Package 'shinybusy'

Title Busy Indicators and Notifications for 'Shiny' Applications

Version 0.3.3

Description Add indicators (spinner, progress bar, gif) in your 'shiny'
applications to show the user that the server is busy. And other tools to let
your users know something is happening (send notifications, reports, ...).

License GPL-3

Encoding UTF-8

Imports htmltools, shiny, jsonlite, htmlwidgets

RoxygenNote 7.3.1

URL https://github.com/dreamRs/shinybusy,
https://dreamrs.github.io/shinybusy/

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

Suggests testthat, covr, knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Fanny Meyer [aut],
Victor Perrier [aut, cre],
Silex Technologies [fnd] (https://www.silex-ip.com)

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

Repository CRAN

Date/Publication 2024-03-09 11:20:02 UTC

R topics documented:

---

add_busy_bar   *Automatic busy indicator (Progress bar)*

Description

Make a progress bar appear on top of the page.

Usage

```
add_busy_bar(
  timeout = 1000,
  color = "#112446",
  centered = FALSE,
  height = "8px"
)
```

Arguments

timeout	Number of milliseconds after the server is busy to display the progress bar.
color	Progress bar color.
centered	Center the progress bar or not.
height	Height of the bar.

Examples

```
library(shiny)
library(shinybusy)

ui <- fluidPage(

  # Use this function somewhere in UI
  add_busy_bar(color = "#FF0000"),

  headerPanel('Iris k-means clustering'),

  tags$br(),
  actionButton("quick", "Quick calculation (nothing happens)"),
  actionButton("sleep", "Long calculation (progress bar on top)")
)

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

  observeEvent(input$quick, {
    Sys.sleep(0.1)
  })

  observeEvent(input$sleep, {
    Sys.sleep(5)
  })

}

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

---

add_busy_gif   *Automatic busy indicator (GIF)*

Description

Make a GIF play when server is busy and stop when idle.

Usage

```
add_busy_gif(
  src,
  timeout = 100,
 position = c("top-right", "top-left", "bottom-right", "bottom-left", "full-page",
    "free"),
  margins = c(10, 10),
  overlay_color = "rgba(0, 0, 0, 0.5)",
  overlay_css = NULL,
```

```
    height = "50px",
    width = "50px"
)
```

Arguments

src	Path to the GIF, an URL or a file in www/ folder.
timeout	Number of milliseconds after the server is busy to display the GIF.
position	Where to display the GIF: `'top-right'`, `'top-left'`, `'bottom-right'`, `'bottom-left'`, `'full-page'`.
margins	Distance from margins, a vector of length two, where first element is distance from top/bottom, second element distance from right/left.
overlay_color	Background color for the overlay if position = "full-page".
overlay_css	Additional CSS for the overlay, for example "z-index: 1000;" to make it appear above everything.
height, width	Height and width of the spinner, default to '50px' for both, must be specified.

Value

An HTML tag that should be used in UI.

Examples

```
library(shiny)
library(shinybusy)

ui <- fluidPage(

  # Use this function somewhere in UI
  # with navBarPage use the "header" argument
  add_busy_gif(
    src = "https://jeroen.github.io/images/banana.gif",
    height = 70, width = 70
  ),

  actionButton("sleep", "Long calculation")
)

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

  observeEvent(input$sleep, {
    Sys.sleep(5)
  })

}

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

---

add_busy_spinner *Automatic busy indicator (spinner)*

Description

Add a spinner in an application each time the server take more 100 milliseconds to respond.

