FALSE, no_outline = TRUE)
```

Arguments

outputId	The name of the output slot that the downloadHandler is assigned to.
label	The label that should appear on the button.
style	Style of the button, to choose between simple, bordered, minimal, stretch, jelly, gradient, fill, material-circle, material-flat, pill, float, unite.
color	Color of the button : default, primary, warning, danger, success, royal.
size	Size of the button : xs,sm, md, lg.
block	Logical, full width button.
no_outline	Logical, don't show outline when navigating with keyboard/interact using mouse or touch.

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h2("Download bttn"),
  downloadBttn(
    outputId = "downloadData",
    style = "bordered",
    color = "primary"
  )
)

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

  output$downloadData <- downloadHandler(
    filename = function() {
      paste('data-', Sys.Date(), '.csv', sep='')
    },
    content = function(con) {
      write.csv(mtcars, con)
    }
  )

}

shinyApp(ui, server)

}


## End(Not run)
```

dropdown *Dropdown*

Description

Create a dropdown menu

Usage

```
dropdown(..., style = "default", status = "default", size = "md",
  icon = NULL, label = NULL, tooltip = FALSE, right = FALSE,
  up = FALSE, width = NULL, animate = FALSE, inputId = NULL)
```

Arguments

...	List of tag to be displayed into the dropdown menu.
style	Character. if default use Bootstrap button (like an actionButton), else use an actionBttn, see argument style (in actionBttn documentation) for possible values.
status	Add a class to the buttons, you can use Bootstrap status like 'info', 'primary', 'danger', 'warning' or 'success'. Or use an arbitrary strings to add a custom class, e.g. : with status = 'myClass', buttons will have class btn-myClass.
size	Size of the button : default, lg, sm, xs.
icon	An icon to appear on the button.
label	Label to appear on the button. If circle = TRUE and tooltip = TRUE, label is used in tooltip.
tooltip	Put a tooltip on the button, you can customize tooltip with tooltipOptions.
right	Logical. The dropdown menu starts on the right.
up	Logical. Display the dropdown menu above.
width	Width of the dropdown menu content.
animate	Add animation on the dropdown, can be logical or result of animateOptions.
inputId	Optional, id for the button, the button act like an actionButton, and you can use the id to toggle the dropdown menu server-side.

Details

This function is similar to dropdownButton but don't use Bootstrap, so you can put pickerInput in it. Moreover you can add animations on the appearance / disappearance of the dropdown with animate.css.

See Also

animateOptions for animation, tooltipOptions for tooltip and actionBttn for the button.

Examples

```
## Not run:
## Only run examples in interactive R sessions
if (interactive()) {

library("shiny")
library("shinyWidgets")

ui <- fluidPage(
  tags$h2("pickerInput in dropdown"),
  br(),
  dropdown(

    tags$h3("List of Input"),

    pickerInput(inputId = 'xcol2',
                label = 'X Variable',
                choices = names(iris),
                options = list(`style` = "btn-info")),

    pickerInput(inputId = 'ycol2',
                label = 'Y Variable',
                choices = names(iris),
                selected = names(iris)[[2]],
                options = list(`style` = "btn-warning")),

    sliderInput(inputId = 'clusters2',
                label = 'Cluster count',
                value = 3,
                min = 1, max = 9),

    style = "unite", icon = icon("gear"),
    status = "danger", width = "300px",
    animate = animateOptions(
      enter = animations$fading_entrances$fadeInLeftBig,
      exit = animations$fading_exits$fadeOutRightBig
    )
  ),

  plotOutput(outputId = 'plot2')
)

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

  selectedData2 <- reactive({
    iris[, c(input$xcol2, input$ycol2)]
  })

  clusters2 <- reactive({
    kmeans(selectedData2(), input$clusters2)
  })
```

```
  output$plot2 <- renderPlot({
    palette(c("#E41A1C", "#377EB8", "#4DAF4A",
              "#984EA3", "#FF7F00", "#FFFF33",
              "#A65628", "#F781BF", "#999999"))

    par(mar = c(5.1, 4.1, 0, 1))
    plot(selectedData2(),
         col = clusters2()$cluster,
         pch = 20, cex = 3)
    points(clusters2()$centers, pch = 4, cex = 4, lwd = 4)
  })

}

shinyApp(ui = ui, server = server)

}

## End(Not run)
```

`dropdownButton` *Dropdown Button*

Description

Create a dropdown menu with Bootstrap where you can put input elements.

Usage

```
dropdownButton(..., circle = TRUE, status = "default",
  size = "default", icon = NULL, label = NULL, tooltip = FALSE,
  right = FALSE, up = FALSE, width = NULL, margin = "10px",
  inputId = NULL)
```

Arguments

`...`	List of tag to be displayed into the dropdown menu.
`circle`	Logical. Use a circle button
`status`	Add a class to the buttons, you can use Bootstrap status like 'info', 'primary', 'danger', 'warning' or 'success'. Or use an arbitrary strings to add a custom class, e.g. : with `status = 'myClass'`, buttons will have class `btn-myClass`.
`size`	Size of the button : default, lg, sm, xs.
`icon`	An icon to appear on the button.
`label`	Label to appear on the button. If circle = TRUE and tooltip = TRUE, label is used in tooltip.
`tooltip`	Put a tooltip on the button, you can customize tooltip with `tooltipOptions`.
`right`	Logical. The dropdown menu starts on the right.

up	Logical. Display the dropdown menu above.
width	Width of the dropdown menu content.
margin	Value of the dropdown margin-right and margin-left menu content.
inputId	Optional, id for the button, the button act like an actionButton, and you can use the id to toggle the dropdown menu server-side with toggleDropdownButton.

Details

It is possible to know if a dropdown is open or closed server-side with input$<inputId>_state.

Note

pickerInput doesn't work inside dropdownButton because that's also a dropdown and you can't nest them. Instead use dropdown, it has similar features but is built differently so it works.

Examples

```
## Not run:
## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  dropdownButton(
    inputId = "mydropdown",
    label = "Controls",
    icon = icon("sliders"),
    status = "primary",
    circle = FALSE,
    sliderInput(
      inputId = "n",
      label = "Number of observations",
      min = 10, max = 100, value = 30
    ),
    prettyToggle(
      inputId = "na",
      label_on = "NAs keeped",
      label_off = "NAs removed",
      icon_on = icon("check"),
      icon_off = icon("remove")
    )
  ),
  tags$div(style = "height: 140px;"), # spacing
  verbatimTextOutput(outputId = "out"),
  verbatimTextOutput(outputId = "state")
)

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

```
  output$out <- renderPrint({
    cat(
      " # n\n", input$n, "\n",
      "# na\n", input$na
    )
  })

  output$state <- renderPrint({
    cat("Open:", input$mydropdown_state)
  })

}

shinyApp(ui, server)

}

## End(Not run)
```

`inputSweetAlert` *Launch an input text dialog*

Description

Launch a popup with a text input

Usage

```
inputSweetAlert(session, inputId, title = NULL, text = NULL,
  type = NULL, btn_labels = "Ok", placeholder = NULL)
```

Arguments

`session`	The `session` object passed to function given to shinyServer.
`inputId`	The `input` slot that will be used to access the value.
`title`	Title of the alert.
`text`	Text of the alert.
`type`	Type of the alert : info, success, warning or error.
`btn_labels`	Labels for button(s).
`placeholder`	A character string giving the user a hint as to what can be entered into the control.

See Also

`sendSweetAlert`, `confirmSweetAlert`

Examples

```
## Not run:

if (interactive()) {

library("shiny")
library("shinyWidgets")


ui <- fluidPage(
  tags$h1("Confirm sweet alert"),
  actionButton(inputId = "go", label = "Launch input text dialog"),
  verbatimTextOutput(outputId = "res")
)
server <- function(input, output, session) {

  observeEvent(input$go, {
    inputSweetAlert(
      session = session, inputId = "mytext",
      title = "What's your name ?"
    )
  })

  output$res <- renderPrint(input$mytext)

}

shinyApp(ui = ui, server = server)

}


## End(Not run)
```

---

knobInput *Knob Input*

Description

Knob Input

Usage

```
knobInput(inputId, label, value, min = 0, max = 100, step = 1,
  angleOffset = 0, angleArc = 360, cursor = FALSE,
  thickness = NULL, lineCap = c("default", "round"),
  displayInput = TRUE, displayPrevious = FALSE,
  rotation = c("clockwise", "anticlockwise"), fgColor = NULL,
  inputColor = NULL, bgColor = NULL, readOnly = FALSE, skin = NULL,
  width = NULL, height = NULL, immediate = TRUE)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
value	Initial value.
min	Minimum allowed value, default to 0.
max	Maximum allowed value, default to 100.
step	Specifies the interval between each selectable value, default to 1.
angleOffset	Starting angle in degrees, default to 0.
angleArc	Arc size in degrees, default to 360.
cursor	Display mode "cursor", don't work properly if width is not set in pixel, (TRUE or FALSE).
thickness	Gauge thickness, numeric value.
lineCap	Gauge stroke endings, 'default' or 'round'.
displayInput	Hide input in the middle of the knob (TRUE or FALSE).
displayPrevious	Display the previous value with transparency (TRUE or FALSE).
rotation	Direction of progression, 'clockwise' or 'anticlockwise'.
fgColor	Foreground color.
inputColor	Input value (number) color.
bgColor	Background color.
readOnly	Disable knob (TRUE or FALSE).
skin	Change Knob skin, only one option available : 'tron'.
width	The width of the input, e.g. 400px, or 100%.
height	The height of the input, e.g. 400px, or 100%.
immediate	If TRUE (default), server-side value is updated each time value change, if FALSE value is updated when user release the widget.

Value

Numeric value server-side.

See Also

updateKnobInput for updating the value server-side.

Examples

```
## Not run:

if (interactive()) {

library("shiny")
library("shinyWidgets")
```

```
ui <- fluidPage(
  knobInput(
    inputId = "myKnob",
    label = "Display previous:",
    value = 50,
    min = -100,
    displayPrevious = TRUE,
    fgColor = "#428BCA",
    inputColor = "#428BCA"
  ),
  verbatimTextOutput(outputId = "res")
)

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

  output$res <- renderPrint(input$myKnob)

}

shinyApp(ui = ui, server = server)

}


## End(Not run)
```

`materialSwitch` *Material Design Switch Input Control*

Description

A toggle switch to turn a selection on or off.

Usage

```
materialSwitch(inputId, label = NULL, value = FALSE,
  status = "default", right = FALSE, inline = FALSE, width = NULL)
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`label`	Input label.
`value`	TRUE or FALSE.
`status`	Color, must be a valid Bootstrap status : default, primary, info, success, warning, danger.
`right`	Should the the label be on the right? default to FALSE.
`inline`	Display the input inline, if you want to place buttons next to each other.
`width`	The width of the input, e.g. '400px', or '100%'.

Value

A switch control that can be added to a UI definition.

See Also

updateMaterialSwitch, switchInput

Examples

```
materialSwitch(inputId = "somevalue", label = "")
## Not run:
## Only run examples in interactive R sessions
if (interactive()) {

ui <- fluidPage(
  materialSwitch(inputId = "somevalue", label = ""),
  verbatimTextOutput("value")
)
server <- function(input, output) {
  output$value <- renderText({ input$somevalue })
}
shinyApp(ui, server)
}

## End(Not run)
```

---

multiInput *Create a multiselect input control*

Description

A user-friendly replacement for select boxes with the multiple attribute

Usage

```
multiInput(inputId, label, choices = NULL, selected = NULL,
  options = NULL, width = NULL, choiceNames = NULL,
  choiceValues = NULL)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
choices	List of values to select from.
selected	The initially selected value.

options	List of options passed to multi (enable_search = FALSE for disabling the search bar for example).
width	The width of the input, e.g. 400px, or 100%.
choiceNames	List of names to display to the user.
choiceValues	List of values corresponding to choiceNames.

Value

A multiselect control

See Also

updateMultiInput to update value server-side.

Examples

```
## Not run:
## Only run examples in interactive R sessions
if (interactive()) {

library("shiny")
library("shinyWidgets")


# simple use

ui <- fluidPage(
  multiInput(
    inputId = "id", label = "Fruits :",
    choices = c("Banana", "Blueberry", "Cherry",
                "Coconut", "Grapefruit", "Kiwi",
                "Lemon", "Lime", "Mango", "Orange",
                "Papaya"),
    selected = "Banana", width = "350px"
  ),
  verbatimTextOutput(outputId = "res")
)

server <- function(input, output, session) {
  output$res <- renderPrint({
    input$id
  })
}

shinyApp(ui = ui, server = server)


# with options

ui <- fluidPage(
  multiInput(
```

```
    inputId = "id", label = "Fruits :",
    choices = c("Banana", "Blueberry", "Cherry",
                "Coconut", "Grapefruit", "Kiwi",
                "Lemon", "Lime", "Mango", "Orange",
                "Papaya"),
    selected = "Banana", width = "400px",
    options = list(
      enable_search = FALSE,
      non_selected_header = "Choose between:",
      selected_header = "You have selected:"
    )
  ),
  verbatimTextOutput(outputId = "res")
)

server <- function(input, output, session) {
  output$res <- renderPrint({
    input$id
  })
}

shinyApp(ui = ui, server = server)

}

## End(Not run)
```

---

noUiSliderInput *Numeric range slider*

Description

A minimal numeric range slider with a lot of features.

Usage

```
noUiSliderInput(inputId, label = NULL, min, max, value, step = NULL,
  tooltips = TRUE, connect = TRUE, padding = 0, margin = NULL,
  limit = NULL, orientation = c("horizontal", "vertical"),
  direction = c("ltr", "rtl"), behaviour = "tap", range = NULL,
  pips = NULL, format = wNumbFormat(), update_on = c("end",
  "change"), color = NULL, inline = FALSE, width = NULL,
  height = NULL)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
min	Minimal value that can be selected.

max	Maximal value that can be selected.
value	The initial value of the slider. as many cursors will be created as values provided.
step	numeric, by default, the slider slides fluently. In order to make the handles jump between intervals, you can use the step option.
tooltips	logical, display slider's value in a tooltip above slider.
connect	logical, vector of length value + 1, color slider between handle(s).
padding	numeric, padding limits how close to the slider edges handles can be.
margin	numeric, when using two handles, the minimum distance between the handles can be set using the margin option.
limit	numeric, the limit option is the opposite of the margin option, limiting the maximum distance between two handles.
orientation	The orientation setting can be used to set the slider to "vertical" or "horizontal".
direction	"ltr" or "rtl", By default the sliders are top-to-bottom and left-to-right, but you can change this using the direction option, which decides where the upper side of the slider is.
behaviour	Option to handle user interaction, a value or several between "drag", "tap", "fixed", "snap" or "none". See https://refreshless.com/nouislider/behaviour-option/ for more examples.
range	list, can be used to define non-linear sliders.
pips	list, used to generate points along the slider.
format	numbers format, see wNumbFormat.
update_on	When to send value to server: "end" (when slider is released) or "update" (each time value changes).
color	color in Hex format for the slider.
inline	If TRUE, it's possible to position sliders side-by-side.
width	The width of the input, e.g. 400px, or 100%.
height	The height of the input, e.g. 400px, or 100%.

Value

a ui definition

Note

See updateNoUiSliderInput for updating slider value server-side. And demoNoUiSlider for examples.

Examples

```
## Not run:

if (interactive()) {

### examples ----
```

```
# see ?demoNoUiSlider
demoNoUiSlider("more")


###  basic usage ----

library( shiny )
library( shinyWidgets )


ui <- fluidPage(

  tags$br(),

  noUiSliderInput(
    inputId = "noui1",
    min = 0, max = 100,
    value = 20
  ),
  verbatimTextOutput(outputId = "res1"),

  tags$br(),

  noUiSliderInput(
    inputId = "noui2", label = "Slider vertical:",
    min = 0, max = 1000, step = 50,
    value = c(100, 400), margin = 100,
    orientation = "vertical",
    width = "100px", height = "300px"
  ),
  verbatimTextOutput(outputId = "res2")

)

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

  output$res1 <- renderPrint(input$noui1)
  output$res2 <- renderPrint(input$noui2)

}

shinyApp(ui, server)


}


## End(Not run)
```

---

numericRangeInput *Numeric Range Input*

Description

Create an input group of numeric inputs that function as a range input.

Usage

```
numericRangeInput(inputId, label, value, width = NULL,
  separator = " to ")
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
value	The initial value(s) for the range. A numeric vector of length one will be duplicated to represent the minimum and maximum of the range; a numeric vector of two or more will have its minimum and maximum set the minimum and maximum of the range.
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit.
separator	String to display between the start and end input boxes.

