Package ‘spsComps’

July 13, 2023

Title 'systemPipeShiny' UI and Server Components
Version 0.3.3.0
Date 2023-07-10
Description The systemPipeShiny (SPS) framework comes with many UI and server components. However, installing the whole framework is heavy and takes some time. If you would like to use UI and server components from SPS in your own Shiny apps, do not hesitate to try this package.

Depends R (>= 4.0.0), shiny (>= 1.5.0)
Imports assertthat, stringr, glue (>= 1.4.0), magrittr, shinytoastr, shinyAce, htmltools, utils, R6, crayon
Suggests testthat, shinyjqui, spsUtil (>= 0.2.0)
License GPL (>= 3)
Encoding UTF-8

BugReports https://github.com/lz100/spsComps/issues
URL https://github.com/lz100/spsComps
RoxygenNote 7.2.3
Config/testthat/edition 3
NeedsCompilation no
Author Le Zhang [aut, cre]
Maintainer Le Zhang <lezhang100@gmail.com>
Repository CRAN
Date/Publication 2023-07-12 22:30:09 UTC

R topics documented:

addLoader ................................................................. 2
animateAppend ........................................................ 9
animateIcon ............................................................ 10
animateUI ............................................................... 12
**addLoader**

Add/remove CSS loaders from server to any Shiny/HTML component. It is useful to indicate busy status when some code is running in the server and when it finishes, remove the loader to indicate clear status.

**Value**

CSS load in R6 class

**Methods**

**Public methods:**

- `addLoader$new()`
- `addLoader$show()`
- `addLoader$hide()`
- `addLoader$destroy()`
- `addLoader$recreate()`
- `addLoader$clone()`

**Method** `new()`: create a loader object

**Usage:**

```r
addLoader$new(
  target_selector = "", 
  isID = TRUE, 
  type = "default", 
  src = "", 
  id = "", 
  height = NULL, 
  width = height, 
  color = "#337ab7", 
  opacity = 1, 
  method = "replace", 
  block = TRUE, 
  center = TRUE, 
  bg_color = "#eee", 
  footer = NULL, 
  z_index = 2000, 
  alert = FALSE, 
  session = shiny::getDefaultReactiveDomain()
)
```

**Arguments:**

- `target_selector` string, which Shiny component you want to add the loader to? a shiny component ID or a valid CSS selector if `isID = FALSE`. for example, you have a button and want to add animation to it:
  ```r
  actionButton(inputId = "btn")
  ```
  This function is used in server only, so if you are in shiny module, use `ns()` for ID on UI but **DO NOT** add the `ns()` wrapper on server.

- `isID` bool, is your selector an ID?

- `type` string, one of "circle", "dual-ring", "facebook", "heart", "ring", "roller", "default", "ellipsis", "grid", "hourglass", "ripple", "spinner", "gif", default is "default".

- `src` string, online URL or local path of the gif animation file if you would like to upload your own loader.

- `id` string, the unique ID for the loader, if not provided, a random ID will be given. If you are using shiny modules, **DO NOT** use `session$ns('YOUR_ID')` to wrap it. Loaders live on the top level of the document.
addLoader

height string, (r)em, "1.5rem", "1.5em", or pixel, like "10px". Default is NULL, will be automatically calculated based on the target component. It is recommend to use NULL for "replace" and "inline" method to let it automatically be calculated, but required for "full_screen" method.

width string, default is the same as height to make it square.

color string, any valid CSS color name, or hex color code

opacity number, between 0-1

method one of "replace", "inline", "full_screen", see details

block bool, for some input components, once the loader starts, it can also block user interaction with the component, very useful for "inline" method, eg. prevent users from clicking the button while some process is still running.

center bool, try to place the load to the center of the target for "inline" and "replace" and center of the screen for "full_screen".

bg_color string, any valid CSS color name, or hex color code. Only works for "full_screen" method.

footer Additional Shiny/HTML component to add below the loader, like a title h1("load title"). inline method does not have a footer.

z_index number, only works for "full_screen" method, what CSS layer should the overlay be placed. In HTML, all elements have the default of 0.

alert bool, should alert if target cannot be found or other javascript errors? mainly for debugging

session shiny session

Details:

Methods:

• replace: use a HTML div with the same CSS styles to replace the original target, but add the loader inside and remove original content inside. When the loader is hide, show the original div and hide this loader div. Height and width is the original div’s height unless specially specified. Good example of this will be some plot outputs.

• inline: append the loader as the first child of target HTML container. loader’s height and width is the original div’s height unless specially specified. In addition, this methods will disable all inputs and buttons inside the target container, so this method can be useful on some buttons.

• full_screen: Do not change anything of the target HTML container, add an overlay to cover the whole page when show and hide the overlay when hide. This method requires the height to be specified manually. Under this method, bg_color and z_index can also be changed.

New container:

addLoader$new() method only stores the loader information, the loader is add to your document upon the first time addLoader$show() is called.

Required javascript and css files:

Since spsComps 0.3.1 all dependencies will be added automatically. If you don’t see them working, try to manually add spsDepend('addLoader') or spsDepend('css-loader') (old name) somewhere in your UI to add the dependency.

Returns: A R6 loader object

Method show(): show the loader
addLoader

Usage:
addLoader$show(alert = FALSE)

Arguments:
alert  bool, if the target selector or loader is not found, alert on UI? For debugging purposes.

Details: Make sure your target element is visible when the time you call this show method, otherwise, you will not get it if height and width is rely on auto-calculation for "replace" and "inline" method. "full_screen" method is not affected.

Method hide(): hide the loader

Usage:
addLoader$hide(alert = FALSE)

Arguments:
alert  bool, if the target selector or loader is not found, alert on UI? For debugging purposes.

Method destroy(): Destroy current loader

Usage:
addLoader$destroy(alert = FALSE)

Arguments:
alert  bool, if the target selector or loader is not found, alert on UI? For debugging purposes.

Details: hide and remove current loader from the current document

Method recreate(): recreate the loader

Usage:
addLoader$recreate(
  type = "default",
  src = NULL,
  id = "",
  height = NULL,
  width = height,
  color = "#337ab7",
  opacity = 1,
  method = "replace",
  block = TRUE,
  center = TRUE,
  bg_color = "#eee",
  footer = NULL,
  z_index = 2000,
  alert = FALSE
)

Arguments:
  type string, one of "circle", "dual-ring", "facebook", "heart", "ring", "roller", "default", "ellipsis", "grid", "hourglass", "ripple", "spinner", "gif", default is "default".
  src string, online URL or local path of the gif animation file if you would like to upload your own loader.
id  string, the unique ID for the loader, if not provided, a random ID will be given. If you are using shiny modules, DO NOT use session$ns('YOUR_ID') to wrap it. Loaders live on the top level of the document.

height  string, (r)em, "1.5rem", "1.5em", or pixel, like "10px". Default is NULL, will be automatically calculated based on the target component. It is recommend to use NULL for "replace" and "inline" method to let it automatically be calculated, but required for "full_screen" method.

width  string, default is the same as height to make it square.

color  string, any valid CSS color name, or hex color code

opacity  number, between 0-1

method  one of "replace", "inline", "full_screen", see details

block  bool, for some input components, once the loader starts, it can also block user interaction with the component, very useful for "inline" method, eg. prevent users from clicking the button while some process is still running.

center  bool, try to place the load to the center of the target for "inline" and "replace" and center of the screen for "full_screen".

bg_color  string, any valid CSS color name, or hex color code. Only works for "full_screen" method.

footer  Additional Shiny/HTML component to add below the loader, like a title h1("load title"). inline method does not have a footer.

z_index  number, only works for "full_screen" method, what CSS layer should the overlay be placed. In HTML, all elements have the default of 0.

alert  bool, should alert if target cannot be found or other javascript errors? mainly for debugging

Details:  This method will first disable then destroy (remove) current loader, and finally store new information of the new loader.

Note:  this method only refresh loader object on the server, the loader is not recreated until the next time show method is called.

Method  clone():  The objects of this class are cloneable with this method.

Usage:
addLoader$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.

Examples
if (interactive()){
  ui <- fluidPage(
    h4("Use buttons to show and hide loaders with different methods"),
    spsDepend("addLoader"), # optional
tags$b("Replace"), br(),
actionButton("b_re_start", "Replace"),
actionButton("b_re_stop", "stop replace"),
br(), tags$b("Inline"), br(),
actionButton("b_in_start", "Inline"),
actionButton("b_in_stop", "stop inline"),
server <- function(input, output, session) {
# Init loaders
loaders <- addLoader$new("b_re_start", type = "facebook")
loaders <- addLoader$new("b_in_start", color = "green", method = "inline")
loaders <- addLoader$new("b_fs_start", color = "pink", method = "full_screen",
bg_color = "#eee", height = "30rem", type = "heart"
)
loaders <- addLoader$new("chunk", type = "spinner", color = "orange",
footer = h5("chunk loader")
)

# toggle loaders
## replace
observeEvent(input$b_re_start, { loader_replace$show()
})
observeEvent(input$b_re_stop, { loader_replace$hide()
})
## inline
observeEvent(input$b_in_start, { loader_inline$show()
})
observeEvent(input$b_in_stop, { loader_inline$hide()
})
## full screen
observeEvent(input$b_fs_start, { loader_fs$show()
Sys.sleep(2)
loader_fs$hide()
})
## chunk

observeEvent(input$chunk_start, {
  loader_chunk$show()
})
observeEvent(input$chunk_stop, {
  loader_chunk$hide()
})

shinyApp(ui, server)

if (interactive()){
  ui <- bootstrapPage(
    spsDepend("addLoader"), # optional
    h4("Add loaders to Shiny `render` events"),
    tags$b("Replace"), br(),
    selectizeInput(inputId = "n_re",
      label = "Change this to render the following plot",
      choices = c(10, 20, 35, 50)),
    plotOutput(outputId = "p_re"),
    br(),
    selectInput(inputId = "n_fs",
      label = "Change this to render the following plot",
      choices = c(10, 20, 35, 50)),
    plotOutput(outputId = "p_fs"))
  )

  server <- function(input, output, session) {
    # create loaders
    l_re <- addLoader$new("p_re")
    l_fs <- addLoader$new("p_fs",
      color = "pink", method = "full_screen",
      bg_color = "#eee", height = "30rem", type = "grid",
      footer = h4("Replotting..."
    )
    # use loaders in rendering
    output$p_re <- renderPlot({
      on.exit(l_re$hide())
      # to make it responsive
      # (always create a new one by calculating the new height and width)
      l_re$recreate()$show()
      Sys.sleep(1)
      hist(faithful$eruptions,
        probability = TRUE,
        breaks = as.numeric(input$n_re),
        xlab = "Duration (minutes)",
        main = "Geyser eruption duration"
      )
    })
    output$p_fs <- renderPlot({
      on.exit(l_fs$hide())
      l_fs$show()
    })
  }
}
animateAppend

```
Sys.sleep(1)
hist(faithful$eruptions,
   probability = TRUE,
   breaks = as.numeric(input$n_fs),
   xlab = "Duration (minutes)",
   main = "Geyser eruption duration")
```