Usage

```
add_busy_spinner(
  spin = "double-bounce",
  color = "#112446",
  timeout = 100,
 position = c("top-right", "top-left", "bottom-right", "bottom-left", "full-page"),
  onstart = TRUE,
  margins = c(10, 10),
  height = "50px",
  width = "50px"
)
```

Arguments

spin	Style of the spinner, see [spin_epic]() or [spin_kit]() for possible choices. Note that for spin_epic, height and width are ignored.
color	Color for the spinner, in a valid CSS format.
timeout	Number of milliseconds after the server is busy to display the spinner.
position	Where to display the spinner: `'top-right'`, `'top-left'`, `'bottom-right'`, `'bottom-left'`, `'full-page'`.
onstart	Logical, display the spinner when the application starts ?
margins	Distance from margins, a vector of length two, where first element is distance from top/bottom, second element distance from right/left.
height, width	Height and width of the spinner, default to `'50px'` for both, must be specified.

Examples

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

  ui <- fluidPage(

    # Use this function somewhere in UI
    add_busy_spinner(spin = "cube-grid"),
    # or use a different spinner
    # add_busy_spinner(spin = "radar", margins = c(10, 20)),
```

```
        headerPanel('Iris k-means clustering'),

        sidebarLayout(
          sidebarPanel(
            selectInput('xcol', 'X Variable', names(iris)),
            selectInput('ycol', 'Y Variable', names(iris),
                        selected=names(iris)[[2]]),
            numericInput('clusters', 'Cluster count', 3,
                         min = 1, max = 9),
            actionButton("sleep", "Long calculation")
          ),
          mainPanel(
            plotOutput('plot1')
          )
        )
      )

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

        selectedData <- reactive({
          iris[, c(input$xcol, input$ycol)]
        })

        clusters <- reactive({
          kmeans(selectedData(), input$clusters)
        })

        output$plot1 <- renderPlot({
          palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
                    "#FF7F00", "#FFFF33", "#A65628", "#F781BF",
                    "#999999"))

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

        observeEvent(input$sleep, {
          Sys.sleep(5)
        })

      }

      shinyApp(ui, server)
    }
```

---

add_loading_state   *Loading state for Shiny Outputs*

Description

Call this function once in your UI to automatically add loading indicators to several outputs when they are being regenerated.

Usage

```
add_loading_state(
  selector,
  spinner = c("standard", "hourglass", "circle", "arrows", "dots", "pulse"),
  text = NULL,
  timeout = 600,
  svgColor = "#383838",
  svgSize = "45px",
  messageColor = "#383838",
  messageFontSize = "14px",
  backgroundColor = "rgba(255,255,255,0.9)",
  ...
)
```

Arguments

selector	CSS selector to match outputs, for example use ".shiny-plot-output" to select all shiny::plotOutput() in your application, or "#my_chart" to select a specific output. You can use a vector to select multiple outputs.
spinner	Name of the spinner to use.
text	An optional text to be displayed under the spinner.
timeout	In milliseconds, time after the output has been regenerated for removing the loading state.
svgColor	Changes the SVG Icons color. You can use HEX, RGB or RGBA.
svgSize	Changes the SVG Icons width and height.
messageColor	Changes the color of the message text.
messageFontSize	
	Changes the font-size of the message text.
backgroundColor	
	Changes the background color. You can use HEX, RGB or RGBA.
...	Other options passed to the JavaScript method, see this link for all available options.

Value

An HTML tag that you can use in Shiny UI.

Note

This function is experimental, if you encounter bugs or bad behavior, please report issue here.

Examples

```
library(shinybusy)
library(shiny)

ui <- fluidPage(

  # Use once in UI
  add_loading_state(
    ".shiny-plot-output",
    text = "Please wait...",
    svgColor = "steelblue"
  ),

  tags$h3("Loading state"),
  actionButton("refresh", "Refresh charts"),
  actionButton("modal", "Open modal window"),

  fluidRow(
    column(
      width = 6,
      plotOutput(outputId = "plot1")
    ),
    column(
      width = 6,
      plotOutput(outputId = "plot2")
    )
  )

)

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

  output$plot1 <- renderPlot({
    input$refresh
    if (input$refresh > 0) {
      Sys.sleep(2)
    }
    barplot(table(floor(runif(100) * 6)))
  })

  output$plot2 <- renderPlot({
    input$refresh
    if (input$refresh > 0) {
      Sys.sleep(2)
    }
    plot(rnorm(50), rnorm(50))
  })


  observeEvent(input$modal, {
    showModal(modalDialog(
      title = "Works in modal too",
```