Examples

```
## Not run:

if (interactive()) {

### examples ----

# see ?demoNumericRange
demoNumericRange()


###  basic usage ----

library( shiny )
library( shinyWidgets )


ui <- fluidPage(

  tags$br(),

  numericRangeInput(
    inputId = "noui1", label = "Numeric Range Input:",
    value = c(100, 400)
  ),
  verbatimTextOutput(outputId = "res1")

)

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

```
  output$res1 <- renderPrint(input$noui1)

}

shinyApp(ui, server)


}


## End(Not run)
```

---

panel *Create a panel*

Description

Create a panel (box) with basic border and padding, you can use Bootstrap status to style the panel, see http://getbootstrap.com/components/#panels.

Usage

```
panel(..., heading = NULL, footer = NULL, status = "default")
```

Arguments

...	UI elements to include inside the panel.
heading	Title for the panel in a plain header.
footer	Footer for the panel.
status	Bootstrap status for contextual alternative.

Value

A UI definition.

Examples

```
## Not run:

if (interactive()) {
library("shiny")
library("shinyWidgets")

ui <- fluidPage(

  # Default
  panel(
```

```
    "Content goes here",
    checkboxInput(inputId = "id1", label = "Label")
  ),

  # With header and footer
  panel(
    "Content goes here",
    checkboxInput(inputId = "id2", label = "Label"),
    heading = "My title",
    footer = "Something"
  ),

  # With status
  panel(
    "Content goes here",
    checkboxInput(inputId = "id3", label = "Label"),
    heading = "My title",
    status = "primary"
  )
)

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

}

shinyApp(ui = ui, server = server)
}


## End(Not run)
```

`pickerGroup-module` *Picker Group*

Description

Group of mutually dependent `pickerInput` for filtering `data.frame`'s columns.

Usage

```
pickerGroupUI(id, params, label = NULL, btn_label = "Reset filters",
  options = list())

pickerGroupServer(input, output, session, data, vars)
```

Arguments

`id`	Module's id.
`params`	A named list of parameters passed to each `pickerInput`, you can use : 'inputId' (obligatory, must be variable name), 'label', 'placeholder'.

label	Character, global label on top of all labels.
btn_label	Character, reset button label.
options	See pickerInput options argument.
input	standard shiny input.
output	standard shiny output.
session	standard shiny session.
data	a data.frame, or an object that can be coerced to data.frame.
vars	character, columns to use to create filters, must correspond to variables listed in params.

Value

a reactive function containing data filtered.

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)


data("mpg", package = "ggplot2")


ui <- fluidPage(
  fluidRow(
    column(
      width = 10, offset = 1,
      tags$h3("Filter data with picker group"),
      panel(
        pickerGroupUI(
          id = "my-filters",
          params = list(
            manufacturer = list(inputId = "manufacturer", title = "Manufacturer:"),
            model = list(inputId = "model", title = "Model:"),
            trans = list(inputId = "trans", title = "Trans:"),
            class = list(inputId = "class", title = "Class:")
          )
        ), status = "primary"
      ),
      dataTableOutput(outputId = "table")
    )
  )
)

server <- function(input, output, session) {
  res_mod <- callModule(
```

```
    module = pickerGroupServer,
    id = "my-filters",
    data = mpg,
    vars = c("manufacturer", "model", "trans", "class")
  )
  output$table <- renderDataTable(res_mod())
}

shinyApp(ui, server)

}


## End(Not run)
```

`pickerInput` *Select picker Input Control*

Description

Create a select picker (https://developer.snapappointments.com/bootstrap-select/)

Usage

```
pickerInput(inputId, label = NULL, choices, selected = NULL,
  multiple = FALSE, options = list(), choicesOpt = NULL,
  width = NULL, inline = FALSE)
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`label`	Display a text in the center of the switch.
`choices`	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.
`selected`	The initially selected value (or multiple values if multiple = TRUE). If not specified then defaults to the first value for single-select lists and no values for multiple select lists.
`multiple`	Is selection of multiple items allowed?
`options`	List of options, see pickerOptions for all available options. For limit the number of selections, see example below.
`choicesOpt`	Options for choices in the dropdown menu.
`width`	The width of the input : 'auto', 'fit', '100px', '75%'.
`inline`	Put the label and the picker on the same line.

Value

A select control that can be added to a UI definition.

See Also

updatePickerInput to update value server-side.

Examples

```
## Not run:
## Only run examples in interactive R sessions
if (interactive()) {

# You can run the gallery to see other examples
shinyWidgetsGallery()


# Simple example
library("shiny")
ui <- fluidPage(
  pickerInput(inputId = "somevalue", label = "A label", choices = c("a", "b")),
  verbatimTextOutput("value")
)
server <- function(input, output) {
  output$value <- renderPrint({ input$somevalue })
}
shinyApp(ui, server)


### Add actions box for selecting
# deselecting all options

library("shiny")
library("shinyWidgets")

ui <- fluidPage(
  br(),
  pickerInput(
    inputId = "p1",
    label = "Select all option",
    choices = rownames(mtcars),
    multiple = TRUE,
    options = list(`actions-box` = TRUE)
  ),
  br(),
  pickerInput(
    inputId = "p2",
    label = "Select all option / custom text",
    choices = rownames(mtcars),
    multiple = TRUE,
    options = list(
      `actions-box` = TRUE,
      `deselect-all-text` = "None...",
      `select-all-text` = "Yeah, all !",
      `none-selected-text` = "zero"
    )
```

```
  )
)

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

}

shinyApp(ui = ui, server = server)



### Customize the values displayed in the box ----

library("shiny")
library("shinyWidgets")

ui <- fluidPage(
  br(),
  pickerInput(
    inputId = "p1",
    label = "Default",
    multiple = TRUE,
    choices = rownames(mtcars),
    selected = rownames(mtcars)[1:5]
  ),
  br(),
  pickerInput(
    inputId = "p1b",
    label = "Default with | separator",
    multiple = TRUE,
    choices = rownames(mtcars),
    selected = rownames(mtcars)[1:5],
    options = list(`multiple-separator` = " | ")
  ),
  br(),
  pickerInput(
    inputId = "p2",
    label = "Static",
    multiple = TRUE,
    choices = rownames(mtcars),
    selected = rownames(mtcars)[1:5],
    options = list(`selected-text-format`= "static",
                   title = "Won't change")
  ),
  br(),
  pickerInput(
    inputId = "p3",
    label = "Count",
    multiple = TRUE,
    choices = rownames(mtcars),
    selected = rownames(mtcars)[1:5],
    options = list(`selected-text-format`= "count")
  ),
```

```
  br(),
  pickerInput(
    inputId = "p3",
    label = "Customize count",
    multiple = TRUE,
    choices = rownames(mtcars),
    selected = rownames(mtcars)[1:5],
    options = list(
      `selected-text-format`= "count",
      `count-selected-text` = "{0} models choosed (on a total of {1})"
    )
  )
)

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

}

shinyApp(ui = ui, server = server)



### Limit the number of selections ----

library(shiny)
library(shinyWidgets)
ui <- fluidPage(
  pickerInput(
    inputId = "groups",
    label = "Select one from each group below:",
    choices = list(
      Group1 = c("1", "2", "3", "4"),
      Group2 = c("A", "B", "C", "D")
    ),
    multiple = TRUE,
    options =  list("max-options-group" = 1)
  ),
  verbatimTextOutput(outputId = "res_grp"),
  pickerInput(
    inputId = "groups_2",
    label = "Select two from each group below:",
    choices = list(
      Group1 = c("1", "2", "3", "4"),
      Group2 = c("A", "B", "C", "D")
    ),
    multiple = TRUE,
    options =  list("max-options-group" = 2)
  ),
  verbatimTextOutput(outputId = "res_grp_2"),
  pickerInput(
    inputId = "classic",
    label = "Select max two option below:",
    choices = c("A", "B", "C", "D"),
```

```
    multiple = TRUE,
    options =  list(
      "max-options" = 2,
      "max-options-text" = "No more!"
    )
  ),
  verbatimTextOutput(outputId = "res_classic")
)
server <- function(input, output) {
  output$res_grp <- renderPrint(input$groups)
  output$res_grp_2 <- renderPrint(input$groups_2)
  output$res_classic <- renderPrint(input$classic)
}
shinyApp(ui, server)

}

## End(Not run)
```

`pickerOptions` *Options for 'pickerInput'*

Description

Wrapper of options available here: https://developer.snapappointments.com/bootstrap-select/options/

Usage

```
pickerOptions(actionsBox = NULL, container = NULL,
  countSelectedText = NULL, deselectAllText = NULL,
  dropdownAlignRight = NULL, dropupAuto = NULL, header = NULL,
  hideDisabled = NULL, iconBase = NULL, liveSearch = NULL,
  liveSearchNormalize = NULL, liveSearchPlaceholder = NULL,
  liveSearchStyle = NULL, maxOptions = NULL, maxOptionsText = NULL,
  mobile = NULL, multipleSeparator = NULL, noneSelectedText = NULL,
  noneResultsText = NULL, selectAllText = NULL,
  selectedTextFormat = NULL, selectOnTab = NULL, showContent = NULL,
  showIcon = NULL, showSubtext = NULL, showTick = NULL,
  size = NULL, style = NULL, tickIcon = NULL, title = NULL,
  virtualScroll = NULL, width = NULL, windowPadding = NULL)
```

Arguments

`actionsBox`	When set to true, adds two buttons to the top of the dropdown menu (Select All & Deselect All). Type: boolean; Default: false.
`container`	When set to a string, appends the select to a specific element or selector, e.g., container: 'body' | '.main-body' Type: string | false; Default: false.

countSelectedText
: Sets the format for the text displayed when selectedTextFormat is count or count > #. 0 is the selected amount. 1 is total available for selection. When set to a function, the first parameter is the number of selected options, and the second is the total number of options. The function must return a string. Type: string | function; Default: function.

deselectAllText
: The text on the button that deselects all options when actionsBox is enabled. Type: string; Default: 'Deselect All'.

dropdownAlignRight
: Align the menu to the right instead of the left. If set to 'auto', the menu will automatically align right if there isn't room for the menu's full width when aligned to the left. Type: boolean | 'auto'; Default: false.

dropupAuto
: checks to see which has more room, above or below. If the dropup has enough room to fully open normally, but there is more room above, the dropup still opens normally. Otherwise, it becomes a dropup. If dropupAuto is set to false, dropups must be called manually. Type: boolean; Default: true.

header
: adds a header to the top of the menu; includes a close button by default Type: string; Default: false.

hideDisabled
: removes disabled options and optgroups from the menu data-hide-disabled: true Type: boolean; Default: false.

iconBase
: Set the base to use a different icon font instead of Glyphicons. If changing iconBase, you might also want to change tickIcon, in case the new icon font uses a different naming scheme. Type: string; Default: 'glyphicon'.

liveSearch
: When set to true, adds a search box to the top of the selectpicker dropdown. Type: boolean; Default: false.

liveSearchNormalize
: Setting liveSearchNormalize to true allows for accent-insensitive searching. Type: boolean; Default: false.

liveSearchPlaceholder
: When set to a string, a placeholder attribute equal to the string will be added to the liveSearch input. Type: string; Default: null.

liveSearchStyle
: When set to 'contains', searching will reveal options that contain the searched text. For example, searching for pl with return both Apple, Plum, and Plantain. When set to 'startsWith', searching for pl will return only Plum and Plantain. Type: string; Default: 'contains'.

maxOptions
: When set to an integer and in a multi-select, the number of selected options cannot exceed the given value. This option can also exist as a data-attribute for an <optgroup>, in which case it only applies to that <optgroup>. Type: integer | false; Default: false.

maxOptionsText
: The text that is displayed when maxOptions is enabled and the maximum number of options for the given scenario have been selected. If a function is used, it must return an array. array[0] is the text used when maxOptions is applied to the entire select element. array[1] is the text used when maxOptions is used on an optgroup. If a string is used, the same text is used for both the element and the optgroup. Type: string | array | function; Default: function.

`mobile` When set to true, enables the device's native menu for select menus. Type: boolean; Default: false.

`multipleSeparator`
Set the character displayed in the button that separates selected options. Type: string; Default: ', '.

`noneSelectedText`
The text that is displayed when a multiple select has no selected options. Type: string; Default: 'Nothing selected'.

`noneResultsText`
The text displayed when a search doesn't return any results. Type: string; Default: 'No results matched 0'.

`selectAllText` The text on the button that selects all options when actionsBox is enabled. Type: string; Default: 'Select All'.

`selectedTextFormat`
Specifies how the selection is displayed with a multiple select. 'values' displays a list of the selected options (separated by multipleSeparator. 'static' simply displays the select element's title. 'count' displays the total number of selected options. 'count > x' behaves like 'values' until the number of selected options is greater than x; after that, it behaves like 'count'. Type: 'values' | 'static' | 'count' | 'count > x' (where x is an integer); Default: 'values'.

`selectOnTab` When set to true, treats the tab character like the enter or space characters within the selectpicker dropdown. Type: boolean; Default: false.

`showContent` When set to true, display custom HTML associated with selected option(s) in the button. When set to false, the option value will be displayed instead. Type: boolean; Default: true.

`showIcon` When set to true, display icon(s) associated with selected option(s) in the button. Type: boolean; Default: true.

`showSubtext` When set to true, display subtext associated with a selected option in the button. Type: boolean; Default: false.

`showTick` Show checkmark on selected option (for items without multiple attribute). Type: boolean; Default: false.

`size` When set to 'auto', the menu always opens up to show as many items as the window will allow without being cut off. When set to an integer, the menu will show the given number of items, even if the dropdown is cut off. When set to false, the menu will always show all items. Type: 'auto' | integer | false; Default: 'auto'.

`style` When set to a string, add the value to the button's style. Type: string | null; Default: null.

`tickIcon` Set which icon to use to display as the "tick" next to selected options. Type: string; Default: 'glyphicon-ok'.

`title` The default title for the selectpicker. Type: string | null; Default: null.

`virtualScroll` If enabled, the items in the dropdown will be rendered using virtualization (i.e. only the items that are within the viewport will be rendered). This drastically improves performance for selects with a large number of options. Set to an integer to only use virtualization if the select has at least that number of options. Type: boolean | integer; Default: 600.

width	When set to auto, the width of the selectpicker is automatically adjusted to accommodate the widest option. When set to a css-width, the width of the selectpicker is forced inline to the given value. When set to false, all width information is removed. Type: 'auto' | 'fit' | css-width | false (where css-width is a CSS width with units, e.g. 100px); Default: false.
windowPadding	This is useful in cases where the window has areas that the dropdown menu should not cover - for instance a fixed header. When set to an integer, the same padding will be added to all sides. Alternatively, an array of integers can be used in the format [top, right, bottom, left]. Type: integer | array; Default: 0.

Note

Documentation is from Bootstrap-select page.

Examples

```
if (interactive()) {
  library(shiny)
  library(shinyWidgets)

  ui <- fluidPage(
    pickerInput(
      inputId = "month",
      label = "Select a month",
      choices = month.name,
      multiple = TRUE,
      options = pickerOptions(
        actionsBox = TRUE,
        title = "Please select a month",
        header = "This is a title"
      )
    )
  )

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

  }

  shinyApp(ui, server)
}
```

---

prettyCheckbox *Pretty Checkbox Input*

Description

Create a pretty checkbox that can be used to specify logical values.

Usage

```
prettyCheckbox(inputId, label, value = FALSE, status = "default",
  shape = c("square", "curve", "round"), outline = FALSE,
  fill = FALSE, thick = FALSE, animation = NULL, icon = NULL,
  plain = FALSE, bigger = FALSE, inline = FALSE, width = NULL)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control.
value	Initial value (TRUE or FALSE).
status	Add a class to the checkbox, you can use Bootstrap status like 'info', 'primary', 'danger', 'warning' or 'success'.
shape	Shape of the checkbox between square, curve and round.
outline	Color also the border of the checkbox (TRUE or FALSE).
fill	Fill the checkbox with color (TRUE or FALSE).
thick	Make the content inside checkbox smaller (TRUE or FALSE).
animation	Add an animation when checkbox is checked, a value between smooth, jelly, tada, rotate, pulse.
icon	Optional, display an icon on the checkbox, must be an icon created with icon.
plain	Remove the border when checkbox is checked (TRUE or FALSE).
bigger	Scale the checkboxes a bit bigger (TRUE or FALSE).
inline	Display the input inline, if you want to place checkboxes next to each other.
width	The width of the input, e.g. 400px, or 100%.