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

animateAppend

Append animation to a Shiny element

Description

Append animation to a Shiny element

Usage

```
animateAppend(element, animation, speed = NULL, hover = FALSE)
```

```
animateAppendNested(
   element, 
   animation, 
   speed = NULL, 
   hover = FALSE, 
   display = "inline-block", 
   ...
)
```

Arguments

- **element**: the shiny element to append, must have "shiny.tag" class for animateAppend and can be either "shiny.tag" or "shiny.tag.list" for animateAppendNested
- **speed**: string, one of "fast", "slow"
- **hover**: bool, trigger animation on hover?
- **display**: string, CSS display method for the out-most wrapper, one of the valid css display method, like "block", "inline", "flex", default is "inline-block".
- **...**: other attributes add to the wrapper, for animateAppendNested only
animateIcon

Font awesome animated icons

Details

animateAppend:
Append the animation directly to the element you provide, but can only apply one type of animation.

animateAppendNested:
Append multiple animations to the element you provide by creating a wrapper around the element. Animations are applied on the wrappers. This may cause some unknown issues, especially on the display property. Try change the display may fix the issues. It is safer to use animateAppend.

Read more about CSS display: https://www.w3schools.com/cssref/pr_class_display.asp

Value

returns a Shiny element

Examples

if (interactive()){
  library(shiny)

  ui <- fluidPage(
    icon("house") %>%
      animateAppend("ring"),
    h2("Append animation", class = "text-primary") %>%
      animateAppend("pulse"),
    br(),
    h2("Nested animations", class = "text-primary") %>%
      animateAppendNested("ring") %>%
      animateAppendNested("pulse") %>%
      animateAppendNested("passing"),
    tags$span("Other things"),
    h2("Nested animations display changed", class = "text-primary") %>%
      animateAppendNested("ring") %>%
      animateAppendNested("pulse", display = "block", style = "width: 30%")
  )

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

  }

  shinyApp(ui, server)
}
animateIcon

Description

Greatly enhance the shiny::icon with animations. Built on top of font-awesome-animation.

Usage

animateIcon(
  name,
  animation = NULL,
  speed = NULL,
  hover = FALSE,
  color = "",
  size = NULL,
  ...
)

Arguments

name  string, the name of the font-awesome icon
speed  string, one of "fast", "slow"
hover  bool, trigger animation on hover?
color  string, color of the icon, a valid color name or hex code
size  string, change font-awesome icon size, one of "xs", "sm", "lg", "2x", "3x", "5x", "7x", "10x". See examples.
...  append additional attributes you want to the icon

Details

If you don’t specify any animation, it will work the same as the original shiny::icon function. Fully compatible with any shiny functions that requires an icon as input.

Value

a icon tag

Examples

if(interactive()){
  library(shiny)

  ui <- fluidPage(
    style = "text-align: center;",
    tags$label("same as original icon function"), br(),
    animateIcon("house"), br(),
    tags$label("Change animation and color"), br(),
  )}
animateUI

Add/remove animation to any HTML/shiny component

Description

Add animation to a HTML or component and remove it

Usage

animateUI(selector, animation, speed = NULL, hover = FALSE, isID = TRUE)

animateServer(
  selector,
  animation = NULL,
  speed = NULL,
  hover = FALSE,
  isID = TRUE,
  session = shiny::getDefaultReactiveDomain()
)
animationRemove(
    selector,
    isID = TRUE,
    alert = FALSE,
    session = shiny::getDefaultReactiveDomain()
)

Arguments

selector string, a shiny component ID or a valid CSS selector if isID = FALSE. For example, you have a button and want to add animation to it:

```
actionButton(inputId = "btn")
```

Then the selector is "btn" selector = 'btn'. If you are using shiny modules, use ns() to wrap it in UI for the button actionButton(inputId = ns("btn")), and also add ns() to selector selector = ns('btn') for the animateUI function. If you are using the server side functions animateServer and animationRemove, DO NOT add the ns() wrapper.


speed string, one of "fast", "slow"

hover bool, trigger animation on hover?

isID bool, is your selector an ID?

session the current shiny session

alert bool, for animationRemove only: if the component is not found or it does not contain any animation or the animation is not added by spsComps, alert on UI? More like for debugging purposes.

Details

- animateUI: use on the UI side, which means add the animation when UI loads complete.
- animateServer: use on the server side. Use server to trigger the animation on a component at some point.
- animationRemove: use on the server side, to remove animation on a certain component.

Selector:

Usually for beginners use the shiny component ID is enough, but sometimes a HTML element may not have the 'id' attribute. In this case, you can still animate the element by advanced CSS selector. For these selectors, turn off the isID = FALSE and provide the selector in a single string. Google "CSS selector" to learn more.
only server functions:
If you use animateServer or animationRemove on the server, but not animateUI you don’t have
to load the required CSS and javascript, since spsComps 0.3.1. In case they don’t work, you can
manually add the dependency by adding spsDepend("animation") somewhere in your UI. see
examples.

Value

see details

Examples

if(interactive()){
  library(shiny)
  ui <- fluidPage(
    spsDepend("animation"), # optional
    column(
      6,
      h3("Adding animations from UI"),
      tags$label("to a button"), br(),
      actionButton("btn1", "random button"), br(),
      animateUI("btn1", animation = "ring"),
      tags$label("to some text"), br(),
      p(id = "mytext", class = "text-red", "some move text"), br(),
      animateUI("mytext", animation = "horizontal", speed = "fast"),
      tags$label("on hover, move mouse on the red thumb"), br(),
      actionButton("btn2", ",
        icon = icon(id = "myicon", "thumbs-up"),
        style = "color: red; border: initial; border-color: transparent;"
      ), br(),
      animateUI("btn2", animation = "bounce", speed = "fast", hover = TRUE),
      tags$label("on a plot"), br(),
      plotOutput("plot1"),
      animateUI("plot1", animation = "float", speed = "fast")
    ),
    column(
      6,
      h3("Adding/removing animations from server"),
      tags$label("use a button to control"), br(),
      actionButton("btn3", "animate itself"),
      actionButton("btn4", "stop animation"), br(),
      tags$label("advanced selector in for complex group"), br(),
      sliderInput("myslider",
        label = "animating if less than 5",
        value = 0,
        min = 0, max = 10,
        step = 1
      ),
      sliderInput("
server <- function(input, output, session) {
  output$plot1 <- renderPlot(plot(1:10, 10:1))
  observeEvent(input$myslider, {
    if (input$myslider <= 5) {
      animateServer(
        selector = "#myslider",
        animation = "horizontal", speed = "slow", isID = FALSE
      )
    } else {
      animationRemove(
        selector = "#myslider",
        isID = FALSE
      )
    }
  })
  observeEvent(input$btn3, {
    animateServer("btn3", animation = "flash", speed = "slow")
  })
  observeEvent(input$btn4, {
    animationRemove("btn3")
  })
}
shinyApp(ui, server)

bsAlert

**Bootstrap3 alert**

**Description**

Add a Bootstrap3 alert component to the UI

**Usage**

bsAlert(..., status = "success", closeable = TRUE)

**Arguments**

... any shiny tag or tagList you want to add to the alert body, or any additional attributes you want to add to the alert element.

status string, one of "success", "info", "warning", "danger"

closeable bool, can the alert be closed?
Details
Read more here: https://getbootstrap.com/docs/3.3/components/#alerts

Value
shiny tag element

Examples
if(interactive()) {
library(shiny)
ui <- fluidPage(
  bsAlert(tags$b("Success: "), "You made it", status = "success"),
  bsAlert(tags$b("Info: "), "Something happened", status = "info"),
  bsAlert(tags$b("Warning: "), "Something is not right", status = "warning"),
  bsAlert(tags$b("Danger: "), "Oh no...", status = "danger")
)
server <- function(input, output, session) {}
shinyApp(ui, server)
}

bsPopover
Enhanced Bootstrap3 popover

Description
Add popover to any Shiny element you want. You can also customize color, font size, background color, and more for each individual popover.

Usage
bsPopover(
tag,
title = " ",
content = " ",
placement = "top",
bgcolor = "#ebebeb",
titlecolor = "black",
contentcolor = "black",
titlesize = "14px",
contentsize = "12px",
titleweight = "600",
contentweight = "400",
opacity = 1,
html = FALSE,
trigger = "hover",
click_inside = FALSE
)
bsHoverPopover(
    tag,
    title = "",  
    content = "", 
    placement = "top", 
    bgcolor = "#ebebeb", 
    titlecolor = "black", 
    contentcolor = "black", 
    titlesize = "14px", 
    contentsize = "12px", 
    titleweight = "600", 
    contentweight = "400", 
    opacity = 1, 
    html = FALSE, 
    trigger = "hover", 
    click_inside = FALSE
)

bsPop(
    tag,
    title = "", 
    content = "", 
    placement = "top", 
    status = "primary", 
    titlesize = "14px", 
    contentsize = "12px", 
    titleweight = "600", 
    contentweight = "400", 
    opacity = 1, 
    html = TRUE, 
    trigger = "hover", 
    click_inside = FALSE
)

Arguments

tag           a shiny tag as input

title          string, popover title

content        string, popover content

placement      string, one of "top", "bottom", "left", "right", where to put the tooltip

bgcolor        string, background color, valid value of CSS color name or hex value or rgb value

titlecolor     string, title text color, valid value of CSS color name or hex value or rgb value

contentcolor   string, content text color, valid value of CSS color name or hex value or rgb value

titlesize      string, title text font size, valid value of CSS font size, like "10px", "1rem".
bsPopover

- contentsize: string, content text font size, valid value of CSS font size, like "10px", "1rem".
- titleweight: string, CSS valid title font weight unit
- contentweight: string, CSS valid content font weight unit
- opacity: numeric, between 0 and 1
- html: bool, allow title contain HTML code? like "<strong>abc</strong>"
- trigger: string, how to trigger the tooltip, one or combination of click | hover | focus | manual.
- click_inside: bool, default is FALSE, whether to allow users to click content inside the message. See details.
- status: string, used only for wrapper bsPop, see details

Details

2. For font weight, see: https://www.w3schools.com/cssref/pr_font_weight.asp
3. bsHoverPopover is the old name but we still keep it for backward compatibility.