```
      actionButton("refresh2", "Refresh chart"),
      plotOutput(outputId = "plot3")
    ))
  })

  output$plot3 <- renderPlot({
    input$refresh2
    if (input$refresh2 > 0) {
      Sys.sleep(2)
    }
    hist(rnorm(500))
  })

}

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

---

block   *Block / unblock an UI element*

Description

Block / unblock an UI element

Usage

```
block(
  id,
  text = "Loading",
  type = c("standard", "hourglass", "circle", "arrows", "dots", "pulse"),
  ...,
  selector = NULL,
  session = shiny::getDefaultReactiveDomain()
)

unblock(
  id,
  selector = NULL,
  timeout = 0,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

id	Id of the element to block, for exemple an `outputId`.
text	Text displayed below the blocking indicator. Must be a single character string.

type	Type of blocking indicator.
...	Other configuration option, see online documentation.
selector	Custom CSS selector, if used id is ignored.
session	Default Shiny session.
timeout	Unblock after a delay.

Value

No value.

Examples

```
library(shinybusy)
library(shiny)

ui <- fluidPage(
  tags$h3("Block Output"),
  fluidRow(
    column(
      width = 6,
      plotOutput(outputId = "plot1"),
      actionButton("block_manually", "Block / unblock")
    ),
    column(
      width = 6,
      plotOutput(outputId = "plot2"),
      actionButton("block_reac", "Block when calculating in reactive()")
    )
  )
)

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

  output$plot1 <- renderPlot({
    barplot(table(floor(runif(100) * 6)))
  })

  observeEvent(input$block_manually, {
    if (input$block_manually %% 2 == 1) {
      block(id = "plot1", type = "pulse", svgColor = "#5ea4d8")
    } else {
      unblock(id = "plot1")
    }
  })

  data_r <- reactive({
    input$block_reac
    block(
      id = "plot2",
      type = "circle",
      text = "Calculating, please wait...",
```

```
      messageColor = "#FFF",
      svgColor = "#FFF",
      backgroundColor = "#5ea4d8"
    )
    Sys.sleep(3)
    data <- data.frame(x = rnorm(50), y = rnorm(50))
    unblock(id = "plot2", timeout = 300)
    return(data)
  })

  output$plot2 <- renderPlot({
    plot(data_r())
  })

}

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

---

block_output	*Block an output*

Description

Block an output until it is recalculated.

Usage

```
block_output(
  output,
  type = c("standard", "hourglass", "circle", "arrows", "dots", "pulse"),
  text = "Loading...",
  timeout = 0,
  ...,
  minHeight = NULL
)
```

Arguments

output	An output element.
type	Type of blocking indicator.
text	Text displayed below the blocking indicator. Must be a single character string.
timeout	Unblock after a delay.
...	Other configuration option, see online documentation.
minHeight	Set a minimal height to the ouput element.

Value

A shiny.tag or shiny.tag.list object (the output element modified).