Value

TRUE or FALSE server-side.

Note

Due to the nature of different checkbox design, certain animations are not applicable in some arguments combinations. You can find examples on the pretty-checkbox official page : https://lokesh-coder.github.io/pretty-checkbox/.

See Also

See updatePrettyCheckbox to update the value server-side. See prettySwitch and prettyToggle for similar widgets.

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h1("Pretty checkbox"),
  br(),

  fluidRow(
    column(
      width = 4,
      prettyCheckbox(inputId = "checkbox1",
                     label = "Click me!"),
      verbatimTextOutput(outputId = "res1"),
      br(),
      prettyCheckbox(inputId = "checkbox4",  label = "Click me!",
                     outline = TRUE,
                    plain = TRUE, icon = icon("thumbs-up")),
      verbatimTextOutput(outputId = "res4")
    ),
    column(
      width = 4,
      prettyCheckbox(inputId = "checkbox2",
                     label = "Click me!", thick = TRUE,
                     animation = "pulse", status = "info"),
      verbatimTextOutput(outputId = "res2"),
      br(),
      prettyCheckbox(inputId = "checkbox5",
                     label = "Click me!", icon = icon("check"),
                     animation = "tada", status = "default"),
      verbatimTextOutput(outputId = "res5")
    ),
    column(
      width = 4,
      prettyCheckbox(inputId = "checkbox3",  label = "Click me!",
                     shape = "round", status = "danger",
                    fill = TRUE, value = TRUE),
      verbatimTextOutput(outputId = "res3")
    )
  )

)

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

  output$res1 <- renderPrint(input$checkbox1)
  output$res2 <- renderPrint(input$checkbox2)
  output$res3 <- renderPrint(input$checkbox3)
```

```
  output$res4 <- renderPrint(input$checkbox4)
  output$res5 <- renderPrint(input$checkbox5)

}

shinyApp(ui, server)



# Inline example ----

ui <- fluidPage(
  tags$h1("Pretty checkbox: inline example"),
  br(),
  prettyCheckbox(inputId = "checkbox1",
                 label = "Click me!",
                 status = "success",
                 outline = TRUE,
                 inline = TRUE),
  prettyCheckbox(inputId = "checkbox2",
                 label = "Click me!",
                 thick = TRUE,
                 shape = "curve",
                 animation = "pulse",
                 status = "info",
                 inline = TRUE),
  prettyCheckbox(inputId = "checkbox3",
                 label = "Click me!",
                 shape = "round",
                 status = "danger",
                 value = TRUE,
                 inline = TRUE),
  prettyCheckbox(inputId = "checkbox4",
                 label = "Click me!",
                 outline = TRUE,
                 plain = TRUE,
                 animation = "rotate",
                 icon = icon("thumbs-up"),
                 inline = TRUE),
  prettyCheckbox(inputId = "checkbox5",
                 label = "Click me!",
                 icon = icon("check"),
                 animation = "tada",
                 status = "primary",
                 inline = TRUE),
  verbatimTextOutput(outputId = "res")
)

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

  output$res <- renderPrint(c(input$checkbox1,
                              input$checkbox2,
                              input$checkbox3,
```

```
                              input$checkbox4,
                              input$checkbox5))

}

shinyApp(ui, server)

}


## End(Not run)
```

---

prettyCheckboxGroup *Pretty Checkbox Group Input Control*

Description

Create a group of pretty checkboxes that can be used to toggle multiple choices independently. The server will receive the input as a character vector of the selected values.

Usage

```
prettyCheckboxGroup(inputId, label, choices = NULL, selected = NULL,
  status = "default", shape = c("square", "curve", "round"),
  outline = FALSE, fill = FALSE, thick = FALSE, animation = NULL,
  icon = NULL, plain = FALSE, bigger = FALSE, inline = FALSE,
  width = NULL, choiceNames = NULL, choiceValues = NULL)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control.
choices	List of values to show checkboxes for. If elements of the list are named then that name rather than the value is displayed to the user. If this argument is provided, then choiceNames and choiceValues must not be provided, and vice-versa. The values should be strings; other types (such as logicals and numbers) will be coerced to strings.
selected	The values that should be initially selected, if any.
status	Add a class to the checkbox, you can use Bootstrap status like 'info', 'primary', 'danger', 'warning' or 'success'.
shape	Shape of the checkbox between square, curve and round.
outline	Color also the border of the checkbox (TRUE or FALSE).
fill	Fill the checkbox with color (TRUE or FALSE).
thick	Make the content inside checkbox smaller (TRUE or FALSE).
animation	Add an animation when checkbox is checked, a value between smooth, jelly, tada, rotate, pulse.

icon	Optional, display an icon on the checkbox, must be an icon created with icon.
plain	Remove the border when checkbox is checked (TRUE or FALSE).
bigger	Scale the checkboxes a bit bigger (TRUE or FALSE).
inline	If TRUE, render the choices inline (i.e. horizontally).
width	The width of the input, e.g. 400px, or 100%.
choiceNames	List of names to display to the user.
choiceValues	List of values corresponding to choiceNames

Value

A character vector or NULL server-side.

See Also

updatePrettyCheckboxGroup for updating values server-side.

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h1("Pretty checkbox group"),
  br(),

  fluidRow(
    column(
      width = 4,
      prettyCheckboxGroup(inputId = "checkgroup1",
                          label = "Click me!",
                          choices = c("Click me !", "Me !", "Or me !")),
      verbatimTextOutput(outputId = "res1"),
      br(),
      prettyCheckboxGroup(inputId = "checkgroup4",  label = "Click me!",
                          choices = c("Click me !", "Me !", "Or me !"),
                          outline = TRUE,
                          plain = TRUE, icon = icon("thumbs-up")),
      verbatimTextOutput(outputId = "res4")
    ),
    column(
      width = 4,
      prettyCheckboxGroup(inputId = "checkgroup2",
                          label = "Click me!", thick = TRUE,
                          choices = c("Click me !", "Me !", "Or me !"),
                          animation = "pulse", status = "info"),
      verbatimTextOutput(outputId = "res2"),
```

```
      br(),
      prettyCheckboxGroup(inputId = "checkgroup5",
                          label = "Click me!", icon = icon("check"),
                          choices = c("Click me !", "Me !", "Or me !"),
                          animation = "tada", status = "default"),
      verbatimTextOutput(outputId = "res5")
    ),
    column(
      width = 4,
      prettyCheckboxGroup(inputId = "checkgroup3",  label = "Click me!",
                          choices = c("Click me !", "Me !", "Or me !"),
                          shape = "round", status = "danger",
                          fill = TRUE, inline = TRUE),
      verbatimTextOutput(outputId = "res3")
    )
  )

)

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

  output$res1 <- renderPrint(input$checkgroup1)
  output$res2 <- renderPrint(input$checkgroup2)
  output$res3 <- renderPrint(input$checkgroup3)
  output$res4 <- renderPrint(input$checkgroup4)
  output$res5 <- renderPrint(input$checkgroup5)

}

shinyApp(ui, server)

}


## End(Not run)
```

`prettyRadioButtons` *Pretty radio Buttons Input Control*

Description

Create a set of radio buttons used to select an item from a list.

Usage

```
prettyRadioButtons(inputId, label, choices = NULL, selected = NULL,
  status = "primary", shape = c("round", "square", "curve"),
  outline = FALSE, fill = FALSE, thick = FALSE, animation = NULL,
  icon = NULL, plain = FALSE, bigger = FALSE, inline = FALSE,
  width = NULL, choiceNames = NULL, choiceValues = NULL)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control.
choices	List of values to show radio buttons for. If elements of the list are named then that name rather than the value is displayed to the user. If this argument is provided, then choiceNames and choiceValues must not be provided, and vice-versa. The values should be strings; other types (such as logicals and numbers) will be coerced to strings.
selected	The values that should be initially selected, (if not specified then defaults to the first value).
status	Add a class to the radio, you can use Bootstrap status like 'info', 'primary', 'danger', 'warning' or 'success'.
shape	Shape of the radio between square, curve and round.
outline	Color also the border of the radio (TRUE or FALSE).
fill	Fill the radio with color (TRUE or FALSE).
thick	Make the content inside radio smaller (TRUE or FALSE).
animation	Add an animation when radio is checked, a value between smooth, jelly, tada, rotate, pulse.
icon	Optional, display an icon on the radio, must be an icon created with icon.
plain	Remove the border when radio is checked (TRUE or FALSE).
bigger	Scale the radio a bit bigger (TRUE or FALSE).
inline	If TRUE, render the choices inline (i.e. horizontally).
width	The width of the input, e.g. 400px, or 100%.
choiceNames	List of names to display to the user.
choiceValues	List of values corresponding to choiceNames

Value

A character vector or NULL server-side.

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h1("Pretty radio buttons"),
  br(),

  fluidRow(
```

```
    column(
      width = 4,
      prettyRadioButtons(inputId = "radio1",
                         label = "Click me!",
                         choices = c("Click me !", "Me !", "Or me !")),
      verbatimTextOutput(outputId = "res1"),
      br(),
      prettyRadioButtons(inputId = "radio4",  label = "Click me!",
                         choices = c("Click me !", "Me !", "Or me !"),
                         outline = TRUE,
                         plain = TRUE, icon = icon("thumbs-up")),
      verbatimTextOutput(outputId = "res4")
    ),
    column(
      width = 4,
      prettyRadioButtons(inputId = "radio2",
                         label = "Click me!", thick = TRUE,
                         choices = c("Click me !", "Me !", "Or me !"),
                         animation = "pulse", status = "info"),
      verbatimTextOutput(outputId = "res2"),
      br(),
      prettyRadioButtons(inputId = "radio5",
                         label = "Click me!", icon = icon("check"),
                         choices = c("Click me !", "Me !", "Or me !"),
                         animation = "tada", status = "default"),
      verbatimTextOutput(outputId = "res5")
    ),
    column(
      width = 4,
      prettyRadioButtons(inputId = "radio3",  label = "Click me!",
                         choices = c("Click me !", "Me !", "Or me !"),
                         shape = "round", status = "danger",
                         fill = TRUE, inline = TRUE),
      verbatimTextOutput(outputId = "res3")
    )
  )

)

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

  output$res1 <- renderPrint(input$radio1)
  output$res2 <- renderPrint(input$radio2)
  output$res3 <- renderPrint(input$radio3)
  output$res4 <- renderPrint(input$radio4)
  output$res5 <- renderPrint(input$radio5)

}

shinyApp(ui, server)

}
```

```
## End(Not run)
```

`prettySwitch` *Pretty Switch Input*

Description

A toggle switch to replace checkbox

Usage

```
prettySwitch(inputId, label, value = FALSE, status = "default",
  slim = FALSE, fill = FALSE, bigger = FALSE, inline = FALSE,
  width = NULL)
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`label`	Display label for the control, or `NULL` for no label.
`value`	Initial value (`TRUE` or `FALSE`).
`status`	Add a class to the switch, you can use Bootstrap status like 'info', 'primary', 'danger', 'warning' or 'success'.
`slim`	Change the style of the switch (`TRUE` or `FALSE`), see examples.
`fill`	Change the style of the switch (`TRUE` or `FALSE`), see examples.
`bigger`	Scale the switch a bit bigger (`TRUE` or `FALSE`).
`inline`	Display the input inline, if you want to place switch next to each other.
`width`	The width of the input, e.g. `400px`, or `100%`.

Value

`TRUE` or `FALSE` server-side.

Note

Appearance is better in a browser such as Chrome than in RStudio Viewer

See Also

See `updatePrettySwitch` to update the value server-side.

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h1("Pretty switches"),
  br(),

  fluidRow(
    column(
      width = 4,
      prettySwitch(inputId = "switch1", label = "Default:"),
      verbatimTextOutput(outputId = "res1"),
      br(),
      prettySwitch(inputId = "switch4",
                   label = "Fill switch with status:",
                   fill = TRUE, status = "primary"),
      verbatimTextOutput(outputId = "res4")
    ),
    column(
      width = 4,
      prettySwitch(inputId = "switch2",
                   label = "Danger status:",
                   status = "danger"),
      verbatimTextOutput(outputId = "res2")
    ),
    column(
      width = 4,
      prettySwitch(inputId = "switch3",
                   label = "Slim switch:",
                   slim = TRUE),
      verbatimTextOutput(outputId = "res3")
    )
  )

)

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

  output$res1 <- renderPrint(input$switch1)
  output$res2 <- renderPrint(input$switch2)
  output$res3 <- renderPrint(input$switch3)
  output$res4 <- renderPrint(input$switch4)

}

shinyApp(ui, server)
```

```
}


## End(Not run)
```

---

prettyToggle *Pretty Toggle Input*

Description

A single checkbox that changes appearance if checked or not.

Usage

```
prettyToggle(inputId, label_on, label_off, icon_on = NULL,
  icon_off = NULL, value = FALSE, status_on = "success",
  status_off = "danger", shape = c("square", "curve", "round"),
  outline = FALSE, fill = FALSE, thick = FALSE, plain = FALSE,
  bigger = FALSE, animation = NULL, inline = FALSE, width = NULL)
```

Arguments

inputId	The input slot that will be used to access the value.
label_on	Display label for the control when value is TRUE.
label_off	Display label for the control when value is FALSE
icon_on	Optional, display an icon on the checkbox when value is TRUE, must be an icon created with icon.
icon_off	Optional, display an icon on the checkbox when value is FALSE, must be an icon created with icon.
value	Initial value (TRUE or FALSE).
status_on	Add a class to the checkbox when value is TRUE, you can use Bootstrap status like 'info', 'primary', 'danger', 'warning' or 'success'.
status_off	Add a class to the checkbox when value is FALSE, you can use Bootstrap status like 'info', 'primary', 'danger', 'warning' or 'success'.
shape	Shape of the checkbox between square, curve and round.
outline	Color also the border of the checkbox (TRUE or FALSE).
fill	Fill the checkbox with color (TRUE or FALSE).
thick	Make the content inside checkbox smaller (TRUE or FALSE).
plain	Remove the border when checkbox is checked (TRUE or FALSE).
bigger	Scale the checkboxes a bit bigger (TRUE or FALSE).
animation	Add an animation when checkbox is checked, a value between smooth, jelly, tada, rotate, pulse.
inline	Display the input inline, if you want to place checkboxes next to each other.
width	The width of the input, e.g. 400px, or 100%.

Value

TRUE or FALSE server-side.

See Also

See updatePrettyToggle to update the value server-side.

Examples

```
## Not run:

if (interactive()) {
library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h1("Pretty toggles"),
  br(),

  fluidRow(
    column(
      width = 4,
      prettyToggle(inputId = "toggle1",
                   label_on = "Checked!",
                   label_off = "Unchecked..."),
      verbatimTextOutput(outputId = "res1"),
      br(),
      prettyToggle(inputId = "toggle4",  label_on = "Yes!",
                   label_off = "No..", outline = TRUE,
                   plain = TRUE,
                   icon_on = icon("thumbs-up"),
                   icon_off = icon("thumbs-down")),
      verbatimTextOutput(outputId = "res4")
    ),
    column(
      width = 4,
      prettyToggle(inputId = "toggle2",
                   label_on = "Yes!", icon_on = icon("check"),
                   status_on = "info", status_off = "warning",
                   label_off = "No..", icon_off = icon("remove")),
      verbatimTextOutput(outputId = "res2")
    ),
    column(
      width = 4,
      prettyToggle(inputId = "toggle3",  label_on = "Yes!",
                   label_off = "No..", shape = "round",
                   fill = TRUE, value = TRUE),
      verbatimTextOutput(outputId = "res3")
    )
  )

)
```

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

  output$res1 <- renderPrint(input$toggle1)
  output$res2 <- renderPrint(input$toggle2)
  output$res3 <- renderPrint(input$toggle3)
  output$res4 <- renderPrint(input$toggle4)

}

shinyApp(ui, server)



# Inline example ----


ui <- fluidPage(
  tags$h1("Pretty toggles: inline example"),
  br(),

  prettyToggle(inputId = "toggle1",
               label_on = "Checked!",
               label_off = "Unchecked...",
               inline = TRUE),
  prettyToggle(inputId = "toggle2",
               label_on = "Yep",
               status_on = "default",
               icon_on = icon("ok-circle", lib = "glyphicon"),
               label_off = "Nope",
               status_off = "default",
               icon_off = icon("remove-circle", lib = "glyphicon"),
               plain = TRUE,
               inline = TRUE),
  prettyToggle(inputId = "toggle3",
               label_on = "",
               label_off = "",
               icon_on = icon("volume-up", lib = "glyphicon"),
               icon_off = icon("volume-off", lib = "glyphicon"),
               status_on = "primary",
               status_off = "default",
               plain = TRUE,
               outline = TRUE,
               bigger = TRUE,
               inline = TRUE),
  prettyToggle(inputId = "toggle4",
               label_on = "Yes!",
               label_off = "No..",
               outline = TRUE,
               plain = TRUE,
               icon_on = icon("thumbs-up"),
               icon_off = icon("thumbs-down"),
               inline = TRUE),
```

```
  verbatimTextOutput(outputId = "res")

)

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

  output$res <- renderPrint(c(input$toggle1,
                              input$toggle2,
                              input$toggle3,
                              input$toggle4))

}

shinyApp(ui, server)



}


## End(Not run)
```

---

progress-bar *Progress Bars*

Description

Create a progress bar to provide feedback on calculation.