Click inside the message:
Sometimes developers want to add links for users to click. By default, the message will be gone once mouse leaves the element, but with this option to be TRUE, when users move the mouse inside, the message element will not be gone, so users can click on the links or other content.
Once this option is used, the triggering method is set to "manual" and animation will be removed.
This is related to the Javascript method used behind, some compromises have to be made.
When adding the links, you may also want to turn html = TRUE in combined.

Convenient wrapper function:
bsPop is the convenient function for bsPopover, which has the background and content color set to 5 different bootstrap colors, you can use status to set, one of "primary", "info", "success", "warning", "danger"

Value

shiny tag

Examples

```r
if(interactive()){
  library(shiny)
  library(magrittr)
  ui <- fluidPage(
    br(), br(), br(), br(), br(), br(), column(2),
    actionButton("", "Popover on the left") %>%
      bsPopover("Popover on the left", "content", "left"),
    actionButton("", "Popover on the top") %>%
      bsPopover("Popover on the top", "content", "top"),
    actionButton("", "Popover on the right") %>%
      bsPopover("Popover on the right", "content", "right"),
  )
}
```
```r
actionButton("", "Popover on the bottom") %>%
  bsPopover("Popover on the bottom", "content", "bottom"),
  br(), br(), column(2),
actionButton("", "primary color") %>%
  bsPopover(
    "primary color", "content", bgcolor = "#0275d8",
    titlecolor = "white", contentcolor = "#0275d8"),
actionButton("", "danger color") %>%
  bsPopover(
    "danger color", "content", bgcolor = "#d9534f",
    titlecolor = "white", contentcolor = "#d9534f"),
actionButton("", "warning color") %>%
  bsPopover(
    "warning color", "content", bgcolor = "#f0ad4e",
    titlecolor = "white", contentcolor = "#f0ad4e"),
  br(), br(), column(2),
actionButton("", "9px & 14px") %>%
  bsPopover("9px", "14", titlesize = "9px", contentsize = ),
actionButton("", "14px & 12px") %>%
  bsPopover("14px", "12", titlesize = "14px"),
actionButton("", "20px & 9px") %>%
  bsPopover("20px", "9", titlesize = "20px"),
  br(), br(), column(2),
actionButton("", "weight 100 & 800") %>%
  bsPopover("weight 100", "800", titleweight = "100", contentweight = "800"),
actionButton("", "weight 400 & 600") %>%
  bsPopover("weight 400", "600", titleweight = "400", contentweight = "600"),
actionButton("", "weight 600 & 400") %>%
  bsPopover("weight 600", "400", titleweight = "600", contentweight = "400"),
actionButton("", "weight 900 & 200") %>%
  bsPopover("weight 900", "200", titleweight = "900", contentweight = "200"),
  br(), br(), column(2),
actionButton("", "opacity 0.2") %>%
  bsPopover("opacity 0.2", opacity = 0.2),
actionButton("", "opacity 0.5") %>%
  bsPopover("opacity 0.5", opacity = 0.5),
actionButton("", "opacity 0.8") %>%
  bsPopover("opacity 0.8", opacity = 0.8),
actionButton("", "opacity 1") %>%
  bsPopover("opacity 1", opacity = 1),
  br(), br(), column(2),
actionButton("f1", "allow html: 'abc<span class='text-danger'>danger</span>'") %>%
  bsPopover(HTML("abc<span class='text-danger'>danger</span>'"),
    html = TRUE, bgcolor = "#0275d8"),
actionButton("f2", "allow html: '<s>del content</s>'") %>%
  bsPopover(HTML("<s>del content</s>"), html = TRUE, bgcolor = "#d9534f"),
actionButton("", "Clickable with links") %>%
  bsPopover(
    title = "Clickable with links",
    content = "<div>This message has a <a href='https://google.com'>link</a></div>",
    html = TRUE, click_inside = TRUE, bgcolor = "orange"
  )
```

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

if(interactive()){
  library(shiny)
  library(magrittr)
  ui <- fluidPage(
    br(), br(), br(), br(), br(), br(), column(2),
    actionButton("", "primary") %>%
     .bsPop("primary", "primary", status = "primary"),
    actionButton("", "info") %>%
      .bsPop("info", "info", status = "info"),
    actionButton("", "success") %>%
      bsPop("success", "success", status = "success"),
    actionButton("", "warning") %>%
      bsPop("warning", "warning", status = "warning"),
    actionButton("", "danger") %>%
      bsPop("danger", "danger", status = "danger")
  )
  server <- function(input, output, session) {}
  shinyApp(ui, server)
}

---

bsTooltip

**Enhanced Bootstrap3 tooltip**

**Description**

Add tooltip to any Shiny element you want. You can also customize color, font size, background color, trigger event for each individual tooltip.

**Usage**

bsTooltip(
  tag,
  title = ",",
  placement = "top",
  bgcolor = "black",
  textcolor = "white",
  fontsize = "12px",
  fontweight = "400",
  opacity = 1,
  html = FALSE,
  trigger = "hover",
  click_inside = FALSE
)

bsTip(
  tag,
bsTooltip

title = "", placement = "top", status = "primary", fontsize = "12px", fontweight = "400", opacity = 1, html = FALSE, trigger = "hover", click_inside = FALSE

Arguments

tag: a shiny tag as input

title: string, tooltip text

placement: string, one of "top", "bottom", "left", "right", where to put the tooltip

bgcolor: string, background color, valid value of CSS color name or hex value or rgb value

textcolor: string, text color, valid value of CSS color name or hex value or rgb value

fontsize: string, text font size, valid value of CSS font size, like "10px", "1rem".

fontweight: string, valid font weight unit: https://www.w3schools.com/cssref/pr_font_weight.asp

opacity: numeric, between 0 and 1

html: bool, allow title contain HTML code? like "<strong>abc</strong>" click | hover | focus | manual.

trigger: string, how to trigger the tooltip, one or combination of

click_inside: bool, default is FALSE, whether to allow users to click content inside the message. See details.

status: string, used only for wrapper bsTip, see details

Details


Click inside the message:
Sometimes developers want to add links for users to click. By default, the message will be gone once mouse leaves the element, but with this option to be TRUE, when users move the mouse inside, the message element will not be gone, so users can click on the links or other content. Once this option is used, the triggering method is set to "manual" and animation will be removed. This is related to the Javascript method used behind, some compromises have to be made. When adding the links, you may also want to turn html = TRUE in combined.

Convenient wrapper function:
bsTip is the convenient function for bsTooltip, which has the background and content color set to 5 different bootstrap colors, you can use status to set, one of "primary", "info", "success", "warning", "danger"
Value

shiny tag

Examples

```r
if(interactive()){
  library(shiny)
  library(magrittr)
  ui <- fluidPage(
    br(), br(), br(), br(), br(), column(2),
    actionButton("", "Tooltip on the left") %>%
      bsTooltip("Tooltip on the left", "left"),
    actionButton("", "Tooltip on the top") %>%
      bsTooltip("Tooltip on the top", "top"),
    actionButton("", "Tooltip on the right") %>%
      bsTooltip("Tooltip on the right", "right"),
    actionButton("", "Tooltip on the bottom") %>%
      bsTooltip("Tooltip on the bottom", "bottom"),
    br(), br(), column(2),
    actionButton("", "primary color") %>%
      bsTooltip("primary color", bgcolor = "#0275d8"),
    actionButton("", "danger color") %>%
      bsTooltip("danger color", bgcolor = "#d9534f"),
    actionButton("", "warning color") %>%
      bsTooltip("warning color", bgcolor = "#f0ad4e"),
    br(), br(), column(2),
    actionButton("", "9px") %>%
      bsTooltip("9px", fontsize = "9px"),
    actionButton("", "14px") %>%
      bsTooltip("14px", fontsize = "14px"),
    actionButton("", "20px") %>%
      bsTooltip("20px", fontsize = "20px"),
    br(), br(), column(2),
    actionButton("", "combined") %>%
      bsTooltip(
        "custom tooltip", "bottom",
        "#0275d8", "#eee", "15px"
      ),
    actionButton("", "Clickable with links") %>%
      bsTooltip(
        "<div>This message has a <a href=\"https://google.com\" target=\"_blank\">link</a></div>", "bottom",
        html = TRUE, click_inside = TRUE, bgcolor = "orange"
      ),
  )
  server <- function(input, output, session) {}
  shinyApp(ui, server)
}
```
clearableTextInput

A clearable text input input control

description

An UI component with a "X" button in the end to clear the entire entered text. It works the same as Textinput.

usage

clearableTextInput(
  inputId,
  label = "", 
  value = "", 
  placeholder = "", 
  style = "width: 100%;"
)

arguments

inputId ID
label text label above
value default value
placeholder place holder text when value is empty
style additional CSS styles you want to apply

value

a shiny component
Examples

```r
if(interactive()){

  ui <- fluidPage(
    clearableTextInput("input1", "This is a input box", style = "width: 50%;"),
    verbatimTextOutput("out1")
  )

  server <- function(input, output, session) {
    output$out1 <- renderPrint(input$input1)
  }

  shinyApp(ui, server)
}
```

**cssLoader**

*Create a variety of CSS loaders on UI*

**Description**

CSS loaders can improve user experience by adding a small animation icon to a HTML element. `spsComps` provides you 12 different looking CSS loaders. Unlike other Shiny packages, you have full control of the CSS loader here, like position, color, size, opacity, etc.