Examples

```
library(shinybusy)
library(shiny)

ui <- fluidPage(
  tags$h3("Block Output from UI"),
  actionButton("refresh", "Refresh outputs"),
  actionButton("open_modal", "Open modal"),
  fluidRow(
    column(
      width = 4,
      block_output(plotOutput(outputId = "plot1"))
    ),
    column(
      width = 4,
      block_output(
        plotOutput(outputId = "plot2"),
        type = "hourglass",
        messageColor = "#FFF",
        svgColor = "#FFF",
        svgSize = "70px",
        backgroundColor = "#5ea4d8"
      )
    ),
    column(
      width = 4,
      plotOutput(outputId = "plot3")
    )
  )
)

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

  output$plot1 <- renderPlot({
    input$refresh
    barplot(table(floor(runif(100) * 6)))
  })

  data_r <- reactive({
    input$refresh
    Sys.sleep(3)
    data <- data.frame(x = rnorm(50), y = rnorm(50))
    return(data)
  })

  output$plot2 <- renderPlot({
    plot(data_r())
  })
```

```
  output$plot3 <- renderPlot({
    plot(data_r(), main = "Not blocked")
  })


  observeEvent(input$open_modal, {
    showModal(modalDialog(
      block_output(plotOutput(outputId = "plot_modal")),
      actionButton("refresh_modal", "Refresh plot")
    ))
  })

  output$plot_modal <- renderPlot({
    input$refresh_modal
    Sys.sleep(1)
    barplot(table(floor(runif(100) * 6)))
  })
}

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

---

busy-start-up   *Busy indicator at start up*

Description

Show a full-page busy indicator when application is initialized, then removed it after timeout, automatically or manually from server.

Usage

```
busy_start_up(
  loader,
  text = NULL,
  mode = c("timeout", "auto", "manual"),
  timeout = 500,
  color = "#112446",
  background = "#f0f0f0"
)

remove_start_up(timeout = 100, session = shiny::getDefaultReactiveDomain())
```

Arguments

loader   A spinner created with [spin_epic](#) or [spin_kit](#) or a simple HTML tag, to include a GIF (see examples).

text	Optional text to be displayed under the loading animation.
mode	How to remove the start-up page: "timeout", "auto" or "manual", see below for details.
timeout	Time (in milliseconds) to wait before removing the start-up page.
color	Color of text.
background	Background color.
session	Shiny session.

Details

Behavior according to mode argument:

- **timeout**: Busy indicator will be removed after the time (in milliseconds) specified in timeout.
- **manual**: Busy indicator will be removed with remove_start_up from server, timeout from busy_start_up is ignored in favor of that of remove_start_up.
- **auto**: Busy indicator is removed after JavaScript shiny:idle is triggered for the first time, timeout is taken into account.

When using timeout or auto, you can still remove the busy indicator with remove_start_up.

Value

HTML tag that can be included in UI definition.

Examples

```
# with timeout -----------------------------------------------------

library(shiny)
library(shinybusy)

ui <- fluidPage(

  busy_start_up(
    loader = spin_epic("orbit", color = "#FFF"),
    text = "Loading...",
    timeout = 1500,
    color = "#FFF",
    background = "#112446"
  ),

  tags$h1("Ready to play!", class = "text-center")

)

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

}

if (interactive())
```

```
  shinyApp(ui, server)


# manual ----------------------------------------------------------------

library(shiny)
library(shinybusy)

ui <- fluidPage(

  busy_start_up(
    loader = spin_kit(
      spin = "cube-grid",
      color = "#FFF",
      style = "width:50px; height:50px;"
    ),
    text = "Loading...",
    mode = "manual",
    color = "#FFF",
    background = "#112446"
  ),

  tags$h1("Ready to play!", class = "text-center")

)

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

  # Remove after 3 seconds (+timeout)
  observe({
    Sys.sleep(3)
    remove_start_up(timeout = 200)
  })

}

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


# auto & GIF -------------------------------------------------------------

library(shiny)
library(shinybusy)

ui <- fluidPage(

  busy_start_up(
    loader = tags$img(
      src = "https://jeroen.github.io/images/banana.gif",
      width = 100
    ),
    text = "Loading...",
    mode = "auto"
```

```
  ),

  tags$h1("Ready to play!", class = "text-center"),
  plotOutput(outputId = "plot")

)

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

  output$plot <- renderPlot({
    Sys.sleep(2)
    plot(rnorm(100))
  })

}

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

---

config_notify   *Configure options for* [notify()](#) *and others*

Description

Options for [notify()](#) functions, see [online documentation](#) for default values and examples.