Usage

```
progressBar(id, value, total = NULL, display_pct = FALSE,
  size = NULL, status = NULL, striped = FALSE, title = NULL,
  range_value = NULL, unit_mark = "%")

updateProgressBar(session, id, value, total = NULL, title = NULL,
  status = NULL, range_value = NULL, unit_mark = "%")
```

Arguments

id	An id used to update the progress bar.
value	Value of the progress bar between 0 and 100, if >100 you must provide total.
total	Used to calculate percentage if value > 100, force an indicator to appear on top right of the progress bar.
display_pct	logical, display percentage on the progress bar.
size	Size, 'NULL' by default or a value in 'xxs', 'xs', 'sm', only work with package 'shinydashboard'.

status	Color, must be a valid Bootstrap status : primary, info, success, warning, danger.
striped	logical, add a striped effect.
title	character, optional title.
range_value	Default is to display percentage (`[0, 100]`), but you can specify a custom range, e.g. `-50, 50`.
unit_mark	Unit for value displayed on the progress bar, default to `"%"`.
session	The 'session' object passed to function given to shinyServer.

Value

A progress bar that can be added to a UI definition.

See Also

progressSweetAlert for progress bar in a sweet alert

Examples

```
## Not run:
if (interactive()) {

library("shiny")
library("shinyWidgets")

ui <- fluidPage(
  column(
    width = 7,
    tags$b("Default"), br(),
    progressBar(id = "pb1", value = 50),
    sliderInput(
      inputId = "up1",
      label = "Update",
      min = 0,
      max = 100,
      value = 50
    ),
    br(),
    tags$b("Other options"), br(),
    progressBar(
      id = "pb2",
      value = 0,
      total = 100,
      title = "",
      display_pct = TRUE
    ),
    actionButton(
      inputId = "go",
      label = "Launch calculation"
    )
  )
```

```
)

server <- function(input, output, session) {
  observeEvent(input$up1, {
    updateProgressBar(
      session = session,
      id = "pb1",
      value = input$up1
    )
  })
  observeEvent(input$go, {
    for (i in 1:100) {
      updateProgressBar(
        session = session,
        id = "pb2",
        value = i, total = 100,
        title = paste("Process", trunc(i/10))
      )
      Sys.sleep(0.1)
    }
  })
}

shinyApp(ui = ui, server = server)

}

## End(Not run)
```

`progressSweetAlert` *Progress bar in a sweet alert*

Description

Progress bar in a sweet alert

Usage

```
progressSweetAlert(session, id, value, total = NULL,
  display_pct = FALSE, size = NULL, status = NULL, striped = FALSE,
  title = NULL)
```

Arguments

`session`	The `session` object passed to function given to shinyServer.
`id`	An id used to update the progress bar.
`value`	Value of the progress bar between 0 and 100, if >100 you must provide total.
`total`	Used to calculate percentage if value > 100, force an indicator to appear on top right of the progress bar.

display_pct	logical, display percentage on the progress bar.
size	Size, 'NULL' by default or a value in 'xxs', 'xs', 'sm', only work with package 'shinydashboard'.
status	Color, must be a valid Bootstrap status : primary, info, success, warning, danger.
striped	logical, add a striped effect.
title	character, optional title.

Examples

```
## Not run:

if (interactive()) {

library("shiny")
library("shinyWidgets")


ui <- fluidPage(
  tags$h1("Progress bar in Sweet Alert"),
  useSweetAlert(), # /!\ needed with 'progressSweetAlert'
  actionButton(
    inputId = "go",
    label = "Launch long calculation !"
  )
)

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

  observeEvent(input$go, {
    progressSweetAlert(
      session = session, id = "myprogress",
      title = "Work in progress",
      display_pct = TRUE, value = 0
    )
    for (i in seq_len(50)) {
      Sys.sleep(0.1)
      updateProgressBar(
        session = session,
        id = "myprogress",
         value = i*2
      )
    }
    closeSweetAlert(session = session)
    sendSweetAlert(
      session = session,
      title =" Calculation completed !",
      type = "success"
    )
  })

}
```

```
shinyApp(ui = ui, server = server)

}


## End(Not run)
```

`radioGroupButtons` *Buttons Group Radio Input Control*

Description

Create buttons grouped that act like radio buttons.

Usage

```
radioGroupButtons(inputId, label = NULL, choices = NULL,
  selected = NULL, status = "default", size = "normal",
  direction = "horizontal", justified = FALSE, individual = FALSE,
  checkIcon = list(), width = NULL, choiceNames = NULL,
  choiceValues = NULL)
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`label`	Input label.
`choices`	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).
`selected`	The initially selected value.
`status`	Add a class to the buttons, you can use Bootstrap status like 'info', 'primary', 'danger', 'warning' or 'success'. Or use an arbitrary strings to add a custom class, e.g. : with `status = 'myClass'`, buttons will have class `btn-myClass`.
`size`	Size of the buttons ('xs', 'sm', 'normal', 'lg')
`direction`	Horizontal or vertical
`justified`	If TRUE, fill the width of the parent div
`individual`	If TRUE, buttons are separated.
`checkIcon`	A list, if no empty must contain at least one element named 'yes' corresponding to an icon to display if the button is checked.
`width`	The width of the input, e.g. '400px', or '100%'.
`choiceNames, choiceValues`	Same as in `radioButtons`. List of names and values, respectively, that are displayed to the user in the app and correspond to the each choice (for this reason, `choiceNames` and `choiceValues` must have the same length).

Value

A buttons group control that can be added to a UI definition.

Examples

```
## Not run:
## Only run examples in interactive R sessions
if (interactive()) {

ui <- fluidPage(
  radioGroupButtons(inputId = "somevalue", choices = c("A", "B", "C")),
  verbatimTextOutput("value")
)
server <- function(input, output) {
  output$value <- renderText({ input$somevalue })
}
shinyApp(ui, server)
}

## End(Not run)
```

---

searchInput *Search Input*

Description

A text input only triggered when Enter key is pressed or search button clicked

Usage

```
searchInput(inputId, label = NULL, value = "", placeholder = NULL,
  btnSearch = NULL, btnReset = NULL, resetValue = "", width = NULL)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
value	Initial value.
placeholder	A character string giving the user a hint as to what can be entered into the control.
btnSearch	An icon for the button which validate the search.
btnReset	An icon for the button which reset the search.
resetValue	Value used when reset button is clicked, default to "", if NULL value is not reset.
width	The width of the input, e.g. '400px', or '100%'.

Note

The two buttons ('search' and 'reset') act like `actionButton`, you can retrieve their value server-side with `input$<INPUTID>_search` and `input$<INPUTID>_reset`.

See Also

updateSearchInput to update value server-side.

Examples

```
## Not run:
if (interactive()) {
  ui <- fluidPage(
    tags$h1("Search Input"),
    br(),
    searchInput(
      inputId = "search", label = "Enter your text",
      placeholder = "A placeholder",
      btnSearch = icon("search"),
      btnReset = icon("remove"),
      width = "450px"
    ),
    br(),
    verbatimTextOutput(outputId = "res")
  )

  server <- function(input, output, session) {
    output$res <- renderPrint({
      input$search
    })
  }

  shinyApp(ui = ui, server = server)
}

## End(Not run)
```

---

selectizeGroup-module *Selectize Group*

Description

Group of mutually dependent 'selectizeInput' for filtering data.frame's columns (like in Excel).

Usage

```
selectizeGroupUI(id, params, label = NULL, btn_label = "Reset filters",
  inline = TRUE)

selectizeGroupServer(input, output, session, data, vars)
```

Arguments

id	Module's id.
params	A named list of parameters passed to each 'selectizeInput', you can use : 'inputId' (obligatory, must be variable name), 'label', 'placeholder'.
label	Character, global label on top of all labels.
btn_label	Character, reset button label.
inline	If TRUE (the default), 'selectizeInput's are horizontally positioned, otherwise vertically.
input	standard shiny input.
output	standard shiny output.
session	standard shiny session.
data	a data.frame, or an object that can be coerced to data.frame.
vars	character, columns to use to create filters, must correspond to variables listed in params.

Value

a reactive function containing data filtered.

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)

data("mpg", package = "ggplot2")

ui <- fluidPage(
  fluidRow(
    column(
      width = 10, offset = 1,
      tags$h3("Filter data with selectize group"),
      panel(
        selectizeGroupUI(
          id = "my-filters",
          params = list(
            manufacturer = list(inputId = "manufacturer", title = "Manufacturer:"),
```

```
            model = list(inputId = "model", title = "Model:"),
            trans = list(inputId = "trans", title = "Trans:"),
            class = list(inputId = "class", title = "Class:")
          )
        ), status = "primary"
      ),
      dataTableOutput(outputId = "table")
    )
  )
)

server <- function(input, output, session) {
  res_mod <- callModule(
    module = selectizeGroupServer,
    id = "my-filters",
    data = mpg,
    vars = c("manufacturer", "model", "trans", "class")
  )
  output$table <- renderDataTable(res_mod())
}

shinyApp(ui, server)

}


## End(Not run)
```

`sendSweetAlert` *Display a Sweet Alert to the user*

Description

Send a message from the server and launch a sweet alert in the UI.

Usage

```
sendSweetAlert(session, title = "Title", text = NULL, type = NULL,
  btn_labels = "Ok", html = FALSE, closeOnClickOutside = TRUE)
```

Arguments

`session`	The `session` object passed to function given to shinyServer.
`title`	Title of the alert.
`text`	Text of the alert.
`type`	Type of the alert : info, success, warning or error.
`btn_labels`	Label(s) for button(s), can be of length 2, in which case the alert will have two buttons.

`html` Does `text` contains HTML tags ?

`closeOnClickOutside`
Decide whether the user should be able to dismiss the modal by clicking outside of it, or not.

See Also

`confirmSweetAlert`, `inputSweetAlert`

Examples

```
## Not run:
if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h2("Sweet Alert examples"),
  actionButton(
    inputId = "success",
    label = "Launch a success sweet alert",
    icon = icon("check")
  ),
  actionButton(
    inputId = "error",
    label = "Launch an error sweet alert",
    icon = icon("remove")
  ),
  actionButton(
    inputId = "sw_html",
    label = "Sweet alert with HTML",
    icon = icon("thumbs-up")
  )
)

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

  observeEvent(input$success, {
    sendSweetAlert(
      session = session,
      title = "Success !!",
      text = "All in order",
      type = "success"
    )
  })

  observeEvent(input$error, {
    sendSweetAlert(
      session = session,
      title = "Error !!",
      text = "It's broken...",
```

```
      type = "error"
    )
  })

  observeEvent(input$sw_html, {
    sendSweetAlert(
      session = session,
      title = NULL,
      text = tags$span(
        tags$h3("With HTML tags",
                style = "color: steelblue;"),
        "In", tags$b("bold"), "and", tags$em("italic"),
        tags$br(),
        "and",
        tags$br(),
        "line",
        tags$br(),
        "breaks",
        tags$br(),
        "and an icon", icon("thumbs-up")
      ),
      html = TRUE
    )
  })

}

shinyApp(ui, server)


# output in Sweet Alert #

library("shiny")
library("shinyWidgets")

shinyApp(
  ui = fluidPage(
    tags$h1("Click the button"),
    actionButton(
      inputId = "sw_html",
      label = "Sweet alert with plot"
    ),
    # SweetAlert width
    tags$style(".swal-modal {width: 80%;}")
  ),
  server = function(input, output, session) {
    observeEvent(input$sw_html, {
      sendSweetAlert(
        session = session,
        title = "Yay a plot!",
        text = tags$div(
          plotOutput(outputId = "plot"),
          sliderInput(
```

```
            inputId = "clusters",
            label = "Number of clusters",
            min = 2, max = 6, value = 3, width = "100%"
          )
        ),
        html = TRUE
      )
    })
    output$plot <- renderPlot({
      plot(Sepal.Width ~ Sepal.Length,
           data = iris, col = Species,
           pch = 20, cex = 2)
      points(kmeans(iris[, 1:2], input$clusters)$centers,
             pch = 4, cex = 4, lwd = 4)
    })
  }
)

}

## End(Not run)
```

`setBackgroundColor` *Custom background color for your shinyapp*

Description

Allow to change the background color of your shinyapp.

Usage

```
setBackgroundColor(color = "ghostwhite", gradient = c("linear",
  "radial"), direction = c("bottom", "top", "right", "left"))
```

Arguments

`color`	Background color. Use either the fullname or the Hex code (https://www.w3schools.com/colors/colors_hex.asp). If more than one color is used, a gradient background is set.
`gradient`	Type of gradient: `linear` or `radial`.
`direction`	Direction for gradient, by default to `bottom`. Possibles choices are `bottom`, `top`, `right` or `left`, two values can be used, e.g. `c("bottom", "right")`.

Examples

```
## Not run:

if (interactive()) {
```

```
### Uniform color background :

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h2("Change shiny app background"),
  setBackgroundColor("ghostwhite")
)

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

}

shinyApp(ui, server)


### linear gradient background :

library(shiny)
library(shinyWidgets)

ui <- fluidPage(

  # use a gradient in background
  setBackgroundColor(
    color = c("#F7FBFF", "#2171B5"),
    gradient = "linear",
    direction = "bottom"
  ),

  titlePanel("Hello Shiny!"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("obs",
                  "Number of observations:",
                  min = 0,
                  max = 1000,
                  value = 500)
    ),
    mainPanel(
      plotOutput("distPlot")
    )
  )
)

server <- function(input, output, session) {
  output$distPlot <- renderPlot({
    hist(rnorm(input$obs))
  })
}

shinyApp(ui, server)
```

```
### radial gradient background :

library(shiny)
library(shinyWidgets)

ui <- fluidPage(

  # use a gradient in background
  setBackgroundColor(
    color = c("#F7FBFF", "#2171B5"),
    gradient = "radial",
    direction = c("top", "left")
  ),

  titlePanel("Hello Shiny!"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("obs",
                  "Number of observations:",
                  min = 0,
                  max = 1000,
                  value = 500)
    ),
    mainPanel(
      plotOutput("distPlot")
    )
  )
)

server <- function(input, output, session) {
  output$distPlot <- renderPlot({
    hist(rnorm(input$obs))
  })
}

shinyApp(ui, server)