**Usage**

```r
cssLoader(
  type = "default",
  src = "",
  id = "",
  height = "1.5rem",
  width = height,
  color = "#337ab7",
  opacity = 1,
  inline = FALSE,
  is_icon = FALSE,
  ...
)
```

**Arguments**

- **type** string, one of "circle", "dual-ring", "facebook", "heart", "ring", "roller", "default", "ellipsis", "grid", "hourglass", "ripple", "spinner", "gif", default is "default".
- **src** string, online URL or local path of the gif animation file if you would like to upload your own loader.
- **id** string, optional, ID for the component, if not given, a random ID will be given.
height  
  string, pixel, like "10px"; or (r)em, "1.5rem", "1.5em". Default is "1.5rem".

width  
  string, default is the same as height. For most loader, you want to keep width = height for a square shape.

color  
  string, any valid CSS color name, or hex color code

opacity  
  number, between 0-1

inline  
  bool, do you want the loader be inline? This is useful to turn on if you want to add the loader to a shiny::actionButton, so the loader and button label will be on the same line. See examples.

is_icon  
  bool, default uses the HTML div tag, turn on this option will use the i tag for icon. Useful if you want to add the loader as icon argument for the shiny::actionButton. See examples.

...  
  other shiny tags or HTML attributes you want to add to the loader.

Details

'rem' unit:
For most modern web apps, including Shiny, 1rem = 10px

Value
returns a css loader component.

Examples

```r
if (interactive()){  
  library(shiny)  
  heights <- paste0(c(1.5, 3, 5, 8, 10, 15, 20), "rem")  
  colors <- list(  
    colorRampPalette(c("#00d2ff", "#3a7bd5"))(7),  
    colorRampPalette(c("#59c173", "#a17fe0", "#5d26c1"))(7),  
    colorRampPalette(c("#667db6", "#0082c8", "#5d26c1", "#667db6"))(7),  
    colorRampPalette(c("#f2709c", "#ff9472"))(7),  
    colorRampPalette(c("#f5c7d", "#6a82fb"))(7),  
    colorRampPalette(c("#4568dc", "#b06ab3"))(7)  
  )  
  types <- c("circle", "dual-ring", "facebook", "heart", "ring", "roller", "default", "ellipsis", "grid", "hourglass", "ripple", "spinner")  
  ui <- fluidPage(  
    lapply(seq_along(types), function(i){  
      div(  
        h4(types[i]), br(),  
        lapply(1:7, function(x){  
          cssLoader(  
            types[i], height = heights[x],  
            color = colors[if(i > 6) i - 6 else i][x],  
            inline = TRUE  
          )  
        })  
      )  
    }))
```
```r
br()
)
})
server <- function(input, output, session) {
  shinyApp(ui, server)
}
# use with buttons
if (interactive()){
  library(shiny)
  ui <- fluidPage(
    actionButton("btn-a", "",
      ## 'inline = TRUE' is important if you want loader and
      ## text in the same line.
      icon = cssLoader(is_icon = TRUE, inline = TRUE, color = "#3a7bd5"
    )
  ),
  actionButton("btn-b", "Loading",
    icon = cssLoader(type = "hourglass", is_icon = TRUE, color = "#667db6", inline = TRUE)
  )
  server <- function(input, output, session) {}
  shinyApp(ui, server)
}
# use your own
if (interactive()){
  library(shiny)
  spinner <- "https://github.com/lz100/spsComps/blob/master/examples/demo/www/spinner.gif?raw=true"
  eater <- "https://github.com/lz100/spsComps/blob/master/examples/demo/www/bean_eater.gif?raw=true"
  ui <- fluidPage(
    cssLoader("gif", spinner, height = "50px"
  ),
    cssLoader("gif", spinner, height = "100px"
  ),
    cssLoader("gif", eater, height = "150px"
  ),
    cssLoader("gif", eater, height = "200px"
  ),
    actionButton("btn-custom1", "",
      icon = cssLoader(
        type = "gif", src = spinner,
        is_icon = TRUE, inline = TRUE
      )
    ),
    actionButton(
```
A shiny gallery component

Description

Create a gallery to display images or photos
texts, hrefs, images Must have the same length
If there is any image that you do not want to add links, use "" to occupy the space, e.g
hrefs = c("https://xxx.com", "", "https://xxx.com")
If the link is empty, there will be no hover effect on that image, and you cannot click it.
Similar to hrefs, for the texts, use "" to occupy space

Usage

gallery(
texts,
hrefs,
images,
Id = NULL,
title = "Gallery",
title_color = "#0275d8",
image_frame_size = 4,
enlarge = FALSE,
enlarge_method = c("inline", "modal"),
target_blank = FALSE,
style = ""
)

Arguments

texts vector of labels under each image
hreffs vector of links when each image is clicked
images a vector of image sources, can be online URLs or local resource paths.
Id ID of this gallery
### gallery

<table>
<thead>
<tr>
<th>title</th>
<th>Title of gallery</th>
</tr>
</thead>
<tbody>
<tr>
<td>title_color</td>
<td>Title color</td>
</tr>
<tr>
<td>image_frame_size</td>
<td>integer, 1-12, this controls width. How large is each image. 12 is the whole width of the parent container and 1 is 1/12 of the container. Consider numbers that can be fully divided by 12, like 1 (12 per row), 2 (6 per row), 3 (4 per row), 4 (3 per row), 6 (1 per row) or 12 (if you want only 1 image per row).</td>
</tr>
<tr>
<td>enlarge</td>
<td>bool, when click on the image, enlarge it? If enlarge is enabled, click the photo will enlarge instead of jump to the link. Only the title below contains the link if enlarge is enabled.</td>
</tr>
<tr>
<td>enlarge_method</td>
<td>how the photo is enlarged on click, one of &quot;inline&quot; – within the gallery change the size of photo to 12, &quot;modal&quot; – display photo in a pop-up modal.</td>
</tr>
<tr>
<td>target_blank</td>
<td>bool, whether to add target=&quot;_blank&quot; to the link?</td>
</tr>
<tr>
<td>style</td>
<td>additional CSS style you want to add to the most outside component &quot;div&quot;</td>
</tr>
</tbody>
</table>

**Details**

**modal enlarge:**
When view the modal enlarged images, click the "X" button or anywhere outside the image to close the full screen view.

**Value**

a gallery component

**Examples**

```r
if(interactive()){
  texts <- c("p1", "p2", "", "p4", "p5")
             "https://github.com/lz100/spsComps/blob/master/img/2.jpg?raw=true",
             "https://github.com/lz100/spsComps/blob/master/img/5.jpg?raw=true")
               "https://github.com/lz100/spsComps/blob/master/img/2.jpg?raw=true",
               "https://github.com/lz100/spsComps/blob/master/img/5.jpg?raw=true")
  library(shiny)
  ui <- fluidPage(
    column(
      6,
      gallery(texts = texts, hrefs = hrefs, images = images, title = "Default gallery"),
      spsHr(),
      gallery(texts = texts, hrefs = hrefs, images = images,
               image_frame_size = 2, title = "Photo size"),
      spsHr(),
    )
  )
}
```
heightMatcher

Match height of one element to the other element

Description

Match the height of one element to the second element. If the height of second element change, the height of first element will change automatically.

Usage

heightMatcher(div1, div2, isID = TRUE)

Arguments

div1 element ID, or jquery selector if isID = FALSE. The first element that you want to match the height to the other element

div2 matched element ID or selector, the other element

isID bool, if TRUE, div1 and div2 will be treated as ID, otherwise you can use complex jquery selector

Value

tagList containing javascript

Examples

if(interactive()){
  library(shiny)
  library(shinyjqui)
  ui <- fluidPage(
    column(}
3, id = "a",
  style = "border: 1px black solid; background-color: gray;",
  p("This block's height is matched with orange one")
),
shinyjqui::jqui_resizable(column(
  2, id = "b",
  style = "border: 1px black solid; background-color: orange;",
  p("drag the bottom-right corner")
)),
column(
  3, id = "c",
  style = "border: 1px black solid; background-color: red;",
  p("This block's is not matched with others")
),
heightMatcher("a", "b")
)

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

}
# Try to drag `b` from bottom right corner and see what happens to `a`
shinyApp(ui, server)

---

hexLogo

**Hexagon logo and logo panel**

**Description**

Shiny UI widgets to generate hexagon logo(s). `hexLogo()` generates a single hexagon, and `hexPanel()` generates a panel of hex logos.

**Usage**

`hexLogo(`
  id,
  title = "",
  hex_img,
  hex_link = "",
  footer = "",
  footer_link = "",
  x = "-10",
  y = "-20",
  target_blank = FALSE
`)

`hexPanel(`
  id,
  title,
  id,
  title,
hexLogos, hex_links = NULL, hex_titles = NULL, footers = NULL, footer_links = NULL, xs = NULL, ys = NULL, target_blank = FALSE)

Arguments

id  input ID
title  title of the logo, display on top of logo or title of logo panel displayed on the left
hex_img  single value of hex_imgs
hex_link  single value of hex_links
footer  single value of footers
footer_link  single value of footer_links
x  number, X offset, e.g. "-10" instead of -10L
y  number, Y offset
target_blank  bool, whether to add target="_blank" to the link?
hex_imgs  a character vector of logo image source, can be online or local, see details
hex_links  a character vector of links attached to each logo, if not NULL, must be the same length as hex_imgs
hex_titles  similar to hex_links, titles of each logo
footers  a character vector of footer attached to each logo
footer_links  a character vector of footer links, if not NULL, must be the same length as footers
xs  a character vector X coordinate offset value for each logo image, default -10, must be the same length as hex_imgs
ys  Y coordinates offset, must be the same length as xs, default -20

Details

The image in each hexagon is resized to the same size as the hex border and then enlarged 125%. You may want to use x, y offset value to change the image position.