Usage

```
config_notify(
  background = NULL,
  textColor = NULL,
  childClassName = NULL,
  notiflixIconColor = NULL,
  fontAwesomeClassName = NULL,
  fontAwesomeIconColor = NULL,
  backOverlayColor = NULL,
  width = NULL,
  distance = NULL,
  opacity = NULL,
  borderRadius = NULL,
  rtl = NULL,
  messageMaxLength = NULL,
  backOverlay = NULL,
  plainText = NULL,
  showOnlyTheLastOne = NULL,
  clickToClose = NULL,
  pauseOnHover = NULL,
  ID = NULL,
```

```
    className = NULL,
    zindex = NULL,
    fontFamily = NULL,
    fontSize = NULL,
    cssAnimation = NULL,
    cssAnimationDuration = NULL,
    cssAnimationStyle = NULL,
    closeButton = NULL,
    useIcon = NULL,
    useFontAwesome = NULL,
    fontAwesomeIconStyle = NULL,
    fontAwesomeIconSize = NULL,
    ...
)
```

Arguments

`background`	Changes the background color.
`textColor`	Changes the text color.
`childClassName`	Changes the class name.
`notiflixIconColor`	
	Changes the SVG icon color.
`fontAwesomeClassName`	
	Changes the FontAwesome icon class name (FontAwesome has to be added to the project separately.)
`fontAwesomeIconColor`	
	Changes the FontAwesome icon color.
`backOverlayColor`	
	Changes the color of the background overlay.
`width`	Changes the width of the notifications.
`distance`	The distance between positioned notifications and the body element.
`opacity`	Changes the opacity. (Between 0 and 1)
`borderRadius`	Changes the radius of the notifications corners.
`rtl`	Specifies the text direction to "right-to-left".
`messageMaxLength`	
	The maximum length of the notifications message text.
`backOverlay`	Adds a background overlay to the notifications.
`plainText`	Strips all HTML tags.
`showOnlyTheLastOne`	
	Auto-removes all the notifications except for the last one.
`clickToClose`	Removes the notification when it has been clicked without waiting for the delay.
`pauseOnHover`	Auto-remove functionality will be paused for each notification element when the pointer(mouse) enters on it.
`ID`	Changes the ID (attribute) of the notifications.

className	Changes the class name (attribute) of the notifications.
zindex	Changes the z-index of the notifications.
fontFamily	Changes the font-family of the notifications message text.
fontSize	Changes the font-size of the notifications message text.
cssAnimation	Enables/disables CSS animations to show/hide the notifications.
cssAnimationDuration	
	Changes the CSS animations duration as milliseconds.
cssAnimationStyle	
	6 types of styles can be used: fade zoom from-right from-top from-bottom from-left
closeButton	Adds a close button/icon to the notifications. (Notifications with a close button won't disappear until they were clicked.)
useIcon	Allows using built-in SVG or external FontAwesome icons in the notifications. (By default, built-in SVG icons have been defined.)
useFontAwesome	Ignores built-in SVG icons and allows to use of external FontAwesome icons.
fontAwesomeIconStyle	
	2 types of styles can be used: basic shadow
fontAwesomeIconSize	
	Changes the font-size of the FontAwesome icons
...	Other potential arguments.

Value

A config list that can be used in notify() and other notify_* functions.

Examples

```
library(shiny)
library(shinybusy)

ui <- fluidPage(
  tags$h2("config for notify examples"),
  actionButton("success", "Success")
)

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

  observeEvent(input$success, {
    notify_success(
      "Well done!",
      config_notify(
        background = "#0431B4",
        notiflixIconColor = "#FFF"
      )
    )
  })
```

```
  }

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

---

config_report *Configure options for* report() *and others*

Description

Options for report() functions, see online documentation for default values and examples.

Usage