}


## End(Not run)
```

---

setBackgroundImage *Custom background image for your shinyapp*

Description

Allow to change the background image of your shinyapp.

Usage

```
setBackgroundImage(src = NULL)
```

Arguments

src	Url or path to the image, if using local image, the file must be in www/ directory and the path not contain www/.

Examples

```
if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h2("Add a shiny app background image"),
  setBackgroundImage(src = "http://wallpics4k.com/wp-content/uploads/2014/07/470318.jpg")
)

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

}

shinyApp(ui, server)

}
```

---

setShadow *Custom shadows*

Description

Allow to apply a shadow on a given element.

Usage

```
setShadow(id = NULL, class = NULL)
```

Arguments

id	Use this argument if you want to target an individual element.
class	The element to which the shadow should be applied. For example, class is set to box.

Examples

```
if (interactive()) {

 library(shiny)
 library(shinydashboard)
 library(shinydashboardPlus)
 library(shinyWidgets)

 boxTag <- boxPlus(
  title = "Closable box, with label",
  closable = TRUE,
  enable_label = TRUE,
  label_text = 1,
  label_status = "danger",
  status = "warning",
  solidHeader = FALSE,
  collapsible = TRUE,
  p("Box Content")
 )

 shinyApp(
  ui = dashboardPagePlus(
    header = dashboardHeaderPlus(
      enable_rightsidebar = TRUE,
      rightSidebarIcon = "gears"
    ),
    sidebar = dashboardSidebar(),
    body = dashboardBody(

     setShadow(class = "box"),
     setShadow(id = "my-progress"),

     tags$h2("Add shadow to the box class"),
     fluidRow(boxTag, boxTag),
     tags$h2("Add shadow only to the first element using id"),
     tagAppendAttributes(
      verticalProgress(
       value = 10,
       striped = TRUE,
       active = TRUE
      ),
      id = "my-progress"
     ),
     verticalProgress(
       value = 50,
       active = TRUE,
       status = "warning",
       size = "xs"
     ),
     verticalProgress(
       value = 20,
       status = "danger",
```

```
        size = "sm",
        height = "60%"
      )
     ),
     rightsidebar = rightSidebar(),
     title = "DashboardPage"
   ),
   server = function(input, output) { }
  )
}
```

`setSliderColor` *Color editor for sliderInput*

Description

Edit the color of the original shiny's sliderInputs

Usage

```
setSliderColor(color, sliderId)
```

Arguments

`color`	The `color` to apply. This can also be a vector of colors if you want to customize more than 1 slider. Either pass the name of the color such as 'Chartreuse ' and 'Chocolate 'or the HEX notation such as `'#7FFF00'` and `'#D2691E'`.
`sliderId`	The `id` of the customized slider(s). This can be a vector like `c(1, 2)`, if you want to modify the 2 first sliders. However, if you only want to modify the second slider, just use the value 2.

Note

See also https://www.w3schools.com/colors/colors_names.asp to have an overview of all colors.

See Also

See `chooseSliderSkin` to update the global skin of your sliders.

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)
```

```
ui <- fluidPage(

  # only customize the 2 first sliders and the last one
  # the color of the third one is empty
  setSliderColor(c("DeepPink ", "#FF4500", "", "Teal"), c(1, 2, 4)),
  sliderInput("obs", "My pink slider:",
              min = 0, max = 100, value = 50
  ),
  sliderInput("obs2", "My orange slider:",
              min = 0, max = 100, value = 50
  ),
  sliderInput("obs3", "My basic slider:",
              min = 0, max = 100, value = 50
  ),
  sliderInput("obs3", "My teal slider:",
              min = 0, max = 100, value = 50
  ),
  plotOutput("distPlot")
)

server <- function(input, output) {

  output$distPlot <- renderPlot({
    hist(rnorm(input$obs))
  })
}

shinyApp(ui, server)

}



## End(Not run)
```

`shinyWidgets` *shinyWidgets: Custom inputs widgets for Shiny.*

Description

The shinyWidgets package provides several custom widgets to extend those available in package shiny

Examples

```
## Not run:
if (interactive()) {
  shinyWidgets::shinyWidgetsGallery()
}
```

```
## End(Not run)
```

---

shinyWidgetsGallery *Launch the shinyWidget Gallery*

Description

A gallery of widgets available in the package.

Usage

```
shinyWidgetsGallery()
```

Examples

```
## Not run:

if (interactive()) {

 shinyWidgetsGallery()

}


## End(Not run)
```

---

sliderTextInput *Slider Text Input Widget*

Description

Constructs a slider widget with characters instead of numeric values.

Usage

```
sliderTextInput(inputId, label, choices, selected = NULL,
  animate = FALSE, grid = FALSE, hide_min_max = FALSE,
  from_fixed = FALSE, to_fixed = FALSE, from_min = NULL,
  from_max = NULL, to_min = NULL, to_max = NULL,
  force_edges = FALSE, width = NULL, pre = NULL, post = NULL,
  dragRange = TRUE)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
choices	Character vector to select a value from.
selected	The initially selected value, if length > 1, create a range slider.
animate	TRUE to show simple animation controls with default settings, for more details see sliderInput.
grid	Logical, show or hide ticks marks.
hide_min_max	Hides min and max labels.
from_fixed	Fix position of left (or single) handle.
to_fixed	Fix position of right handle.
from_min	Set minimum limit for left handle.
from_max	Set the maximum limit for left handle.
to_min	Set minimum limit for right handle.
to_max	Set the maximum limit for right handle.
force_edges	Slider will be always inside it's container.
width	The width of the input, e.g. 400px, or 100%.
pre	A prefix string to put in front of the value.
post	A suffix string to put after the value.
dragRange	See the same argument in sliderInput.

Value

The value retrieved server-side is a character vector.

See Also

updateSliderTextInput to update value server-side.

Examples

```
## Not run:

if (interactive()) {

library("shiny")
library("shinyWidgets")

ui <- fluidPage(
  br(),
  sliderTextInput(
    inputId = "mySliderText",
    label = "Month range slider:",
    choices = month.name,
```

```
    selected = month.name[c(4, 7)]
  ),
  verbatimTextOutput(outputId = "result")
)

server <- function(input, output, session) {
  output$result <- renderPrint(str(input$mySliderText))
}

shinyApp(ui = ui, server = server)

}


## End(Not run)
```

`spectrumInput` *Palette Color Picker with Spectrum Library*

Description

A widget to select a color within palettes, and with more options if needed.

Usage

```
spectrumInput(inputId, label, choices = NULL, selected = NULL,
  flat = FALSE, options = list(), update_on = c("move", "dragstop",
  "change"), width = NULL)
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`label`	Display label for the control, or `NULL` for no label.
`choices`	List of colors to display in the menu.
`selected`	The initially selected value.
`flat`	Display the menu inline.
`options`	Additional options to pass to spectrum, possible values are described here : https://bgrins.github.io/spectrum/#options.
`update_on`	When to update value server-side: `"move"` (default, each time a new color is selected), `"dragstop"` (when use user stop dragging cursor), `"change"` (when the input is closed).
`width`	The width of the input, e.g. `400px`, or `100%`.

Value

The selected color in Hex format server-side

Examples

```
## Not run:

if (interactive()) {

library("shiny")
library("shinyWidgets")
library("RColorBrewer")

ui <- fluidPage(
  tags$h1("Spectrum color picker"),

  br(),

  spectrumInput(
    inputId = "myColor",
    label = "Pick a color:",
    choices = list(
      list('black', 'white', 'blanchedalmond', 'steelblue', 'forestgreen'),
      as.list(brewer.pal(n = 9, name = "Blues")),
      as.list(brewer.pal(n = 9, name = "Greens")),
      as.list(brewer.pal(n = 11, name = "Spectral")),
      as.list(brewer.pal(n = 8, name = "Dark2"))
    ),
    options = list(`toggle-palette-more-text` = "Show more")
  ),
  verbatimTextOutput(outputId = "res")

)

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

  output$res <- renderPrint(input$myColor)

}

shinyApp(ui, server)

}


## End(Not run)
```

`switchInput` *Bootstrap Switch Input Control*

Description

Create a toggle switch.

Usage

```
switchInput(inputId, label = NULL, value = FALSE, onLabel = "ON",
  offLabel = "OFF", onStatus = NULL, offStatus = NULL,
  size = "default", labelWidth = "auto", handleWidth = "auto",
  disabled = FALSE, inline = FALSE, width = NULL)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display a text in the center of the switch.
value	Initial value (TRUE or FALSE).
onLabel	Text on the left side of the switch (TRUE).
offLabel	Text on the right side of the switch (FALSE).
onStatus	Color (bootstrap status) of the left side of the switch (TRUE).
offStatus	Color (bootstrap status) of the right side of the switch (FALSE).
size	Size of the buttons ('default', 'mini', 'small', 'normal', 'large').
labelWidth	Width of the center handle in pixels.
handleWidth	Width of the left and right sides in pixels.
disabled	Logical, display the toggle switch in disabled state?.
inline	Logical, display the toggle switch inline?
width	The width of the input : 'auto', 'fit', '100px', '75%'.

Value

A switch control that can be added to a UI definition.

Note

For more information, see the project on Github https://github.com/Bttstrp/bootstrap-switch.

See Also

updateSwitchInput, materialSwitch

Examples

```
## Not run:
## Only run examples in interactive R sessions
if (interactive()) {

# Examples in the gallery :
shinyWidgets::shinyWidgetsGallery()

# Basic usage :
ui <- fluidPage(
  switchInput(inputId = "somevalue"),
```

```
  verbatimTextOutput("value")
)
server <- function(input, output) {
  output$value <- renderPrint({ input$somevalue })
}
shinyApp(ui, server)
}

## End(Not run)
```

`textInputAddon` *Text with Add-on Input Control*

Description

Create text field with add-on.

Usage

```
textInputAddon(inputId, label, value = "", placeholder = NULL, addon,
  width = NULL)
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`label`	Display label for the control, or `NULL` for no label.
`value`	Initial value..
`placeholder`	A character string giving the user a hint as to what can be entered into the control.
`addon`	An icon tag, created by icon.
`width`	The width of the input : 'auto', 'fit', '100px', '75%'

Value

A switch control that can be added to a UI definition.

Examples

```
## Not run:
## Only run examples in interactive R sessions
if (interactive()) {
shinyApp(
  ui = fluidPage(
   textInputAddon(inputId = "id", label = "Label", placeholder = "Username", addon = icon("at")),
    verbatimTextOutput(outputId = "out")
  ),
  server = function(input, output) {
    output$out <- renderPrint({
```

```
      input$id
    })
  }
)
}

## End(Not run)
```

`toggleDropdownButton` *Toggle a dropdown menu*

Description

Open or close a dropdown menu server-side.

Usage

```
toggleDropdownButton(inputId, session = getDefaultReactiveDomain())
```

Arguments

`inputId`	Id for the dropdown to toggle.
`session`	Standard shiny `session`.

Examples

```
## Not run:

if (interactive()) {

library("shiny")
library("shinyWidgets")

ui <- fluidPage(
  tags$h2("Toggle Dropdown Button"),
  br(),
  fluidRow(
    column(
      width = 6,
      dropdownButton(
        tags$h3("List of Inputs"),
        selectInput(inputId = 'xcol',
                    label = 'X Variable',
                    choices = names(iris)),
        sliderInput(inputId = 'clusters',
                    label = 'Cluster count',
                    value = 3,
                    min = 1,
                    max = 9),
```

```
        actionButton(inputId = "toggle2",
                     label = "Close dropdown"),
        circle = TRUE, status = "danger",
        inputId = "mydropdown",
        icon = icon("gear"), width = "300px"
      )
    ),
    column(
      width = 6,
      actionButton(inputId = "toggle1",
                   label = "Open dropdown")
    )
  )
)

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

  observeEvent(list(input$toggle1, input$toggle2), {
    toggleDropdownButton(inputId = "mydropdown")
  }, ignoreInit = TRUE)

}

shinyApp(ui = ui, server = server)

}



## End(Not run)
```

`tooltipOptions` *Tooltip options*

Description

List of options for tooltip for a dropdown menu button.

Usage

```
tooltipOptions(placement = "right", title = "Params", html = FALSE)
```

Arguments

`placement`	Placement of tooltip : right, top, bottom, left.
`title`	Text of the tooltip
`html`	Logical, allow HTML tags inside tooltip

updateAirDateInput *Change the value of* airDatepickerInput *on the client*

Description

Change the value of airDatepickerInput on the client

Usage

```
updateAirDateInput(session, inputId, label = NULL, value = NULL,
  clear = FALSE, options = NULL)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The id of the input object.
label	The label to set for the input object.
value	The value to set for the input object.
clear	Logical, clear all previous selected dates.
options	Options to update, see available ones here: http://t1m0n.name/air-datepicker/docs/.

Examples

```
## Not run:

demoAirDatepicker("update")


## End(Not run)
```

updateAwesomeCheckbox *Change the value of an awesome checkbox input on the client*

Description

Change the value of an awesome checkbox input on the client

Usage

```
updateAwesomeCheckbox(session, inputId, label = NULL, value = NULL)
```

Arguments

session	standard shiny session
inputId	The id of the input object.
label	The label to set for the input object.
value	The value to set for the input object.

See Also

awesomeCheckbox

Examples

```
## Not run:

if (interactive()) {

library("shiny")
library("shinyWidgets")


ui <- fluidPage(
  awesomeCheckbox(
    inputId = "somevalue",
    label = "My label",
    value = FALSE
  ),

  verbatimTextOutput(outputId = "res"),

  actionButton(inputId = "updatevalue", label = "Toggle value"),
  textInput(inputId = "updatelabel", label = "Update label")
)

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

  output$res <- renderPrint({
    input$somevalue
  })

  observeEvent(input$updatevalue, {
    updateAwesomeCheckbox(
      session = session, inputId = "somevalue",
      value = as.logical(input$updatevalue %%2)
    )
  })

  observeEvent(input$updatelabel, {
    updateAwesomeCheckbox(
      session = session, inputId = "somevalue",
      label = input$updatelabel
    )
```

```
  }, ignoreInit = TRUE)

}

shinyApp(ui = ui, server = server)

}


## End(Not run)
```

---

updateAwesomeCheckboxGroup *Change the value of a* awesomeCheckboxGroup *input on the client*

Description

Change the value of a awesomeCheckboxGroup input on the client

Usage

```
updateAwesomeCheckboxGroup(session, inputId, label = NULL,
  choices = NULL, selected = NULL, inline = FALSE,
  status = "primary")
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The id of the input object.
label	Input label.
choices	List of values to show checkboxes for.
selected	The values that should be initially selected, if any.
inline	If TRUE, render the choices inline (i.e. horizontally)
status	Color of the buttons.