If your image source is local, you need to add your local directory to the shiny server, e.g. addResourcePath("sps", "www"). This example add www folder under my current working directory as sps to the server. Then you can access my images by hex_imgs = "sps/my_img.png".

some args in hexPanel are character vectors, use NULL for the default value. If you want to change value but not all of your logos, use "" to occupy space in the vector. e.g. I have 3 logos, but I only want to add 2 footer and only 1 footer has a link: footers = c("footer1", "footer2", ""), footer_links = c("", "https://mylink", ""). By doing so footers and footer_links has the same required length.
Value

HTML elements, tagList

Examples

```r
if(interactive()){
  ui <- fluidPage(
    hexLogo(
      "logo", "Logo",
      hex_img = "https://live.staticflickr.com/7875/46106952034_954b8775fa_b.jpg",
      hex_link = "https://www.google.com",
      footer = "Footer",
      footer_link = "https://www.google.com"
    ),
    hexLogo(
      "x", "Change X offset",
      hex_img = "https://live.staticflickr.com/7875/46106952034_954b8775fa_b.jpg",
      x = "40"
    ),
    hexLogo(
      "y", "Change Y offset",
      hex_img = "https://live.staticflickr.com/7875/46106952034_954b8775fa_b.jpg",
      y = "-60"
    ),
    hexPanel(
      "demo1", "basic panel:",
      rep("https://live.staticflickr.com/7875/46106952034_954b8775fa_b.jpg", 2)
    ),
    hexPanel(
      "demo2", "panel with links:",
      c(paste0("https://d33wubrfki0l68.cloudfront.net/",
        "2c6239d331be6d037c251c71c3902792f8c4ddd2/12f67/css/images/hex/ggplot2.png"),
        paste0("https://d33wubrfki0l68.cloudfront.net/",
        "621a9c8c5d7b47c4b6d72e8f01f28d14310e8370/193fc/css/images/hex/dplyr.png")),
      c("https://ggplot2.tidyverse.org/", "https://dplyr.tidyverse.org/"),
      c("ggplot2", "dplyr")
    ),
    hexPanel(
      "demo3", "footer with links:",
      rep("https://live.staticflickr.com/7875/46106952034_954b8775fa_b.jpg", 2),
      footers = c("hex1", "hex2"),
      footer_links = rep("https://www.google.com", 2)
    ),
    hexPanel(
      "demo4", "panel offsets",
      hex_imgs = rep("https://live.staticflickr.com/7875/46106952034_954b8775fa_b.jpg", 4),
      footers = paste0("hex", 1:4),
      ys = seq(-20, -50, by = -10),
      xs = seq(20, 50, by = 10)
  )
}
```
server <- function(input, output, session) {
  shinyApp(ui, server)
}

## hrefTab

*Display a list of links in a row of buttons*

### Description

`hrefTab` creates a small section of link buttons.

### Usage

```r
hrefTab(
  label_texts,  # individual tab labels
  hrefs,       # individual tab links
  Id = NULL,   # optional element ID
  title = "A list of tabs", # element title
  title_color = "#0275d8", # title color
  bg_colors = "#337ab7", # individual tab button background color, either 1 value to apply for all of them or specify for each of them in a vector
  text_colors = "white", # individual tab button text color, either 1 value to apply for all of them or specify for each of them in a vector
  target_blank = FALSE,  # bool, whether to add target="_blank" to the link?
  ...                   # other arguments to be passed to the html element
)
```

### Arguments

- `label_texts`: individual tab labels
- `hrefs`: individual tab links
- `Id`: optional element ID
- `title`: element title
- `title_color`: title color
- `bg_colors`: individual tab button background color, either 1 value to apply for all of them or specify for each of them in a vector
- `text_colors`: individual tab button text color, either 1 value to apply for all of them or specify for each of them in a vector
- `target_blank`: bool, whether to add target="_blank" to the link?
- `...`: other arguments to be passed to the html element
Details

1. label_texts, hrefs must be the same length
2. If more than one value is provided for bg_colors or/and text_colors, the length of these 2 vectors must be the same as label_texts
3. Use "" to occupy the space if you do not want a label contains a link, e.g hrefs = c("https://google.com/", "", "")
4. If a label does not have a link, you cannot click it and there is no hovering effects.

Value

a Shiny component

Examples

```r
if(interactive()){
  ui <- fluidPage(
    hrefTab(
      title = "Default",
      label_texts = c("Bar Plot", "PCA Plot", "Scatter Plot"),
      hrefs = c("https://google.com/", "", "")
    ),
    hrefTab(
      title = "Different background",
      label_texts = c("Bar Plot", "PCA Plot", "Scatter Plot"),
      hrefs = c("https://google.com/", "", ""),
      bg_colors = c("#eee", "orange", "green")
    ),
    hrefTab(
      title = "Different background and text colors",
      label_texts = c("Bar Plot", "Disabled", "Scatter Plot"),
      hrefs = c("https://google.com/", "", ""),
      bg_colors = c("green", "#eee", "orange"),
      text_colors = c("#caffc1", "black", "blue")
    )
  )

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

  }

  shinyApp(ui, server)
}
```
**hrefTable**

**Description**

creates a table in Shiny which the cells are hyper reference (links) buttons. This function is similar to **hrefTab**, but that function only creates a single row of link buttons, and this function creates a table of rows.

The table has two columns, the first column is row names, second column is different link buttons.

**Usage**

```r
hrefTable(
  item_titles,
  item_labels,
  item_hrefs,
  item_title_colors = "#0275d8",
  item_bg_colors = "#337ab7",
  item_text_colors = "white",
  Id = NULL,
  first_col_name = "Category",
  second_col_name = "Options",
  title = "A Table buttons with links",
  title_color = "#0275d8",
  target_blank = FALSE,
  ...
)
```

**Arguments**

- `item_titles` vector of strings, a vector of titles for table row names
- `item_labels` list, a list of character vectors to specify button labels in each table row, one vector per row
- `item_hrefs` list, a list of character vectors to specify button hrefs links in each table row, one vector per row
- `item_title_colors` a single character value or a character vector to specify button title text colors of each row name
- `item_bg_colors` a single character value or a list, a list of character vectors to specify button background colors in each table row, one vector per row
- `item_text_colors` a single character value or a list, a list of character vectors to specify button text colors in each table row, one vector per row
- `Id` optional ID
- `first_col_name` first column name
- `second_col_name` second column name
- `title` title of this table
- `title_color` table title color
- `target_blank` bool, whether to add target="_blank" to the link?
- `...` other HTML param you want to pass to the table
Details

1. `item_titles`, `item_labels`, `item_hrefs` must have the same length. Each vector in `item_labels`, `item_hrefs` must also have the same length. For example, if we want to make a table of two rows, the first row has 1 cell and the second row has 2 cells:

```r
hrefTable(
  item_titles = c("row 1", "row 2"),
  item_labels = list(c("cell 1"), c("cell 1", "cell 2")),
  item_hrefs = list(c("link1"), c("link1", "link2"))
)
```

1. If `item_title_colors`, `item_text_colors` are given more than one value, the list must have the same length as `item_titles`, and length of each vector in the list must match the vector in `item_labels` in the same order.
2. If `item_title_colors` is given more than one value, the vector must have the same length as `item_titles`.
3. Use `""` to occupy the space if you do not want a label contains a link, e.g `item_hrefs = list(c("https://www.google.com/"), c("",""))`
4. If a label does not have a link, you cannot click it and there is no hovering effects.

Value

HTML elements

Examples

```r
if(interactive()){
  ui <- fluidPage(
    hrefTable(
      title = "default",
      item_titles = c("workflow 1", "unclickable"),
      item_labels = list(c("tab 1"), c("tab 3", "tab 4")),
      item_hrefs = list(c("https://www.google.com/"), c("",""))
    ),
    hrefTable(
      title = "Change button color and text color",
      item_titles = c("workflow 1", "No links"),
      item_labels = list(c("tab 1"), c("tab 3", "tab 4")),
      item_hrefs = list(c("https://www.google.com/"), c("","")),
      item_bg_colors = list(c("blue"), c("red", "orange")),
      item_text_colors = list(c("black"), c("yellow", "green"))
    ),
    hrefTable(
      title = "Change row name colors and width",
      item_titles = c("Green", "Red", "Orange"),
      item_labels = list(c("tab 1"), c("tab 3", "tab 4"), c("tab 5", "tab 6", "tab 7")),
      item_hrefs = list(c("https://www.google.com/"),
                        c("",""),
                        c("https://www.google.com/", "https://www.google.com/", ""))
    )
  )
```

### incRv

**In-line numeric operation for reactiveVal**

#### Description

In-place operations like `i += 1`, `i -= 1` is not support in R. These functions implement these operations in R. This set of functions will apply this kind of operations on `shiny::reactiveVal` objects.

#### Usage

```r
incRv(react, value = 1)
multRv(react, value = 2)
diviRv(react, value = 2)
```

#### Arguments

- **react**: reactiveVal object, when it is called, should return an numeric object
- **value**: the numeric value to do the operation on react

#### Details

- `incRv(i)` is the same as `i <- i + 1`. `incRv(i, -1)` is the same as `i <- i - 1`. `multRv(i)` is the same as `i <- i * 2`. `diviRv(i)` is the same as `i <- i / 2`.

#### Value

No return, will directly change the reactiveVal object provided to the react argument

#### See Also

If you want `shiny::reactiveValues` version of these operators or just normal numeric objects, use `spsUtil::inc`, `spsUtil::mult`, and `spsUtil::divi`. 
Examples

reactiveConsole(TRUE)
rv <- reactiveVal(0)
incRv(rv) # add 1
rv()
incRv(rv) # add 1
rv()
incRv(rv, -1) # minus 1
rv()
incRv(rv, -1) # minus 1
rv()
rv2 <- reactiveVal(1)
multRv(rv2) # times 2
rv2()
multRv(rv2) # times 2
rv2()
diviRv(rv2) # divide 2
rv2()
diviRv(rv2) # divide 2
rv2()
reactiveConsole(FALSE)

# Real shiny example
if(interactive()){  
  ui <- fluidPage(
    textOutput("text"),
    actionButton("b", "increase by 1")
  )
  server <- function(input, output, session) {
    rv <- reactiveVal(0)
    observeEvent(input$b, {
      incRv(rv)
    })
    output$text <- renderText({
      rv()
    })
  }
  shinyApp(ui, server)
}

onNextInput

Wait for the next input change

Description

This is a server function that runs like a callback when the next time any input value changes. This is useful for to watch dynamically added components from the server and then do something. For example, loading a shiny module UI from server by renderUI and loading the shiny module server from server by moduleServer. Loading the server must wait until renderUI is finished. However, in shiny renderUI is asynchronous. It means moduleServer is immediately executed
after renderUI. The result is module’s server part cannot find the UI, because it is still updating.
This function is hack to solve this problem by waiting for the next input settlement operation called
from Shiny javascript to R so one can start other server actions.

