Package 'shinyWidgets'

January 28, 2018

Title Custom Inputs Widgets for Shiny

Version 0.4.1

Description Some custom inputs widgets to use in Shiny applications, like a toggle switch to replace checkboxes. And other components to pimp your apps.

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

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

Depends R (>= 3.1.0)

Imports shiny (>= 0.14), htmltools, jsonlite

License GPL-3 | file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

Suggests shinydashboard, formatR, viridisLite, RColorBrewer, testthat, covr

NeedsCompilation no

Author Victor Perrier [aut, cre],
Fanny Meyer [aut],
Silvio Moreto [ctb, cph] (bootstrap-select),
Ana Carolina [ctb, cph] (bootstrap-select),
caseyjhol [ctb, cph] (bootstrap-select),
Matt Bryson [ctb, cph] (bootstrap-select),
t0xicCode [ctb, 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),
David Granjon [ctb] (jQuery Knob shiny interface),
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)

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

Repository CRAN

Date/Publication 2018-01-28 16:03:27 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.

Examples

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

 ui <- fluidPage(
  actionBttn(inputId = "id1", label = "Go!", style = "unite")
 )

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

 }

 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)
```

---

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

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)
```

---

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.

Value

a colorSelectorInput control

Functions

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

Examples

```
## Not run:
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)

}

## End(Not run)
```

confirmSweetAlert *Launch a confirmation dialog*

Description

Launch a popup to ask confirmation to the user

Usage

```
confirmSweetAlert(session, inputId, title = "Are you sure ?", text = NULL,
  type = NULL, danger_mode = FALSE, btn_labels = c("Cancel", "Confirm"))
```

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.
danger_mode	Logical, activate danger mode (focus on cancel button).
btn_labels	Labels for buttons.

See Also

sendSweetAlert, inputSweetAlert

Examples

```
## Not run:

if (interactive()) {

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


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

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

```
 observeEvent(input$go, {
   confirmSweetAlert(
     session = session, inputId = "myconfirmation", type = "warning",
     title = "Want to confirm ?", danger_mode = TRUE
   )
 })

  output$res <- renderPrint(input$myconfirmation)

}

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

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` for possible values.
status	Color, must be a valid Bootstrap status : default, primary, info, success, warning, danger.
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`.

Details

This function is similar to `dropdownButton` but don't use Boostrap, 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

Usage

```
dropdownButton(..., circle = TRUE, status = "default", size = "default",
  icon = NULL, label = NULL, tooltip = FALSE, right = FALSE,
  up = FALSE, width = NULL, 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.
inputId	Optional, id for the button, the button act like an actionButton, and you can use the id to toggle the droddown menu server-side.

Examples

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

dropdownButton(
 "Your contents goes here ! You can pass several elements",
 circle = TRUE, status = "danger", icon = icon("gear"), width = "300px",
 tooltip = tooltipOptions(title = "Click to see inputs !")
)

}

## 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)
```

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%.

Value

Numeric value server-side.

See Also

updateKnobInput for ganging 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
choiceValues	List of value to retrieve in server

Value

A multiselect control

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)
```

---

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)
```

---

pickerInput *Select picker Input Control*

Description

Create a select picker (https://silviomoreto.github.io/bootstrap-select/)

Usage

```
pickerInput(inputId, label = NULL, choices, selected = NULL,
  multiple = FALSE, options = NULL, 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	Options to customize the select picker, see https://silviomoreto.github.io/bootstrap-select/options/.
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.

Examples

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

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


# Simple example
```

```
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)


}

## End(Not run)
```

---

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, 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 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).
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)

}


## 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 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 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, 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 checkboxes a bit bigger (`TRUE` or `FALSE`).
`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, 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 an icon created with icon.
icon_off	Optional, display an icon on the checkbox when value is FALSE, must 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).
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)
}


## End(Not run)
```

---

progressBar *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)
```

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.

Value

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

Examples

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

 ui <- fluidPage(
   tags$b("Default"), br(),
   progressBar(id = "pb1", value = 50),
   sliderInput(inputId = "up1", label = "Update", min = 0, max = 100, value = 50)
 )

 server <- function(input, output, session) {
   observeEvent(input$up1, {
     updateProgressBar(session = session, id = "pb1", value = input$up1)
   })
 }

 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, 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.
`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 serverside with `input$<INPUTID>_search` and `input$<INPUTID>_reset`.

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)
```

`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")
```

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.

See Also

confirmSweetAlert, inputSweetAlert

Examples

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

shinyApp(
  ui = fluidPage(
    tags$h1("Click the button"),
    actionButton(
      inputId = "success",
      label = "Launch a success sweet alert"
    ),
    actionButton(
      inputId = "error",
      label = "Launch an error sweet alert"
    )
  ),
  server = function(input, output, session) {
    observeEvent(input$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"
      )
    })
  }
)

}

## 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()
```

`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.

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(), 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.
`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

All options are described here http://bootstrapswitch.com/options.html.

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)
```

Arguments

inputId	Id for the dropdown to toggle

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

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)
```

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.

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`

`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

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)
```

---

updateProgressBar *Update a progress bar*

Description

Change the value of a progress bar on the client

Usage

```
updateProgressBar(session, id, value, total = NULL, status = NULL)
```

Arguments

session	The ‘session‘ object passed to function given to shinyServer.
id	The id of the progress bar to update
value	Value of the progress bar between 0 and 100, if >100 you must provide total
total	Used to calculate percentage if value > 100
status	Color

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)
```

---

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)
```

`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`

Index