See Also

awesomeCheckboxGroup

Examples

```
## Not run:

if (interactive()) {

library("shiny")
library("shinyWidgets")
```

```
ui <- fluidPage(
  awesomeCheckboxGroup(
    inputId = "somevalue",
    choices = c("A", "B", "C"),
    label = "My label"
  ),

  verbatimTextOutput(outputId = "res"),

  actionButton(inputId = "updatechoices", label = "Random choices"),
  textInput(inputId = "updatelabel", label = "Update label")
)

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

  output$res <- renderPrint({
    input$somevalue
  })

  observeEvent(input$updatechoices, {
    updateAwesomeCheckboxGroup(
      session = session, inputId = "somevalue",
      choices = sample(letters, sample(2:6))
    )
  })

  observeEvent(input$updatelabel, {
    updateAwesomeCheckboxGroup(
      session = session, inputId = "somevalue",
      label = input$updatelabel
    )
  }, ignoreInit = TRUE)

}

shinyApp(ui = ui, server = server)

}


## End(Not run)
```

---

updateAwesomeRadio *Change the value of a radio input on the client*

Description

Change the value of a radio input on the client

Usage

```
updateAwesomeRadio(session, inputId, label = NULL, choices = NULL,
  selected = NULL, inline = FALSE, status = "primary",
  checkbox = FALSE)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The id of the input object.
label	Input label.
choices	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)
selected	The initially selected value
inline	If TRUE, render the choices inline (i.e. horizontally)
status	Color of the buttons
checkbox	Checkbox style

See Also

awesomeRadio

Examples

```
## Not run:

if (interactive()) {

library("shiny")
library("shinyWidgets")


ui <- fluidPage(
  awesomeRadio(
    inputId = "somevalue",
    choices = c("A", "B", "C"),
    label = "My label"
  ),

  verbatimTextOutput(outputId = "res"),

  actionButton(inputId = "updatechoices", label = "Random choices"),
  textInput(inputId = "updatelabel", label = "Update label")
)

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

  output$res <- renderPrint({
    input$somevalue
  })
```

```
  observeEvent(input$updatechoices, {
    updateAwesomeRadio(
      session = session, inputId = "somevalue",
      choices = sample(letters, sample(2:6))
    )
  })

  observeEvent(input$updatelabel, {
    updateAwesomeRadio(
      session = session, inputId = "somevalue",
      label = input$updatelabel
    )
  }, ignoreInit = TRUE)

}

shinyApp(ui = ui, server = server)

}


## End(Not run)
```

`updateCheckboxGroupButtons` *Change the value of a checkboxes group buttons input on the client*

Description

Change the value of a radio group buttons input on the client

Usage

```
updateCheckboxGroupButtons(session, inputId, label = NULL,
  choices = NULL, selected = NULL, status = "default",
  size = "normal", checkIcon = list(), choiceNames = NULL,
  choiceValues = NULL)
```

Arguments

`session`	The session object passed to function given to shinyServer.
`inputId`	The id of the input object.
`label`	The label to set.
`choices`	The new choices for the input.
`selected`	The values selected.
`status`	Status, only used if choices is not NULL.

size	Size, only used if choices is not NULL.
checkIcon	Icon, only used if choices is not NULL.
choiceNames, choiceValues	List of names and values, an alternative to choices.

See Also

checkboxGroupButtons

Examples

```
## Not run:
if (interactive()) {

library(shiny)
library(shinyWidgets)

# Example 1 ----

ui <- fluidPage(

  radioButtons(inputId = "up", label = "Update button :", choices = c("All", "None")),

  checkboxGroupButtons(
    inputId = "btn", label = "Power :",
    choices = c("Nuclear", "Hydro", "Solar", "Wind"),
    selected = "Hydro"
  ),

  verbatimTextOutput(outputId = "res")

)

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

  observeEvent(input$up, {
    if (input$up == "All"){
     updateCheckboxGroupButtons(session, "btn", selected = c("Nuclear", "Hydro", "Solar", "Wind"))
    } else {
      updateCheckboxGroupButtons(session, "btn", selected = character(0))
    }
  }, ignoreInit = TRUE)

  output$res <- renderPrint({
    input$btn
  })
}

shinyApp(ui = ui, server = server)


# Example 2 ----
```

```
library("shiny")
library("shinyWidgets")

ui <- fluidPage(
  checkboxGroupButtons(
    inputId = "somevalue",
    choices = c("A", "B", "C"),
    label = "My label"
  ),

  verbatimTextOutput(outputId = "res"),

  actionButton(inputId = "updatechoices", label = "Random choices"),
  pickerInput(
    inputId = "updateselected", label = "Update selected:",
    choices = c("A", "B", "C"), multiple = TRUE
  ),
  textInput(inputId = "updatelabel", label = "Update label")
)

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

  output$res <- renderPrint({
    input$somevalue
  })

  observeEvent(input$updatechoices, {
    newchoices <- sample(letters, sample(2:6))
    updateCheckboxGroupButtons(
      session = session, inputId = "somevalue",
      choices = newchoices
    )
    updatePickerInput(
      session = session, inputId = "updateselected",
      choices = newchoices
    )
  })

  observeEvent(input$updateselected, {
    updateCheckboxGroupButtons(
      session = session, inputId = "somevalue",
      selected = input$updateselected
    )
  }, ignoreNULL = TRUE, ignoreInit = TRUE)

  observeEvent(input$updatelabel, {
    updateCheckboxGroupButtons(
      session = session, inputId = "somevalue",
      label = input$updatelabel
    )
  }, ignoreInit = TRUE)
```

```
}

shinyApp(ui = ui, server = server)

}

## End(Not run)
```

---

updateKnobInput *Change the value of a knob input on the client*

Description

Change the value of a knob input on the client

Usage

```
updateKnobInput(session, inputId, label = NULL, value = NULL,
  options = NULL)
```

Arguments

session	Standard shiny session.
inputId	The id of the input object.
label	The label to set for the input object.
value	The value to set for the input object.
options	List of additional parameters to update, use knobInput's arguments.

Examples

```
## Not run:

if (interactive()) {

library("shiny")
library("shinyWidgets")

ui <- fluidPage(
  tags$h1("knob update examples"),
  br(),

  fluidRow(

    column(
      width = 6,
      knobInput(
        inputId = "knob1", label = "Update value:",
        value = 75, angleOffset = 90, lineCap = "round"
```

```
      ),
      verbatimTextOutput(outputId = "res1"),
      sliderInput(
        inputId = "upknob1", label = "Update knob:",
        min = 0, max = 100, value = 75
      )
    ),

    column(
      width = 6,
      knobInput(
        inputId = "knob2", label = "Update label:",
        value = 50, angleOffset = -125, angleArc = 250
      ),
      verbatimTextOutput(outputId = "res2"),
      textInput(inputId = "upknob2", label = "Update label:")
    )

  )
)

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

  output$res1 <- renderPrint(input$knob1)

  observeEvent(input$upknob1, {
    updateKnobInput(
      session = session,
      inputId = "knob1",
      value = input$upknob1
    )
  }, ignoreInit = TRUE)


  output$res2 <- renderPrint(input$knob2)
  observeEvent(input$upknob2, {
    updateKnobInput(
      session = session,
      inputId = "knob2",
      label = input$upknob2
    )
  }, ignoreInit = TRUE)

}

shinyApp(ui = ui, server = server)

}


## End(Not run)
```

updateMaterialSwitch *Change the value of a materialSwitch input on the client*

Description

Change the value of a materialSwitch input on the client

Usage

```
updateMaterialSwitch(session, inputId, value = NULL)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The id of the input object.
value	The value to set for the input object.

See Also

materialSwitch

updateMultiInput *Change the value of a multi input on the client*

Description

Change the value of a multi input on the client

Usage

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

Arguments

session	The session object passed to function given to shinyServer.
inputId	The id of the input object.
label	The label to set.
selected	The values selected.
choices	The new choices for the input.

Note

Thanks to Ian Fellows for this one !

See Also

multiInput

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)

fruits <- c("Banana", "Blueberry", "Cherry",
            "Coconut", "Grapefruit", "Kiwi",
            "Lemon", "Lime", "Mango", "Orange",
            "Papaya")

ui <- fluidPage(
  tags$h2("Multi update"),
  multiInput(
    inputId = "my_multi",
    label = "Fruits :",
    choices = fruits,
    selected = "Banana",
    width = "350px"
  ),
  verbatimTextOutput(outputId = "res"),
  selectInput(
    inputId = "selected",
    label = "Update selected:",
    choices = fruits,
    multiple = TRUE
  ),
  textInput(inputId = "label", label = "Update label:")
)

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

  output$res <- renderPrint(input$my_multi)

  observeEvent(input$selected, {
    updateMultiInput(
      session = session,
      inputId = "my_multi",
      selected = input$selected
    )
  })

  observeEvent(input$label, {
    updateMultiInput(
      session = session,
      inputId = "my_multi",
```

```
      label = input$label
    )
  }, ignoreInit = TRUE)
}

shinyApp(ui, server)

}


## End(Not run)
```

`updateNoUiSliderInput` *Change the value of a no ui slider input on the client*

Description

Change the value of a no ui slider input on the client

Usage

```
updateNoUiSliderInput(session, inputId, value = NULL, range = NULL,
  disable = FALSE)
```

Arguments

`session`	The `session` object passed to function given to `shinyServer`.
`inputId`	The id of the input object.
`value`	The new value.
`range`	The new range, must be of length 2 with `c(min, max)`.
`disable`	logical, disable or not the slider, if disabled the user can no longer modify the slider value

Examples

```
## Not run:

if (interactive()) {

 demoNoUiSlider("update")

}


## End(Not run)
```

updateNumericRangeInput *Change the value of a numeric range input*

Description

Change the value of a numeric range input.

Usage

```
updateNumericRangeInput(session, inputId, label, value)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
value	The initial value(s) for the range. A numeric vector of length one will be duplicated to represent the minimum and maximum of the range; a numeric vector of two or more will have its minimum and maximum set the minimum and maximum of the range.

updatePickerInput *Change the value of a select picker input on the client*

Description

Change the value of a picker input on the client

Usage

```
updatePickerInput(session, inputId, label = NULL, selected = NULL,
  choices = NULL, choicesOpt = NULL)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The id of the input object.
label	Display a text in the center of the switch.
selected	The initially selected value (or multiple values if multiple = TRUE). If not specified then defaults to the first value for single-select lists and no values for multiple select lists.
choices	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.
choicesOpt	Options for choices in the dropdown menu

See Also

pickerInput.

Examples

```
## Not run:
if (interactive()) {

library("shiny")
library("shinyWidgets")

ui <- fluidPage(
  tags$h2("Update pickerInput"),

  fluidRow(
    column(
      width = 5, offset = 1,
      pickerInput(
        inputId = "p1",
        label = "classic update",
        choices = rownames(mtcars)
      )
    ),
    column(
      width = 5,
      pickerInput(
        inputId = "p2",
        label = "disabled update",
        choices = rownames(mtcars)
      )
    )
  ),

  fluidRow(
    column(
      width = 10, offset = 1,
      sliderInput(
        inputId = "up",
        label = "Select between models with mpg greater than :",
        width = "50%",
        min = min(mtcars$mpg),
        max = max(mtcars$mpg),
        value = min(mtcars$mpg),
        step = 0.1
      )
    )
  )

)

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

```
  observeEvent(input$up, {
    mtcars2 <- mtcars[mtcars$mpg >= input$up, ]

    # Method 1
    updatePickerInput(session = session, inputId = "p1",
                      choices = rownames(mtcars2))

    # Method 2
    disabled_choices <- !rownames(mtcars) %in% rownames(mtcars2)
    updatePickerInput(
      session = session, inputId = "p2",
      choices = rownames(mtcars),
      choicesOpt = list(
        disabled = disabled_choices,
        style = ifelse(disabled_choices,
                       yes = "color: rgba(119, 119, 119, 0.5);",
                       no = "")
      )
    )
  }, ignoreInit = TRUE)

}

shinyApp(ui = ui, server = server)

}

## End(Not run)
```

`updatePrettyCheckbox` *Change the value of a pretty checkbox on the client*

Description

Change the value of a pretty checkbox on the client

Usage

```
updatePrettyCheckbox(session, inputId, label = NULL, value = NULL)
```

Arguments

`session`	The session object passed to function given to shinyServer.
`inputId`	The id of the input object.
`label`	The label to set for the input object.
`value`	The value to set for the input object.

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h1("Pretty checkbox update value"),
  br(),

  prettyCheckbox(inputId = "checkbox1",
              label = "Update me!",
              shape = "curve", thick = TRUE, outline = TRUE),
  verbatimTextOutput(outputId = "res1"),
  radioButtons(
    inputId = "update", label = "Value to set:",
    choices = c("FALSE", "TRUE")
  )

)

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

  output$res1 <- renderPrint(input$checkbox1)

  observeEvent(input$update, {
    updatePrettyToggle(session = session,
                       inputId = "checkbox1",
                       value = as.logical(input$update))
  })

}

shinyApp(ui, server)

}


## End(Not run)
```

---

updatePrettyCheckboxGroup

Change the value of a pretty checkbox on the client

Description

Change the value of a pretty checkbox on the client

Usage

```
updatePrettyCheckboxGroup(session, inputId, label = NULL,
  choices = NULL, selected = NULL, inline = FALSE,
  choiceNames = NULL, choiceValues = NULL, prettyOptions = list())
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The id of the input object.
label	The label to set for the input object.
choices	The choices to set for the input object, updating choices will reset parameters like status, shape, ... on the checkboxes, you can re-specify (or change them) in argument prettyOptions.
selected	The value to set for the input object.
inline	If TRUE, render the choices inline (i.e. horizontally).
choiceNames	The choices names to set for the input object.
choiceValues	The choices values to set for the input object.
prettyOptions	Arguments passed to prettyCheckboxGroup for styling checkboxes.

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h1("Update pretty checkbox group"),
  br(),

  fluidRow(
    column(
      width = 6,
      prettyCheckboxGroup(inputId = "checkgroup1",
                          label = "Update my value!",
                          choices = month.name[1:4],
                          status = "danger",
                          icon = icon("remove")),
      verbatimTextOutput(outputId = "res1"),
      br(),
      checkboxGroupInput(
        inputId = "update1", label = "Update value :",
        choices = month.name[1:4], inline = TRUE
      )
    ),
    column(
```

```
      width = 6,
      prettyCheckboxGroup(inputId = "checkgroup2",
                          label = "Update my choices!", thick = TRUE,
                          choices = month.name[1:4],
                          animation = "pulse", status = "info"),
      verbatimTextOutput(outputId = "res2"),
      br(),
      actionButton(inputId = "update2", label = "Update choices !")
    )
  )

)

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

  output$res1 <- renderPrint(input$checkgroup1)

  observeEvent(input$update1, {
    if (is.null(input$update1)) {
      selected_ <- character(0) # no choice selected
    } else {
      selected_ <- input$update1
    }
   updatePrettyCheckboxGroup(session = session, inputId = "checkgroup1", selected = selected_)
  }, ignoreNULL = FALSE)

  output$res2 <- renderPrint(input$checkgroup2)
  observeEvent(input$update2, {
    updatePrettyCheckboxGroup(
      session = session, inputId = "checkgroup2",
     choices = sample(month.name, 4), prettyOptions = list(animation = "pulse", status = "info")
    )
  }, ignoreInit = TRUE)

}

shinyApp(ui, server)

}



## End(Not run)
```

---

updatePrettyRadioButtons *Change the value pretty radio buttons on the client*

Description

Change the value pretty radio buttons on the client

Usage

```
updatePrettyRadioButtons(session, inputId, label = NULL,
  choices = NULL, selected = NULL, inline = FALSE,
  choiceNames = NULL, choiceValues = NULL, prettyOptions = list())
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The id of the input object.
label	The label to set for the input object.
choices	The choices to set for the input object, updating choices will reset parameters like status, shape, ... on the radio buttons, you can re-specify (or change them) in argument prettyOptions.
selected	The value to set for the input object.
inline	If TRUE, render the choices inline (i.e. horizontally).
choiceNames	The choices names to set for the input object.
choiceValues	The choices values to set for the input object.
prettyOptions	Arguments passed to prettyRadioButtons for styling radio buttons