Usage

onNextInput(expr, session = getDefaultReactiveDomain())

Arguments

expr code expression, wrap inside {}

session shiny session

Details

Common usage:
This function adds a on.exit statement to the parent observer, renderXX, and other reactive
events, so make sure you use them inside these functions instead of plain server.

server = function(input, output, session) {
# ok
output$someID <- renderUI({
onNextInput({...})
div(...)
})

# following is not ok
onNextInput({...})
}

About this function:
This function fixes the issue in shiny #3348. Until there is an official support for this feature, this
function is useful.

Value

an observeEvent that runs only one time to watch for the next input change.

Examples

if(interactive()){  
library(shiny)

# Simple example
ui <- fluidPage(
uoutput("someui")
)
server <- function(input, output, session) {
output$someui <- renderUI({
    # we update the text of new rendered text input to 3 random letters
})

uiOutput("someui")

uiOutput("someui")

uiOutput("someui")
# after `textInput` is displayed, and it only works for one time.
onNextInput({
  updateTextInput(inputId = "mytext", value = paste0(sample(letters, 3), collapse = ""))
})
textInput("mytext", "some text")
# if you directly have update event like following line, it won't work
# updateTextInput(inputId = "mytext", value = paste0(sample(letters, 3), collapse = ""))
}
shinyApp(ui, server)

# complex example with modules
modUI <- function(id) {
  ns <- NS(id)
  textInput(ns("mytext"), "some text")
}
modServer = function(id) {
  moduleServer(
    id,
    function(input, output, session) {
      updateTextInput(inputId = "mytext", value = paste0(sample(letters, 3), collapse = ""))
    }
  )
}
u = fluidPage(
  actionButton("a", "load module UI"),
  uiOutput("mod_container")
)
server = function(input, output, session) {
  # everytime you click, render a new module UI and update the text value
  # immediately
  observeEvent(input$a, {
    output$mod_container <- renderUI({
      onNextInput(modServer("mod"))
      modUI("mod")
    })
  })
  # Without `onNextInput`, module server call will not work
  # uncomment below and, comment `onNextInput` line to see the difference
  # modServer("mod")
}
shinyApp(ui, server)

---

**pgPaneUI**

A draggable progress panel
Description

Creates a panel that displays multiple progress items. Use `pgPaneUI` on UI side and use `pgPaneUpdate` to update it.

A overall progress is automatically calculated on the bottom.

Usage

```r
pgPaneUI(
  pane_id,
  titles,
  pg_ids,
  title_main = NULL,
  opened = FALSE,
  top = "3%",
  right = "2%"
)

pgPaneUpdate(pane_id, pg_id, value, session = getDefaultReactiveDomain())
```

Arguments

- **pane_id**: Progress panel main ID, use `ns` wrap it on `pgPaneUI` but not on `pgPaneUpdate` if using shiny module
- **titles**: labels to display for each progress, must have the same length as `pg_ids`
- **pg_ids**: a character vector of IDs for each progress. Don’t forget to use `ns` wrap each ID.
- **title_main**: If not specified and `pane_id` contains `plot`, title will be `Plot Prepare`; has `df` will be `Data Prepare`, if neither will be "Progress"
- **opened**: bool, if this panel is opened at start
- **top**: css style off set to the current window top
- **right**: css style off set to the current window right
- **pg_id**: a character string of ID indicating which progress within this panel you want to update. Do not use `ns(pg_id)` to wrap it on server
- **value**: 0-100 number to update the progress you use `pg_id` to choose
- **session**: current shiny session

Value

returns HTML elements

Examples

```r
if(interactive()){
  # try to slide c under 0
  ui <- fluidPage(
    h4("Use your mouse to drag it"),
```
renderDesc

Render some collapsible markdown text

Description

write some text in markdown format and it will help you render to markdown, use shiny::markdown but it is collapsible.

Usage

renderDesc(id, desc)

Arguments

id element ID
desc one character string in markdown format
shinyCatch

Value

HTML elements

Examples

```r
if(interactive()){
  desc <- 
    "
    # Some desc
    - xxxx
    - bbbb

    This is a [link](https://www.google.com/).

    `Some other things`
    > other markdown things

    1. aaa
    2. bbb
    3. ccc

    ui <- fluidPage(
      renderDesc(id = "desc", desc),
    )

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

    shinyApp(ui, server)
}
```

shinyCatch  

Shiny exception handling

Description

Exception in Shiny apps can crash the app. Most time we don’t want the app to crash but just stop this code block, inform users and continue with other code blocks. This function is designed to handle these issues.

Usage

```r
shinyCatch(
  expr,
  position = "bottom-right",
  blocking_level = "none",
  shiny = TRUE,
  prefix = "SPS",
)```
trace_back = spsOption("traceback")
)

Arguments

expr              expression
position          client side message bar position, one of: c("top-right", "top-center", "top-left", "top-full-width", "bottom-right", "bottom-center", "bottom-left", "bottom-full-width").
blocking_level    what level you want to block the execution, one of "error", "warning", "message", default is "none", do not block following code execution.
shiny             bool, only show message on console log but not in Shiny app when it is FALSE. Useful if you want to keep the exception only to the server and hide from your users. You do not need to set it to FALSE when purely work outside shiny, it will automatically detect if you are working in a Shiny environment or not.
prefix            character, what prefix to display on console for the log, e.g. for error, the default will be displayed as "SPS-ERROR". You can make your own prefix, like prefix = "MY", then, it will be "MY-ERROR". Use "" if you do not want any prefix, like prefix = "", then, it will just be "ERROR". multiple levels
trace_back        bool, added since spsComps 0.2, if the expression is blocked or has errors, call the full trace back? It will display called functions and code source file and line number if possible. Default follows the SPS spsOption("traceback") setting. You can set it by running spsOption("traceback", TRUE). If you do not set it, it will be FALSE. or you can just manually set it for each individual shinyCatch call shinyCatch({...}, trace_back = TRUE).

Details

Blocking:
- The blocking works similar to shiny’s shiny::req() and shiny::validate(). If anything inside fails, it will block the rest of the code in your reactive expression domain.
- It will show error, warning, message by a toaster bar on client end and also log the text on server console depending on the blocking_level (dual-end logging).
- If blocks at error level, function will be stopped and other code in the same reactive context will be blocked.
- If blocks at warning level, function will be stopped and other code in the same reactive context will be blocked.
- If blocks at message level, function will be stopped.
- message level blocks all 3 levels.
- If blocking_level is other than these 3, no exceptions will be block, and if there is any error, NULL will return and following code will continue to run.

To use it:
Since spsComps 0.3.1 to have the message displayed on shiny UI, you don’t need to attach the dependencies manually by adding spsDepend("shinyCatch") or spsDepend("toastr") (old name) on UI. This becomes optional, only in the case that automatic attachment is not working.

Display:
Messages will be displayed for 3 seconds, and 5s for warnings. Errors will never go away on UI unless users’ mouse hover on the bar or manually click it.
**environment:**
shinyCatch uses the same environment as where it is called, it means if you assign a variable inside the expression, you can still get it from outside the shinyCatch, see examples.

**Value**

see description and details

**Examples**

```r
if(interactive()){
  ui <- fluidPage(
    spsDepend("shinyCatch"), # optional
    h4("Run this example on your own computer to better understand exception catch and dual-end logging", class = "text-center"),
    column(
      6,
      actionButton("btn1","error and blocking"),
      actionButton("btn2","error no blocking"),
      actionButton("btn3","warning but still returns value"),
      actionButton("btn4","warning but blocking returns"),
      actionButton("btn5","message"),
    ),
    column(
      6,
    verbatimTextOutput("text")
  )
  server <- function(input, output, session) {
    fn_warning <- function() {
      warning("this is a warning!")
      return("warning returns")
    }
    observeEvent(input$btn1, {
      shinyCatch(stop("error with blocking"), blocking_level = "error")
      output$text <- renderPrint("You shouldn't see me")
    })
    observeEvent(input$btn2, {
      shinyCatch(stop("error without blocking"))
      output$text <- renderPrint("I am not blocked by error")
    })
    observeEvent(input$btn3, {
      return_value <- shinyCatch(fn_warning())
      output$text <- renderPrint("warning and blocked")
    })
    observeEvent(input$btn4, {
      return_value <- shinyCatch(fn_warning(), blocking_level = "warning")
      print(return_value)
      output$text <- renderPrint("other things")
    })
    observeEvent(input$btn5, {
      shinyCatch(message("some message"))
    }
  }
```
shinyCheckPkg <- renderPrint("some message")
}
shinyApp(ui, server)
}

# outside shiny examples
shinyCatch(message("this message"))
try({shinyCatch(stop("this error")); "no block"}, silent = TRUE)
try({shinyCatch(stop("this error"), blocking_level = "error"); "blocked"}, silent = TRUE)

# get variable from outside
shinyCatch({my_val <- 123})
my_val

shinyCheckPkg  Shiny package checker

Description
A server end function to check package namespace for some required packages of users’ environment. If all packages are installed, a successful message will be displayed on the bottom-right. If not, pop up a message box in shiny to tell users how to install the missing packages.
This is useful when some of packages are required by a shiny app. Before running into that part of code, using this function to check the required package and pop up warnings will prevent app to crash.

Usage
shinyCheckPkg(
  session,
  cran_pkg = NULL,
  bioc_pkg = NULL,
  github = NULL,
  quietly = FALSE
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>shiny session</td>
</tr>
<tr>
<td>cran_pkg</td>
<td>a vector of package names</td>
</tr>
<tr>
<td>bioc_pkg</td>
<td>a vector of package names</td>
</tr>
<tr>
<td>github</td>
<td>a vector of github packages, github package must use the format of &quot;github user name/ repository name&quot;, eg. c(&quot;user1/pkg1&quot;, &quot;user2/pkg2&quot;)</td>
</tr>
<tr>
<td>quietly</td>
<td>bool, should warning messages be suppressed?</td>
</tr>
</tbody>
</table>

Value
TRUE if pass, sweet alert massage and FALSE if fail
Examples

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

  ui <- fluidPage(
    tags$label('Check if package "pkg1", "pkg2", "bioxxx",
               github package "user1/pkg1" are installed'), br(),
    actionButton("check_random_pkg", "check random_pkg"), br(),
    tags$label('We can combine `spsValidate` to block server code to prevent
               crash if some packages are not installed.'), br(),
    tags$label('If "shiny" is installed, make a plot.'), br(),
    actionButton("check_shiny", "check shiny"), br(),
    tags$label('If "ggplot99" is installed, make a plot.'), br(),
    actionButton("check_gg99", "check ggplot99"), br(),
    plotOutput("plot_pkg")
  )

  server <- function(input, output, session) {
    observeEvent(input$check_random_pkg, {
      shinyCheckPkg(session, cran_pkg = c("pkg1", "pkg2"),
                   bioc_pkg = "bioxxx", github = "user1/pkg1")
    })
    observeEvent(input$check_shiny, {
      spsValidate(Verbose = FALSE, {
        if(!shinyCheckPkg(session, cran_pkg = c("shiny"))) stop("Install packages")
      })
      output$plot_pkg <- renderPlot(plot(1))
    })
    observeEvent(input$check_gg99, {
      spsValidate(
        if(!shinyCheckPkg(session, cran_pkg = c("ggplot99"))) stop("Install packages")
      )
      output$plot_pkg <- renderPlot(plot(99))
    })
  }

  shinyApp(ui, server)
}
```