```
config_report(
  svgColor = NULL,
  titleColor = NULL,
  messageColor = NULL,
  buttonBackground = NULL,
  buttonColor = NULL,
  backOverlayColor = NULL,
  className = NULL,
  width = NULL,
  backgroundColor = NULL,
  borderRadius = NULL,
  rtl = NULL,
  zindex = NULL,
  backOverlay = NULL,
  fontFamily = NULL,
  svgSize = NULL,
  plainText = NULL,
  titleFontSize = NULL,
  titleMaxLength = NULL,
  messageFontSize = NULL,
  messageMaxLength = NULL,
  buttonFontSize = NULL,
  buttonMaxLength = NULL,
  cssAnimation = NULL,
  cssAnimationDuration = NULL,
  cssAnimationStyle = NULL,
  ...
)
```

Arguments

svgColor	Changes the built-in SVG icon color.
titleColor	Changes the title text color.

messageColor	Changes the message text color.
buttonBackground	
	Changes the button background color.
buttonColor	Changes the button text color.
backOverlayColor	
	Changes the color of the background overlay.
className	Changes the class name (attribute).
width	Changes the width.
backgroundColor	
	Changes the background color.
borderRadius	Changes the radius of the corners.
rtl	Specifies the text direction to "right-to-left".
zindex	Changes the z-index.
backOverlay	Adds a background overlay.
fontFamily	Changes the font-family.
svgSize	Changes the built-in SVG icons width and height. (Notiflix uses square scaled icons.)
plainText	Strips all HTML tags.
titleFontSize	Changes the font-size of the title text.
titleMaxLength	The maximum length of the title text.
messageFontSize	
	Changes the font-size of the message text.
messageMaxLength	
	The maximum length of the message text.
buttonFontSize	Changes the font-size of the button text.
buttonMaxLength	
	The maximum length of the button text.
cssAnimation	Enables/disables CSS animations to show/hide.
cssAnimationDuration	
	Changes the CSS animations duration as milliseconds.
cssAnimationStyle	
	2 types of styles can be used: fade zoom.
...	Other potential arguments.

Value

A config list that can be used in [report()](#) and other report_* functions.

Examples

```
library(shiny)
library(shinybusy)

ui <- fluidPage(
  tags$h2("Config for report() examples"),
  actionButton("success", "Success"),
  actionButton("failure", "Failure"),
  actionButton("info", "Info")
)

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

  observeEvent(input$success, {
    report_success(
      "Well done!",
      "All in order",
      config_report(
        svgColor = "#0431B4",
        titleColor = "#0431B4"
      )
    )
  })

  observeEvent(input$failure, {
    report_failure(
      "Oups...",
      "Something went wrong",
      config_report(
        svgColor = "#DF01D7",
        titleColor = "#DF01D7"
      )
    )
  })

  observeEvent(input$info, {
    report_info(
      "For your information",
      tags$p(
        style = "font-style: italic;",
        "Lorem ipsum dolor sit amet"
      ),
      config_report(width = "560px", borderRadius = "5px")
    )
  })

}

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

---

html-dependencies   *HTML dependencies used by shinybusy*

Description

HTML dependencies used by shinybusy

Usage

```
html_dependency_spinkit()

html_dependency_epic()

html_dependency_shinybusy()

html_dependency_freezeframe()

html_dependency_nanobar()

html_dependency_notiflix()

html_dependency_startup()

html_dependency_loading()

html_dependency_busy()

html_dependency_notify()

html_dependency_report()

html_dependency_block()
```

Value

an `htmltools::htmlDependency()`.

---

logo_silex   *Silex logo for Shiny use*

Description

Silex logo for Shiny use

Usage

```
logo_silex()
```

Value

Path to gif

---

manual-gif	*Manual busy indicator (GIF)*

Description

Manual busy indicator (GIF)

Usage

```
use_busy_gif(
  src,
  timeout = 100,
 position = c("top-right", "top-left", "bottom-right", "bottom-left", "full-page",
    "free"),
  margins = c(10, 10),
  overlay_color = "rgba(0, 0, 0, 0.5)",
  overlay_css = NULL,
  height = "50px",
  width = "50px"
)

play_gif(session = shiny::getDefaultReactiveDomain())

stop_gif(session = shiny::getDefaultReactiveDomain())
```