Examples

```
## Not run:

 if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h1("Update pretty radio buttons"),
  br(),

  fluidRow(
    column(
      width = 6,
      prettyRadioButtons(inputId = "radio1",
                         label = "Update my value!",
                         choices = month.name[1:4],
                         status = "danger",
                         icon = icon("remove")),
      verbatimTextOutput(outputId = "res1"),
      br(),
      radioButtons(
        inputId = "update1", label = "Update value :",
        choices = month.name[1:4], inline = TRUE
      )
    ),
    column(
```

```
      width = 6,
      prettyRadioButtons(inputId = "radio2",
                         label = "Update my choices!", thick = TRUE,
                         choices = month.name[1:4],
                         animation = "pulse", status = "info"),
      verbatimTextOutput(outputId = "res2"),
      br(),
      actionButton(inputId = "update2", label = "Update choices !")
    )
  )

)

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

  output$res1 <- renderPrint(input$radio1)

  observeEvent(input$update1, {
    updatePrettyRadioButtons(
      session = session,
      inputId = "radio1",
      selected = input$update1
    )
  }, ignoreNULL = FALSE)

  output$res2 <- renderPrint(input$radio2)
  observeEvent(input$update2, {
    updatePrettyRadioButtons(
      session = session, inputId = "radio2",
      choices = sample(month.name, 4),
      prettyOptions = list(animation = "pulse",
                           status = "info",
                           shape = "round")
    )
  }, ignoreInit = TRUE)

}

shinyApp(ui, server)

 }


## End(Not run)
```

---

updatePrettySwitch *Change the value of a pretty switch on the client*

Description

Change the value of a pretty switch on the client

Usage

```
updatePrettySwitch(session, inputId, label = NULL, value = NULL)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The id of the input object.
label	The label to set for the input object.
value	The value to set for the input object.

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h1("Pretty switch update value"),
  br(),

  prettySwitch(inputId = "switch1", label = "Update me !"),
  verbatimTextOutput(outputId = "res1"),
  radioButtons(
    inputId = "update", label = "Value to set:",
    choices = c("FALSE", "TRUE")
  )

)

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

  output$res1 <- renderPrint(input$switch1)

  observeEvent(input$update, {
    updatePrettySwitch(session = session, inputId = "switch1",
                       value = as.logical(input$update))
  })

}

shinyApp(ui, server)

}


## End(Not run)
```

updatePrettyToggle *Change the value of a pretty toggle on the client*

Description

Change the value of a pretty toggle on the client

Usage

```
updatePrettyToggle(session, inputId, label = NULL, value = NULL)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The id of the input object.
label	The label to set for the input object.
value	The value to set for the input object.

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h1("Pretty toggle update value"),
  br(),

  prettyToggle(inputId = "toggle1",
               label_on = "Checked!",
               label_off = "Unchecked..."),
  verbatimTextOutput(outputId = "res1"),
  radioButtons(
    inputId = "update", label = "Value to set:",
    choices = c("FALSE", "TRUE")
  )

)

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

  output$res1 <- renderPrint(input$toggle1)

  observeEvent(input$update, {
    updatePrettyToggle(session = session,
                       inputId = "toggle1",
```

```
                        value = as.logical(input$update))
  })

}

shinyApp(ui, server)

}


## End(Not run)
```

`updateRadioGroupButtons` *Change the value of a radio group buttons input on the client*

Description

Change the value of a radio group buttons input on the client

Usage

```
updateRadioGroupButtons(session, inputId, label = NULL, choices = NULL,
  selected = NULL, status = "default", size = "normal",
  checkIcon = list(), choiceNames = NULL, choiceValues = NULL)
```

Arguments

`session`	The session object passed to function given to shinyServer.
`inputId`	The id of the input object.
`label`	The label to set.
`choices`	The new choices for the input.
`selected`	The value selected.
`status`	Status, only used if choices is not NULL.
`size`	Size, only used if choices is not NULL.
`checkIcon`	Icon, only used if choices is not NULL.
`choiceNames, choiceValues`	List of names and values, an alternative to choices.

Examples

```
## Not run:

if (interactive()) {

library("shiny")
```

```
library("shinyWidgets")

ui <- fluidPage(
  radioGroupButtons(
    inputId = "somevalue",
    choices = c("A", "B", "C"),
    label = "My label"
  ),

  verbatimTextOutput(outputId = "res"),

  actionButton(inputId = "updatechoices", label = "Random choices"),
  pickerInput(
    inputId = "updateselected", label = "Update selected:",
    choices = c("A", "B", "C"), multiple = FALSE
  ),
  textInput(inputId = "updatelabel", label = "Update label")
)

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

  output$res <- renderPrint({
    input$somevalue
  })

  observeEvent(input$updatechoices, {
    newchoices <- sample(letters, sample(2:6))
    updateRadioGroupButtons(
      session = session, inputId = "somevalue",
      choices = newchoices
    )
    updatePickerInput(
      session = session, inputId = "updateselected",
      choices = newchoices
    )
  })

  observeEvent(input$updateselected, {
    updateRadioGroupButtons(
      session = session, inputId = "somevalue",
      selected = input$updateselected
    )
  }, ignoreNULL = TRUE, ignoreInit = TRUE)

  observeEvent(input$updatelabel, {
    updateRadioGroupButtons(
      session = session, inputId = "somevalue",
      label = input$updatelabel
    )
  }, ignoreInit = TRUE)

}
```

```
shinyApp(ui = ui, server = server)

}


## End(Not run)
```

---

updateSearchInput *Change the value of a search input on the client*

Description

Change the value of a search input on the client

Usage

```
updateSearchInput(session, inputId, label = NULL, value = NULL,
  placeholder = NULL, trigger = FALSE)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The id of the input object.
label	The label to set for the input object.
value	The value to set for the input object.
placeholder	The placeholder to set for the input object.
trigger	Logical, update value server-side as well.

Note

By default, only UI value is updated, use `trigger = TRUE` to update both UI and Server value.

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  tags$h2("Update searchinput"),
  searchInput(
    inputId = "search", label = "Enter your text",
    placeholder = "A placeholder",
    btnSearch = icon("search"),
    btnReset = icon("remove"),
```

```
    width = "450px"
  ),
  br(),
  verbatimTextOutput(outputId = "res"),
  br(),
  textInput(
    inputId = "update_search",
    label = "Update search"
  ),
  checkboxInput(
    inputId = "trigger_search",
    label = "Trigger update search",
    value = TRUE
  )
)

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

  output$res <- renderPrint({
    input$search
  })

  observeEvent(input$update_search, {
    updateSearchInput(
      session = session,
      inputId = "search",
      value = input$update_search,
      trigger = input$trigger_search
    )
  }, ignoreInit = TRUE)
}

shinyApp(ui, server)

}


## End(Not run)
```

`updateSliderTextInput` *Change the value of a slider text input on the client*

Description

Change the value of a slider text input on the client

Usage

```
updateSliderTextInput(session, inputId, label = NULL, selected = NULL,
  choices = NULL, from_fixed = NULL, to_fixed = NULL)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The id of the input object.
label	The label to set.
selected	The values selected.
choices	The new choices for the input.
from_fixed	Fix the left handle (or single handle).
to_fixed	Fix the right handle.

See Also

sliderTextInput

Examples

```
## Not run:

if (interactive()) {
library("shiny")
library("shinyWidgets")

ui <- fluidPage(
  br(),
  sliderTextInput(
    inputId = "mySlider",
    label = "Pick a month :",
    choices = month.abb,
    selected = "Jan"
  ),
  verbatimTextOutput(outputId = "res"),
  radioButtons(
    inputId = "up",
    label = "Update choices:",
    choices = c("Abbreviations", "Full names")
  )
)

server <- function(input, output, session) {
  output$res <- renderPrint(str(input$mySlider))

  observeEvent(input$up, {
    choices <- switch(
      input$up,
      "Abbreviations" = month.abb,
      "Full names" = month.name
    )
    updateSliderTextInput(
      session = session,
      inputId = "mySlider",
```

```
      choices = choices
    )
  }, ignoreInit = TRUE)
}

shinyApp(ui = ui, server = server)
}


## End(Not run)
```

---

updateSpectrumInput *Change the value of a spectrum input on the client*

Description

Change the value of a spectrum input on the client

Usage

```
updateSpectrumInput(session, inputId, selected)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The id of the input object.
selected	The value to select.

Examples

```
## Not run:

if (interactive()) {

library("shiny")
library("shinyWidgets")

ui <- fluidPage(
  tags$h1("Spectrum color picker"),

  br(),

  spectrumInput(
    inputId = "myColor",
    label = "Pick a color:",
    choices = list(
      list('black', 'white', 'blanchedalmond', 'steelblue', 'forestgreen')
    )
  ),
```

```
  verbatimTextOutput(outputId = "res"),
  radioButtons(
    inputId = "update", label = "Update:",
    choices = c(
      'black', 'white', 'blanchedalmond', 'steelblue', 'forestgreen'
    )

  )

)

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

  output$res <- renderPrint(input$myColor)

  observeEvent(input$update, {
    updateSpectrumInput(session = session, inputId = "myColor", selected = input$update)
  }, ignoreInit = TRUE)

}

shinyApp(ui, server)

}


## End(Not run)
```

`updateSwitchInput` *Change the value of a switch input on the client*

Description

Change the value of a switch input on the client

Usage

```
updateSwitchInput(session, inputId, value = NULL, label = NULL,
  onLabel = NULL, offLabel = NULL, onStatus = NULL,
  offStatus = NULL, disabled = NULL)
```

Arguments

`session`	The session object passed to function given to shinyServer.
`inputId`	The id of the input object.
`value`	The value to set for the input object.
`label`	The label to set for the input object.
`onLabel`	The onLabel to set for the input object.

offLabel	The offLabel to set for the input object.
onStatus	The onStatus to set for the input object.
offStatus	The offStatus to set for the input object.
disabled	Logical, disable state.

See Also

switchInput

Examples

```
## Not run:

if (interactive()) {
  library("shiny")
  library("shinyWidgets")


  ui <- fluidPage(
    tags$h1("Update", tags$code("switchInput")),
    br(),
    fluidRow(
      column(
        width = 4,
        panel(
          switchInput(inputId = "switch1"),
          verbatimTextOutput(outputId = "resup1"),
          tags$div(
            class = "btn-group",
            actionButton(
              inputId = "updatevaluetrue",
              label = "Set to TRUE"
            ),
            actionButton(
              inputId = "updatevaluefalse",
              label = "Set to FALSE"
            )
          ),
          heading = "Update value"
        )
      ),

      column(
        width = 4,
        panel(
          switchInput(inputId = "switch2",
                      label = "My label"),
          verbatimTextOutput(outputId = "resup2"),
          textInput(inputId = "updatelabeltext",
                    label = "Update label:"),
          heading = "Update label"
```

```
    )
  ),

  column(
    width = 4,
    panel(
      switchInput(
        inputId = "switch3",
        onLabel = "Yeaah",
        offLabel = "Noooo"
      ),
      verbatimTextOutput(outputId = "resup3"),
      fluidRow(column(
        width = 6,
        textInput(inputId = "updateonLabel",
                  label = "Update onLabel:")
      ),
      column(
        width = 6,
        textInput(inputId = "updateoffLabel",
                  label = "Update offLabel:")
      )),
      heading = "Update onLabel & offLabel"
    )
  )
),

fluidRow(column(
  width = 4,
  panel(
    switchInput(inputId = "switch4"),
    verbatimTextOutput(outputId = "resup4"),
    fluidRow(
      column(
        width = 6,
        pickerInput(
          inputId = "updateonStatus",
          label = "Update onStatus:",
          choices = c("default", "primary", "success",
                      "info", "warning", "danger")
        )
      ),
      column(
        width = 6,
        pickerInput(
          inputId = "updateoffStatus",
          label = "Update offStatus:",
          choices = c("default", "primary", "success",
                      "info", "warning", "danger")
        )
      )
    ),
    heading = "Update onStatus & offStatusr"
```

```
    )
  ),

  column(
    width = 4,
    panel(
      switchInput(inputId = "switch5"),
      verbatimTextOutput(outputId = "resup5"),
      checkboxInput(
        inputId = "disabled",
        label = "Disabled",
        value = FALSE
      ),
      heading = "Disabled"
    )
  ))

)

server <- function(input, output, session) {
  # Update value
  observeEvent(input$updatevaluetrue, {
    updateSwitchInput(session = session,
                      inputId = "switch1",
                      value = TRUE)
  })
  observeEvent(input$updatevaluefalse, {
    updateSwitchInput(session = session,
                      inputId = "switch1",
                      value = FALSE)
  })
  output$resup1 <- renderPrint({
    input$switch1
  })


  # Update label
  observeEvent(input$updatelabeltext, {
    updateSwitchInput(
      session = session,
      inputId = "switch2",
      label = input$updatelabeltext
    )
  }, ignoreInit = TRUE)
  output$resup2 <- renderPrint({
    input$switch2
  })


  # Update onLabel & offLabel
  observeEvent(input$updateonLabel, {
    updateSwitchInput(
      session = session,
```

```
      inputId = "switch3",
      onLabel = input$updateonLabel
    )
  }, ignoreInit = TRUE)
  observeEvent(input$updateoffLabel, {
    updateSwitchInput(
      session = session,
      inputId = "switch3",
      offLabel = input$updateoffLabel
    )
  }, ignoreInit = TRUE)
  output$resup3 <- renderPrint({
    input$switch3
  })


  # Update onStatus & offStatus
  observeEvent(input$updateonStatus, {
    updateSwitchInput(
      session = session,
      inputId = "switch4",
      onStatus = input$updateonStatus
    )
  }, ignoreInit = TRUE)
  observeEvent(input$updateoffStatus, {
    updateSwitchInput(
      session = session,
      inputId = "switch4",
      offStatus = input$updateoffStatus
    )
  }, ignoreInit = TRUE)
  output$resup4 <- renderPrint({
    input$switch4
  })


  # Disabled
  observeEvent(input$disabled, {
    updateSwitchInput(
      session = session,
      inputId = "switch5",
      disabled = input$disabled
    )
  }, ignoreInit = TRUE)
  output$resup5 <- renderPrint({
    input$switch5
  })

}

shinyApp(ui = ui, server = server)

}
```

```
## End(Not run)
```

`updateVerticalTabsetPanel` *Update selected vertical tab*

Description

Update selected vertical tab

Usage

```
updateVerticalTabsetPanel(session, inputId, selected = NULL)
```

Arguments

`session`	The `session` object passed to function given to `shinyServer`.
`inputId`	The id of the `verticalTabsetPanel` object.
`selected`	The name of the tab to make active.