**spsCodeBtn**

Display your code in a bootstrap modal or collapse

**Description**

Developers often want to show their code in a shiny app. This function creates a button that, when clicked, a modal or collapse hidden element will show up to display your code.
Usage

spsCodeBtn(
  id,
  code,
  language = "r",
  label = "",
  title = "Code to Reproduce",
  show_span = FALSE,
  tool_tip = "Show Code",
  placement = "bottom",
  btn_icon = icon("code"),
  display = c("modal", "collapse"),
  size = c("large", "medium", "small"),
  color = "black",
  shape = c("rect", "circular"),
  ...
)

Arguments

  id  element ID
  code code you want to display, in a character string or vector.
  language string, what programming language is the code, use shinyAce::getAceModes() to see options
  label string, label to display on the button
  title string, title of the modal or collapse
  show_span bool, use the <span> tag to show a little label of the left of the button? The span text will use text from tool_tip
  tool_tip string, what tooltip to display when hover on the button
  placement string, where to display the tooltip
  btn_icon icon, shiny::icon(), icon of the button
  display string, one of "modal", "collapse"
  size string, one of "large", "medium", "small", only works for modal
  color string, color of the button
  shape string, shape of the button, one of "rect", "circular",
  ... other args pass to the shiny::actionButton

Details

1. The modal or collapse has an ID, the ID is your button ID + "-modal" or "-collapse", like "my_button-modal"

2. You could update the code inside the collapse use shinyAce::updateAceEditor on server, the code block ID is button ID + "-ace", like "my_button-ace". See examples.
Value

a shiny tagList

Examples

```r
if(interactive()){
  library(shiny)
  my_code <-
    # load package and data
    library(ggplot2)
    data(mpg, package="ggplot2")
    # mpg <- read.csv("http://goo.gl/uEeRGu")

    # Scatterplot
    theme_set(theme_bw()) # pre-set the bw theme.
    g <- ggplot(mpg, aes(cty, hwy))
    g + geom_jitter(width = .5, size=1) +
    labs(subtitle="mpg: city vs highway mileage",
         y="hwy",
         x="cty",
         title="Jittered Points")

  html_code <-
  "<!DOCTYPE html>
  <html>
  <body>
  <h2>ABC</h2>
  <p id="demo">Some HTML</p>
  </body>
  </html>
 ",

  ui <- fluidPage(
    fluidRow(
      column(6,
        h3("Display by modal"),
        column(6, h4("default"),
          spsCodeBtn(id = "a", my_code)
        ),
        column(6, h4("change color and shape"),
          spsCodeBtn(
            id = "b", c(my_code, my_code),
            color = "red", shape = "circular")
        )
      )
    )
  )
}
```
column(  
6,  
h3("Display by collapse"),  
  column(  
6, h4("collapse"),  
     spsCodeBtn(id = "c", my_code, display = "collapse")  
  ),  
  column(  
6, h4("different programming language"),  
     spsCodeBtn(  
       id = "d", html_code,  
       language = "html", display = "collapse")  
  )  
),  
),  
fluidRow(  
  column(  
6,  
  h3("Update code"),  
     spsCodeBtn(  
       "update-code",  
       "# No code here",  
       display = "collapse"  
     ),  
     actionButton("update", "change code in the left `spsCodeBtn`"),  
     actionButton("changeback", "change it back")  
  )  
)  
)  

server <- function(input, output, session) {  
  observeEvent(input$update, {  
    shinyAce::updateAceEditor(  
      session, editorId = "update-code-ace",  
      value = "# code has changed!\n 1+1"  
    )  
  })  
  observeEvent(input$changeback, {  
    shinyAce::updateAceEditor(  
      session, editorId = "update-code-ace",  
      value = "# No code here"  
    )  
  })  
  shinyApp(ui, server)  
}  

spsDepend  

Add commonly used HTML dependencies
spsDepend

Description
Add dependencies for some server end functions. For most UI functions, the dependency has been automatically attached for you when you call the function. Most server functions will also attach the dependency for you automatically too. However, a few server functions have to append the dependency before app start like addLoader. So you would need to call in this function somewhere in your UI. Read help of each function for details.

Usage
spsDepend(dep = "", js = TRUE, css = TRUE, listing = TRUE)

Arguments
- dep: dependency names, see details
- js: bool, use only javascript from this resource if there are both js and css files?
- css: bool, use only CSS from this resource if there are both js and css files?
- listing: bool, if your dep is invalid, list all options? FALSE will mute it.

Details
For dep, current options are:
- basic: spsComps basic css and js
- update_pg: spsComps pgPaneUpdate function required, js and css
- update_timeline: spsComps spsTimeline function required, js only
- font-awesome: font-awesome, css only
- toastr: comes from shinytoastr package, toastr.js, css and js
- pop-tip: enable enhanced bootstrap popover and tips, required for bsHoverPopover function. js only
- gotop: required by spsGoTop function. js and css
- animation: required for animation related functions to add animations for icons and other elements, like animateServer. js and css
- css-loader: required for loader functions, like addLoader. js and css
- sweetalert2: sweetalert2.js, required by shinyCheckPkg. js only

Value
htmltools::htmlDependency object

Examples
# list all options
spsDepend(""")
# try some options
spsDepend("basic")
spsDepend("font-awesome")
# Then add it to your shiny app
if(interactive()){
    library(shiny)

    ui <- fluidPage(
        tags$i(class = "fa fa-house"),
        spsDepend("font-awesome")
    )

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

    shinyApp(ui, server)
}

---

spsGoTop  

**Go top button**

**Description**

add a go top button on your shiny app. When the user clicks the button, scroll the window all the way to the top. Just add this function anywhere in you UI.

**Usage**

spsGoTop(  
    id = "gotop",
    icon = NULL,
    right = "1rem",
    bottom = "10rem",
    color = "#337ab7"
)

**Arguments**

- **id**  
  element ID
- **icon**  
  shiny::icon if you do not want to use the default rocket image
- **right**  
  character string, css style, the button’s position to window right
- **bottom**  
  character string, css style, the button’s position to window bottom
- **color**  
  color of the icon.

**Details**

The button hides if you are on very top of the page. If you scroll down 50px, this button will appear.
spsHr

Value

a shiny component

Examples

```r
if(interactive()){
  library(shiny)
  ui <- fluidPage(
    h1("Scroll the page..."),
    lapply(1: 100, function(x) br()),
    spsGoTop("default"),
    spsGoTop("mid", right = "50\%", bottom= "50\%", icon = icon("house"), color = "red"),
    spsGoTop("up", right = "95\%", bottom= "95\%", icon = icon("arrow-up"), color = "green")
  )
}
```

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

shinyApp(ui, server)
```

---

**spsHr**

Create a horizontal line

---

**Description**

Create a horizontal line of your choice

**Usage**

```r
spsHr(
  status = "info",
  width = 0.5,
  other_color = NULL,
  type = "solid",
  opacity = 1
)
```

**Arguments**

- **status** string, one of "primary", "info", "success", "warning", "danger". This determines the color of the line.
- **width** numeric, how wide should the line be, a number larger than 0
- **other_color** string, if you do not like the default 5 status colors, specify a valid CSS color here. If this is provided status will be ignored.
**spsTimeline**

A shiny timeline component

**Description**

This timeline is horizontal, use `spsTimeline` to define it and use `updateSpsTimeline` on server to update it.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>type</strong></td>
<td>string, one of &quot;solid&quot;, &quot;dotted&quot;, &quot;dashed&quot;, &quot;double&quot;, &quot;groove&quot;, &quot;ridge&quot;, &quot;inset&quot;, &quot;outset&quot;</td>
</tr>
<tr>
<td><strong>opacity</strong></td>
<td>numeric, a number larger than 0 smaller than 1</td>
</tr>
</tbody>
</table>

**Details**

Read more about type here: https://www.w3schools.com/css/css_border.asp

**Value**

HTML `<hr>` element

**Examples**