Arguments

src	Path to the GIF, an URL or a file in www/ folder.
timeout	Number of milliseconds after the server is busy to display the GIF.
position	Where to display the GIF: `'top-right'`, `'top-left'`, `'bottom-right'`, `'bottom-left'`, `'full-page'`.
margins	Distance from margins, a vector of length two, where first element is distance from top/bottom, second element distance from right/left.
overlay_color	Background color for the overlay if `position = "full-page"`.
overlay_css	Additional CSS for the overlay, for example `"z-index: 1000;"` to make it appear above everything.
height, width	Height and width of the spinner, default to `'50px'` for both, must be specified.
session	Shiny session.

Value

An HTML tag that should be used in UI.

Examples

```
library(shiny)
library(shinybusy)

ui <- fluidPage(

  # Use this function somewhere in UI
  use_busy_gif(
    src = "https://jeroen.github.io/images/banana.gif",
    height = 70, width = 70
  ),

  actionButton("play", "Play GIF"),
  actionButton("stop", "Stop GIF")
)

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

  observeEvent(input$play, {
    play_gif()
  })

  observeEvent(input$stop, {
    stop_gif()
  })

}

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

---

manual-progressbar *Manual busy indicator (progress bar)*

Description

Declare use_busy_bar() in your UI and update value server-side with update_busy_bar().

Usage

```
use_busy_bar(color = "#112446", centered = FALSE, height = "8px")

update_busy_bar(value, session = shiny::getDefaultReactiveDomain())
```

Arguments

color	Progress bar color.
centered	Center the progress bar or not.
height	Height of the bar.
value	The new value for the progress bar.
session	Shiny session.

Examples

```
library(shiny)
library(shinybusy)

ui <- fluidPage(
  tags$h2("Manual nanobar"),
  use_busy_bar(color = "#01DF01", height = "15px"),
  actionButton(inputId = "go", label = "Go"),
  sliderInput(
    inputId = "set", label = "Set progress",
    min = 0, value = 0, max = 100
  )
)

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

  observeEvent(input$go, {
    update_busy_bar(0)
    for (i in 1:100) {
      Sys.sleep(0.1)
      update_busy_bar(i)
    }
  })

  observeEvent(input$set, {
    update_busy_bar(input$set)
  })

}

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

---

manual-spinner *Manual busy indicator (spinner)*

Description

Declare use_busy_spinner in your UI and show/hide server-side with show_spinner/hide_spinner.

Usage

```
use_busy_spinner(
  spin = "double-bounce",
  color = "#112446",
 position = c("top-right", "top-left", "bottom-right", "bottom-left", "full-page"),
  margins = c(10, 10),
  spin_id = NULL,
  height = "50px",
  width = "50px"
)

show_spinner(spin_id = NULL, session = shiny::getDefaultReactiveDomain())

hide_spinner(spin_id = NULL, session = shiny::getDefaultReactiveDomain())
```

Arguments

spin	Style of the spinner, see `spin_epic` or `spin_kit` for possible choices. Note that for `spin_epic`, height and width are ignored.
color	Color for the spinner, in a valid CSS format.
position	Where to display the spinner: `'top-right'`, `'top-left'`, `'bottom-right'`, `'bottom-left'`, `'full-page'`.
margins	Distance from margins, a vector of length two, where first element is distance from top/bottom, second element distance from right/left.
spin_id	An explicit id for the spinner, useful if you want to use multiple spinners.
height, width	Height and width of the spinner, default to `'50px'` for both, must be specified.
session	Shiny session.