See Also

`verticalTabsetPanel`

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  fluidRow(
    column(
      width = 10, offset = 1,
      tags$h2("Update vertical tab panel example:"),
      verbatimTextOutput("res"),
      radioButtons(
        inputId = "update", label = "Update selected:",
        choices = c("Title 1", "Title 2", "Title 3"),
        inline = TRUE
      ),
      verticalTabsetPanel(
        id = "TABS",
        verticalTabPanel(
```

```
            title = "Title 1", icon = icon("home", "fa-2x"),
            "Content panel 1"
          ),
          verticalTabPanel(
            title = "Title 2", icon = icon("map", "fa-2x"),
            "Content panel 2"
          ),
          verticalTabPanel(
            title = "Title 3", icon = icon("rocket", "fa-2x"),
            "Content panel 3"
          )
        )
      )
    )
  )

  server <- function(input, output, session) {
    output$res <- renderPrint(input$TABS)
    observeEvent(input$update, {
      shinyWidgets:::updateVerticalTabsetPanel(
        session = session,
        inputId = "TABS",
        selected = input$update
      )
    }, ignoreInit = TRUE)
  }

  shinyApp(ui, server)

  }


  ## End(Not run)
```

---

useArgonDash *Use 'argonDash' in 'shiny'*

Description

Allow to use functions from 'argonDash' into a classic 'shiny' app, specifically `argonCard`, `argonTabSet` and `argonInfoCard`.

Usage

```
useArgonDash()
```

Examples

```
## Not run:
if (interactive()) {
```

```
library(shiny)
library(argonR)
library(argonDash)
library(shinyWidgets)

ui <- fluidPage(
  h1("Import argonDash elements inside shiny!", align = "center"),
  h5("Don't need any sidebar, navbar, ...", align = "center"),
  h5("Only focus on basic elements for a pure interface", align = "center"),

  # use this in non dashboard app
  setBackgroundColor(color = "ghostwhite"),
  useArgonDash(),

  fluidRow(
   column(
    width = 6,
     argonCard(
     status = "primary",
     width = 12,
     title = "Card 1",
     hover_lift = TRUE,
     shadow = TRUE,
     icon = "check-bold",
     src = "#",
     "Argon is a great free UI package based on Bootstrap 4
       that includes the most important components and features."
   )
   ),
   column(
    width = 6,
    argonTabSet(
     id = "tab-1",
     card_wrapper = TRUE,
     horizontal = TRUE,
     circle = FALSE,
     size = "sm",
     width = 6,
     iconList = list("cloud-upload-96", "bell-55", "calendar-grid-58"),
     argonTab(
       tabName = "Tab 1",
       active = TRUE,
       sliderInput(
        "number",
        "Number of observations:",
        min = 0,
        max = 100,
        value = 50
       ),
       uiOutput("progress")
     ),
     argonTab(
```

```
      tabName = "Tab 2",
      active = FALSE,
      prettyRadioButtons(
       inputId = "dist",
       inline = TRUE,
       animation = "pulse",
       label = "Distribution type:",
       c("Normal" = "norm",
        "Uniform" = "unif",
        "Log-normal" = "lnorm",
        "Exponential" = "exp")
       ),
       plotOutput("distPlot")
    ),
    argonTab(
      tabName = "Tab 3",
      active = FALSE,
      numericInput("valueBox", "Second value box:", 10, min = 1, max = 100)
    )
   )
  )
 ),
br(),
fluidRow(
 argonInfoCard(
  value = "350,897",
  title = "TRAFFIC",
  stat = 3.48,
  stat_icon = "arrow-up",
  description = "Since last month",
  icon = "chart-bar",
  icon_background = "danger",
  hover_lift = TRUE
 ),
 argonInfoCard(
   value = textOutput("value"),
   title = "NEW USERS",
   stat = -3.48,
   stat_icon = "arrow-down",
   description = "Since last week",
   icon = "chart-pie",
   icon_background = "warning",
   shadow = TRUE
 ),
 argonInfoCard(
   value = "924",
   title = "SALES",
   stat = -1.10,
   stat_icon = "arrow-down",
   description = "Since yesterday",
   icon = "users",
   icon_background = "yellow",
   background_color = "default"
```

```
    ),
    argonInfoCard(
      value = "49,65%",
      title = "PERFORMANCE",
      stat = 12,
      stat_icon = "arrow-up",
      description = "Since last month",
      icon = "percent",
      icon_background = "info",
      gradient = TRUE,
      background_color = "orange",
      hover_lift = TRUE
    )
  )
)

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

 output$progress <- renderUI({
  argonProgress(value = input$number, status = "danger", text = "Custom Text")
 })

 output$distPlot <- renderPlot({
  dist <- switch(input$dist,
                 norm = rnorm,
                 unif = runif,
                 lnorm = rlnorm,
                 exp = rexp,
                 rnorm)

  hist(dist(500))
 })

 output$value <- renderText(input$valueBox)

}

shinyApp(ui, server)

}

## End(Not run)
```

---

useBs4Dash *Use 'bs4Dash' in 'shiny'*

Description

Allow to use functions from 'bs4Dash' into a classic 'shiny' app, specifically `bs4ValueBox`, `bs4InfoBox` and `bs4Card`.

Usage

```
useBs4Dash(old_school = FALSE)
```

Arguments

old_school	FALSE by default. Experimental.

Examples

```
## Not run:
if (interactive()) {

library(shiny)
library(bs4Dash)
library(shinyWidgets)

ui <- fluidPage(
  h1("Import bs4Dash elements inside shiny!", align = "center"),
  h5("Don't need any sidebar, navbar, ...", align = "center"),
  h5("Only focus on basic elements for a pure interface", align = "center"),

  # use this in non dashboard app
  setBackgroundColor(color = "ghostwhite"),
  useBs4Dash(old_school = FALSE),

  # infoBoxes
  fluidRow(
    bs4InfoBox(
      title = "Messages",
      value = 1410,
      icon = "envelope"
      ),
      bs4InfoBox(
        title = "Bookmarks",
        status = "info",
        value = 240,
        icon = "bookmark"
      ),
      bs4InfoBox(
        title = "Comments",
        gradientColor = "danger",
        value = 41410,
        icon = "comments"
      )
  ),

  # valueBoxes
  fluidRow(
    bs4ValueBox(
      value = uiOutput("orderNum"),
      subtitle = "New Orders",
      icon = "credit-card",
```

```
      href = "http://google.com"
    ),
    bs4ValueBox(
      value = "60%",
      subtitle = "Approval Rating",
      icon = "line-chart",
      status = "success"
    ),
    bs4ValueBox(
      value = htmlOutput("progress"),
      subtitle = "Progress",
      icon = "users",
      status = "danger"
    )
  ),

  # Boxes
  fluidRow(
    bs4Card(
     status = "primary",
     sliderInput("orders", "Orders", min = 1, max = 2000, value = 650),
     selectInput(
      "progress",
      "Progress",
       choices = c(
         "0%" = 0, "20%" = 20, "40%" = 40,
          "60%" = 60, "80%" = 80, "100%" = 100
       )
      )
    ),
   bs4Card(
    title = "Histogram box title",
    status = "warning",
    solidHeader = TRUE,
    collapsible = TRUE,
    plotOutput("plot", height = 250)
    )
  )
)

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

  output$orderNum <- renderText({
    prettyNum(input$orders, big.mark=",")
  })

  output$orderNum2 <- renderText({
    prettyNum(input$orders, big.mark=",")
  })

  output$progress <- renderUI({
    tagList(input$progress, tags$sup(style="font-size: 20px", "%"))
  })
```

```
  output$progress2 <- renderUI({
    paste0(input$progress)
  })


  output$plot <- renderPlot({
    hist(rnorm(input$orders))
  })

}

shinyApp(ui, server)

}

## End(Not run)
```

---

useShinydashboard *Use 'shinydashboard' in 'shiny'*

Description

Allow to use functions from 'shinydashboard' into a classic 'shiny' app, specifically `valueBox`, `infoBox` and `box`.

Usage

```
useShinydashboard()
```

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinydashboard)
library(shinyWidgets)

# example taken from ?box

ui <- fluidPage(
  tags$h2("Classic shiny"),

  # use this in non shinydashboard app
  setBackgroundColor(color = "ghostwhite"),
  useShinydashboard(),
  # -----------------
```

```
  # infoBoxes
  fluidRow(
    infoBox(
      "Orders", uiOutput("orderNum2"), "Subtitle", icon = icon("credit-card")
    ),
    infoBox(
      "Approval Rating", "60%", icon = icon("line-chart"), color = "green",
      fill = TRUE
    ),
    infoBox(
      "Progress", uiOutput("progress2"), icon = icon("users"), color = "purple"
    )
  ),

  # valueBoxes
  fluidRow(
    valueBox(
      uiOutput("orderNum"), "New Orders", icon = icon("credit-card"),
      href = "http://google.com"
    ),
    valueBox(
      tagList("60", tags$sup(style="font-size: 20px", "%")),
      "Approval Rating", icon = icon("line-chart"), color = "green"
    ),
    valueBox(
      htmlOutput("progress"), "Progress", icon = icon("users"), color = "purple"
    )
  ),

  # Boxes
  fluidRow(
    box(status = "primary",
        sliderInput("orders", "Orders", min = 1, max = 2000, value = 650),
        selectInput("progress", "Progress",
                    choices = c("0%" = 0, "20%" = 20, "40%" = 40, "60%" = 60, "80%" = 80,
                                "100%" = 100)
        )
    ),
    box(title = "Histogram box title",
        status = "warning", solidHeader = TRUE, collapsible = TRUE,
        plotOutput("plot", height = 250)
    )
  )
)

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

  output$orderNum <- renderText({
    prettyNum(input$orders, big.mark=",")
  })

  output$orderNum2 <- renderText({
    prettyNum(input$orders, big.mark=",")
```

```
  })

  output$progress <- renderUI({
    tagList(input$progress, tags$sup(style="font-size: 20px", "%"))
  })

  output$progress2 <- renderUI({
    paste0(input$progress, "%")
  })


  output$plot <- renderPlot({
    hist(rnorm(input$orders))
  })

}

shinyApp(ui, server)

}


## End(Not run)
```

---

useShinydashboardPlus *Use 'shinydashboardPlus' in 'shiny'*

Description

Allow to use functions from 'shinydashboardPlus' into a classic 'shiny' app.

Usage

```
useShinydashboardPlus()
```

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinydashboard)
library(shinydashboardPlus)
library(shinyWidgets)


# example taken from ?box

ui <- fluidPage(
```

```
tags$h2("Classic shiny"),

# use this in non shinydashboardPlus app
useShinydashboardPlus(),
setBackgroundColor(color = "ghostwhite"),

# boxPlus
fluidRow(
 boxPlus(
   title = "Closable Box with dropdown",
   closable = TRUE,
   status = "warning",
   solidHeader = FALSE,
   collapsible = TRUE,
   enable_dropdown = TRUE,
   dropdown_icon = "wrench",
   dropdown_menu = dropdownItemList(
     dropdownItem(url = "http://www.google.com", name = "Link to google"),
     dropdownItem(url = "#", name = "item 2"),
     dropdownDivider(),
     dropdownItem(url = "#", name = "item 3")
   ),
   p("Box Content")
 ),
 boxPlus(
   title = "Closable box, with label",
   closable = TRUE,
   enable_label = TRUE,
   label_text = 1,
   label_status = "danger",
   status = "warning",
   solidHeader = FALSE,
   collapsible = TRUE,
   p("Box Content")
 )
),

br(),

# gradientBoxes
fluidRow(
  gradientBox(
   title = "My gradient Box",
   icon = "fa fa-th",
   gradientColor = "teal",
   boxToolSize = "sm",
   footer = column(
     width = 12,
     align = "center",
     sliderInput(
       "obs",
       "Number of observations:",
       min = 0, max = 1000, value = 500
```

```
      )
    ),
    plotOutput("distPlot")
   ),
   gradientBox(
    title = "My gradient Box",
    icon = "fa fa-heart",
    gradientColor = "maroon",
    boxToolSize = "xs",
    closable = TRUE,
    footer = "The footer goes here. You can include anything",
    "This is a gradient box"
   )
),

br(),

# extra elements
fluidRow(
column(
 width = 6,
 timelineBlock(
   reversed = FALSE,
   timelineEnd(color = "danger"),
   timelineLabel(2018, color = "teal"),
   timelineItem(
     title = "Item 1",
     icon = "gears",
     color = "olive",
     time = "now",
     footer = "Here is the footer",
     "This is the body"
   ),
   timelineItem(
     title = "Item 2",
     border = FALSE
   ),
   timelineLabel(2015, color = "orange"),
   timelineItem(
     title = "Item 3",
     icon = "paint-brush",
     color = "maroon",
     timelineItemMedia(src = "http://placehold.it/150x100"),
     timelineItemMedia(src = "http://placehold.it/150x100")
   ),
   timelineStart(color = "gray")
 )
),
column(
 width = 6,
 box(
   title = "Box with boxPad containing inputs",
   status = "warning",
```

```
      width = 12,
      fluidRow(
        column(
          width = 6,
          boxPad(
            color = "gray",
            sliderInput(
              "obs2",
              "Number of observations:",
              min = 0, max = 1000, value = 500
            ),
            checkboxGroupInput(
              "variable",
              "Variables to show:",
              c(
                "Cylinders" = "cyl",
                "Transmission" = "am",
                "Gears" = "gear"
              )
            ),

            knobInput(
              inputId = "myKnob",
              skin = "tron",
              readOnly = TRUE,
              label = "Display previous:",
              value = 50,
              min = -100,
              displayPrevious = TRUE,
              fgColor = "#428BCA",
              inputColor = "#428BCA"
            )
          )
        ),
        column(
          width = 6,
          plotOutput("distPlot2", height = "200px"),
          tableOutput("data")
        )
      )
    )
  )
)

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

  output$distPlot <- renderPlot({
   hist(rnorm(input$obs))
  })

  output$distPlot2 <- renderPlot({
    hist(rnorm(input$obs2))
```

```
  })

  output$data <- renderTable({
    head(mtcars[, c("mpg", input$variable), drop = FALSE])
  }, rownames = TRUE)

}

shinyApp(ui, server)

}


## End(Not run)
```

---

useSweetAlert *Load Sweet Alert dependencies*

Description

This function is useless for `sendSweetAlert`, `confirmSweetAlert`, `inputSweetAlert`, but is still needed for `progressSweetAlert`.

Usage

```
useSweetAlert()
```

Note

Use `receiveSweetAlert()` in the UI and `sendSweetAlert()` in the server.

See Also

`sendSweetAlert`, `confirmSweetAlert`, `inputSweetAlert`

---

vertical-tab *Vertical tab panel*

Description

Vertical tab panel

Usage

```
verticalTabsetPanel(..., selected = NULL, id = NULL,
  color = "#112446", contentWidth = 9, menuSide = "left")

verticalTabPanel(title, ..., value = title, icon = NULL,
  box_height = "160px")
```

Arguments

...	For verticalTabsetPanel, verticalTabPanel to include, and for the later, UI elements.
selected	The value (or, if none was supplied, the title) of the tab that should be selected by default. If NULL, the first tab will be selected.
id	If provided, you can use input$id in your server logic to determine which of the current tabs is active. The value will correspond to the value argument that is passed to verticalTabPanel.
color	Color for the tab panels.
contentWidth	Width of the content panel (must be between 1 and 12), menu width will be 12 - contentWidth.
menuSide	Side for the menu: right or left.
title	Display title for tab.
value	Not used yet.
icon	Optional icon to appear on the tab.
box_height	Height for the title box.

See Also

updateVerticalTabsetPanel for updating selected tabs.

Examples

```
## Not run:

if (interactive()) {

library(shiny)
library(shinyWidgets)

ui <- fluidPage(
  fluidRow(
    column(
      width = 10, offset = 1,
      tags$h2("Vertical tab panel example"),
      verticalTabsetPanel(
        verticalTabPanel(
          title = "Title 1", icon = icon("home", "fa-2x"),
          "Content panel 1"
        ),
        verticalTabPanel(
          title = "Title 2", icon = icon("map", "fa-2x"),
          "Content panel 2"
        ),
        verticalTabPanel(
          title = "Title 3", icon = icon("rocket", "fa-2x"),
          "Content panel 3"
        )
```

```
      )
    )
  )
)

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

}

shinyApp(ui, server)

}


## End(Not run)
```

---

wNumbFormat *Format numbers in noUiSliderInput*

Description

Format numbers in noUiSliderInput

Usage

```
wNumbFormat(decimals = NULL, mark = NULL, thousand = NULL,
  prefix = NULL, suffix = NULL, negative = NULL)
```

Arguments

decimals	The number of decimals to include in the result. Limited to 7.
mark	The decimal separator. Defaults to '.' if thousand isn't already set to '.'.
thousand	Separator for large numbers. For example: ' ' would result in a formatted number of 1 000 000.
prefix	A string to prepend to the number. Use cases include prefixing with money symbols such as '$' or ''.
suffix	A number to append to a number. For example: ',-'.
negative	The prefix for negative values. Defaults to '-'.

Value

a named list.

Note

Performed via wNumb JavaScript library : https://refreshless.com/wnumb/.

Examples

```
## Not run:

if (interactive()) {

library( shiny )
library( shinyWidgets )

ui <- fluidPage(
  tags$h3("Format numbers"),
  tags$br(),

  noUiSliderInput(
    inputId = "form1",
    min = 0, max = 10000,
    value = 800,
    format = wNumbFormat(decimals = 3,
                         thousand = ".",
                         suffix = " (US $)")
  ),
  verbatimTextOutput(outputId = "res1"),

  tags$br(),

  noUiSliderInput(
    inputId = "form2",
    min = 1988, max = 2018,
    value = 1988,
    format = wNumbFormat(decimals = 0,
                         thousand = "",
                         prefix = "Year: ")
  ),
  verbatimTextOutput(outputId = "res2"),

  tags$br()

)

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

  output$res1 <- renderPrint(input$form1)
  output$res2 <- renderPrint(input$form2)

}

shinyApp(ui, server)

}


## End(Not run)
```

Index