```r
if(interactive()) {
  library(shiny)
  library(magrittr)
  ui <- fluidPage(
    tags$b("Different status"),
    spsHr("info"),
    spsHr("primary"),
    spsHr("success"),
    spsHr("warning"),
    spsHr("danger"),
    tags$b("custom color"),
    spsHr(other_color = "purple"),
    spsHr(other_color = "pink"),
    tags$b("Different width"),
    lapply(1:5, function(x) spsHr(width = x)),
    tags$b("Different type"),
    c("solid", "dotted", "dashed", "double", "groove", "ridge", "inset", "outset") %>%
      lapply(function(x) spsHr(type = x, width = 3)),
    tags$b("Different opacity"),
    lapply(seq(0.2, 1, 0.2), function(x) spsHr(opacity = x))
  )
  server <- function(input, output, session) {}
  shinyApp(ui, server)
}
```
spsTimeline

Usage

spsTimeline(id, up_labels, down_labels, icons, completes)

updateSpsTimeline(
  session,
  id,
  item_no,
  complete = TRUE,
  up_label = NULL,
  down_label = NULL
)

Arguments

id html ID of the timeline if you are using shiny modules: use namespace function to create the ID but DO NOT use namespace function on server.

up_labels a vector of strings, text you want to display on top of each timeline item, usually like year number. If you do not want any text for a certain items, use "" to occupy the space.

down_labels a vector of strings, text you want to display at the bottom of each timeline item.

icons a list of icon objects. If you do not want an icon for certain items, use div() to occupy the space.

completes a vector of TRUE or FALSE, indicating if the items are completed or not. Completed items will become green.

session current shiny session

item_no integer, which item number counting from left to right you want to update

complete bool, is this item completed or not

up_label the item_no associated up label to update

down_label the item_no associated down label to update

Details

up_labels, down_labels, icons, completes must have the same length.

Value

returns a shiny component

Examples

if(interactive()){
  ui <- fluidPage(
    column(6,
      spsTimeline(
        "b",
        up_labels = c("2000", "2001"),
        down_labels = c("step 1", "step 2"),
      
      )
    )
  )
}
spsTitle

Colorful title element

Description

Add a title element to UI

Usage

spsTitle(
  title,
  level = "2",
  status = "info",
  other_color = NULL,
  opacity = 1,
  ...
)

tabTitle(
  title,
  level = "2",
  status = "info",
  other_color = NULL,
  opacity = 1,
  ...
)
spsValidate

Arguments

title string, title text
level string, level of the title, the larger, the bigger, one of "1", "2", "3", "4", "5", "6"
status string, one of "primary", "info", "success", "warning", "danger". This determines the color of the line.
other_color string, if you do not like the default 5 status colors, specify a valid CSS color here. If this is provided, status will be ignored.
opacity numeric, a number larger than 0 smaller than 1
... other attributes and children add to this element

Value

returns a shiny tag

Examples

if(interactive()) {
  library(shiny)
  library(magrittr)
  ui <- fluidPage(
    tags$b("Different status"),
    c("primary", "info", "success", "warning", "danger") %>%
    lapply(function(x) spsTitle(x, "4", status = x)),
    tags$b("custom color"),
    spsTitle("purple", "4", other_color = "purple"),
    spsTitle("pink", "4", other_color = "pink"),
    tags$b("Different levels"),
    lapply(as.character(1:6), function(x) spsTitle(paste0("H", x), x)),
    tags$b("Different opacity"),
    lapply(seq(0.2, 1, 0.2), function(x) spsTitle(as.character(x), opacity = x))
  )
  server <- function(input, output, session) {
  }
  shinyApp(ui, server)
}

spsValidate  Validate expressions

Description

describe this function is used on server side to usually validate input dataframe or some expression. The usage is similar to shiny::validate but is not limited to shiny render functions and provides better user notification and server-end logging (dual-end logging).
Usage

spsValidate(
  expr,
  vd_name = "my validation",
  pass_msg = glue("validation: '{vd_name}' passed"),
  shiny = TRUE,
  verbose = spsOption("verbose"),
  prefix = ""
)

Arguments

  expr          the expression to validate data or other things. Use stop("your message") or
                generate some errors inside to fail the validation. If there is no error, it will
                return TRUE and display pass_msg on both console and shiny app if verbose =
                TRUE or global SPS option verbose is TRUE.
                If the expression fails, it will block the code following this function within the
                same reactive domain to continue, similar to shinyCatch().

  vd_name       validate title

  pass_msg      string, if pass, what message do you want to show

  shiny         bool, show message on console but hide from users? see shinyCatch() for
                more details

  verbose       bool, show pass message? Default follows global verbose setting, use spsU-
                til::spsOption to set up the value spsOption("verbose, TRUE") to turn on and
                spsOption("verbose, FALSE") to turn off and spsOption("verbose") to check
                current setting, see examples.

  prefix        see prefix in shinyCatch()

Details

- Since spsComps 0.3.1 to have the message displayed on shiny UI, you don’t need to attach
  the dependencies manually by adding spsDepend("spsValidate") or spsDepend("toastr")
  (old name) on UI. This becomes optional, only in the case that automatic attachment is not
  working.

Value

If expression fails, block the code following this validation function and no final return, else TRUE.

Examples

if(interactive()){
  ui <- fluidPage(
    spsDepend("spsValidate"), # optional
    column(
      4,
      h3("click below to make the plot"),
      p("this button will succeed, verbose on"),
    )
  )
server <- function(input, output, session) {
mydata <- datasets::iris
observeEvent(input$vd1, {
spsOption("verbose", TRUE) # use global sps verbose setting
spsValidate({
  is.data.frame(mydata)
}, vd_name = "Is dataframe")
output$p1 <- renderPlot(plot(iris$Sepal.Length, iris$Sepal.Width))
})
observeEvent(input$vd2, {
spsValidate({
  is.data.frame(mydata)
},
  vd_name = "Is dataframe",
  verbose = FALSE # use in-function verbose setting
output$p2 <- renderPlot(plot(iris$Sepal.Length, iris$Sepal.Width))
})
observeEvent(input$vd3, {
spsValidate({
  is.data.frame(mydata)
  if(nrow(mydata) <= 200) stop("Input needs more than 200 rows")
})
print("other things blocked")
output$p3 <- renderPlot(plot(iris$Sepal.Length, iris$Sepal.Width))
})
observeEvent(input$vd4, {
spsValidate({
  is.data.frame(mydata)
})
if(nrow(mydata) <= 200) stop("Input needs more than 200 rows")
}, shiny = FALSE)
print("other things blocked")
output$p4 <- renderPlot(plot(iris$Sepal.Length, iris$Sepal.Width))
})
}
shinyApp(ui, server)
}

# outside shiny example
mydata2 <- list(a = 1, b = 2)
spsValidate({(mydata2)}, "Not empty")
try(spsValidate(stopifnot(is.data.frame(mydata2)), "is dataframe?"), silent = TRUE)

---

**textButton**  
Text input with an action button

---

**Description**

One kind of bootstrap3 input group: a textinput and a button attached to the end

**Usage**

```r

textButton(
  textId,
  btnId = paste0(textId, "_btn"),
  label = "",
  text_value = "",
  placeholder = "",
  tooltip = "",
  placement = "bottom",
  btn_icon = NULL,
  btn_label = "btn",
  style = "",
  ...
)
```

**Arguments**

- **textId**: the text input ID
- **btnId**: the button ID, if not specified, it is "textId" + ".btn" like, textId_btn
- **label**: label of the whole group, on the top
- **text_value**: initial value of the text input
- **placeholder**: placeholder text of the text input
- **tooltip**: a tooltip of the group
- **placement**: where should the tooltip go?
- **btn_icon**: a shiny::icon of the button
textButton

btn_label text on the button
style additional CSS style of the group
... additional args pass to the button, see shiny::actionButton

Value
a shiny input group

Examples

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

  ui <- fluidPage(
    column(
      6,
      textButton(textId = "tbtn_default", label = "default"),
      textButton(
        textId = "tbtn-icon",
        label = "change icon and color",
        btn_icon = icon("house"),
        class = "btn-warning" # pass to the button
      ),
      textButton(
        textId = "tbtn_style",
        label = "change styles",
        style = "color: red; border: 2px dashed green;"
      ),
      textButton(
        textId = "tbtn_submit",
        label = "interact with shiny server",
        btn_label = "Submit",
        placeholder = "type and submit",
        class = "btn-primary"),
      verbatimTextOutput("tbtn_submit_out")
    )
  )

  server <- function(input, output, session) {
    # watch for the button ID "tbtn_submit" + "_btn"
    observeEvent(input$tbtn_submit_btn, {
      output$tbtn_submit_out <- renderPrint(isolate(input$tbtn_submit))
    })
  }

  shinyApp(ui, server)
}
```
Description

Text input group and custom widgets append to left and right.

Usage

```r
textInputGroup(
  textId,
  label = "",
  value = "",
  placeholder = "enter text",
  left_text = NULL,
  right_text = NULL,
  style = "width: 100%;"
)
```

Arguments

- `textId` text box id
- `label` text label for this input group
- `value` default value for the text input
- `placeholder` default placeholder text for the text input if no value
- `left_text` text or icon add to the left side
- `right_text` text or icon add to the right side
- `style` additional style add to the group

Details

If no text is specified for both left and right, the return is almost identical to `clearableTextInput`.

Value

text input group component

Examples

```r
if(interactive()){
  ui <- fluidPage(
    textInputGroup("id1", "left", left_text = "a"),
    textInputGroup("id2", "right", right_text = "b"),
    textInputGroup("id3", "both", left_text = ",", right_text = ".00"),
    textInputGroup("id4", "none"),
  )
  # ui
}
```
textInputGroup("id5", "icon", left_text = icon("house")),
}

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

}

shinyApp(ui, server)
}
addLoader, 2, 51
animateAppend, 9
animateAppendNested (animateAppend), 9
animateIcon, 10
animateServer, 13, 14, 51
animateServer (animateUI), 12
animateUI, 12, 13, 14
animationRemove, 13, 14
animationRemove (animateUI), 12
bsAlert, 15
bsHoverPopover, 18, 51
bsHoverPopover (bsPopover), 16
bsPop, 18
bsPop (bsPopover), 16
bsPopover, 16, 18
bsTip, 21
bsTip (bsTooltip), 20
bsTooltip, 20, 21
clearableTextInput, 23, 62
cssLoader, 24
diviRv (incRv), 37
gallery, 27
heightMatcher, 29
hexLogo, 30
hexLogo(), 30
hexPanel (hexLogo), 30
hexPanel(), 30
hrefTab, 33, 35
hrefTable, 34
htmltools::htmlDependency, 51
incRv, 37
multRv (incRv), 37
observeEvent, 39
onNextInput, 38
pgPaneUI, 40, 41
pgPaneUpdate, 51
pgPaneUpdate (pgPaneUI), 40
renderDesc, 42
shiny::actionButton, 25, 48, 61
shiny::icon, 11, 52, 60
shiny::icon(), 48
shiny::markdown, 42
shiny::reactiveValues, 37
shiny::req(), 44
shiny::validate, 57
shiny::validate(), 44
shinyAce::getAceModes(), 48
shinyAce::updateAceEditor, 48
shinyCatch, 43
shinyCatch(), 58
shinyCheckPkg, 46, 51
spsCodeBtn, 47
spsDepend, 50
spsGoTop, 51, 52
spsHr, 53
spsTimeline, 51, 54
spsTitle, 56
spsUtil::divi, 37
spsUtil::inc, 37
spsUtil::mult, 37
spsUtil::spsOption, 58
spsValidate, 57
tabTitle (spsTitle), 56
textButton, 60
textInputGroup, 62
updateSpsTimeline (spsTimeline), 54