Examples

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

  ui <- fluidPage(

    # Use this function somewhere in UI
    use_busy_spinner(spin = "fading-circle"),

    headerPanel('Iris k-means clustering'),

    sidebarLayout(
      sidebarPanel(
        selectInput('xcol', 'X Variable', names(iris)),
        selectInput('ycol', 'Y Variable', names(iris),
                    selected=names(iris)[[2]]),
        numericInput('clusters', 'Cluster count', 3,
                    min = 1, max = 9),
```

```
        actionButton("sleep", "Long calculation")
      ),
      mainPanel(
        plotOutput('plot1')
      )
    )
  )

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

    selectedData <- reactive({
      iris[, c(input$xcol, input$ycol)]
    })

    clusters <- reactive({
      kmeans(selectedData(), input$clusters)
    })

    output$plot1 <- renderPlot({
      palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
                "#FF7F00", "#FFFF33", "#A65628", "#F781BF",
                "#999999"))

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

    observeEvent(input$sleep, {
      show_spinner()
      Sys.sleep(5)
      hide_spinner()
    })

  }

  shinyApp(ui, server)
}
```

---

modal-gif   *Show a modal with a GIF*

Description

Make a pop-up window appear from the server with a GIF during long computation, remove it when finished.

Usage

```
show_modal_gif(
  src,
  text = NULL,
  height = "100px",
  width = "100px",
  modal_size = "s",
  session = shiny::getDefaultReactiveDomain()
)

remove_modal_gif(session = getDefaultReactiveDomain())
```

Arguments

`src`	Path to the GIF, an URL or a file in www/ folder.
`text`	Additional text to appear under the spinner.
`height, width`	Height and width of the spinner, default to `'50px'` for both, must be specified.
`modal_size`	One of `"s"` for small (the default), `"m"` for medium, or `"l"` for large.
`session`	The `session` object passed to function given to `shinyServer`.

Examples

```
if (interactive()) {

  library(shiny)
  library(shinybusy)

  ui <- fluidPage(

    tags$h1("Modal with spinner"),
    actionButton("sleep1", "Launch a long calculation"),
    actionButton("sleep2", "And another one")
  )

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

    observeEvent(input$sleep1, {
      show_modal_gif(
        src = "https://jeroen.github.io/images/banana.gif"
      )
      Sys.sleep(5)
      remove_modal_gif()
    })

    observeEvent(input$sleep2, {
      show_modal_gif(
        src = "https://jeroen.github.io/images/banana.gif",
        width = "300px", height = "300px",
        modal_size = "m",
```

```
        text = "Please wait..."
      )
      Sys.sleep(5)
      remove_modal_gif()
    })

  }

  shinyApp(ui, server)

}
```

---

modal-progress   *Show a modal with a progress bar*

Description

Make a pop-up window appear from the server with a spinner during long computation, remove it when finished.

Usage

```
show_modal_progress_line(
  value = 0,
  text = "auto",
  color = "#112446",
  stroke_width = 4,
  easing = "linear",
  duration = 1000,
  trail_color = "#eee",
  trail_width = 1,
  height = "15px",
  session = shiny::getDefaultReactiveDomain()
)

show_modal_progress_circle(
  value = 0,
  text = "auto",
  color = "#112446",
  stroke_width = 4,
  easing = "linear",
  duration = 1000,
  trail_color = "#eee",
  trail_width = 1,
  height = "200px",
  session = shiny::getDefaultReactiveDomain()
)
```

```
remove_modal_progress(session = getDefaultReactiveDomain())

update_modal_progress(
  value,
  text = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

value	Initial value or new value to set.
text	Text to display.
color	Main color.
stroke_width	Main width.
easing	CSS animation to use, ex.: "linear", "easeIn", "easeOut", "easeInOut".
duration	Animation duration (in milliseconds).
trail_color	Color of shape behind the main bar.
trail_width	Width of shape behind the main bar.
height	Container height.
session	The session object passed to function given to shinyServer.

Examples

```
if (interactive()) {

  library(shiny)
  library(shinybusy)

  ui <- fluidPage(

    tags$h1("Modal with progress bar"),
    actionButton("sleep1", "Launch a long calculation"),
    actionButton("sleep2", "And another one (different line options)"),
    tags$br(),
    actionButton("sleep3", "With a circle progress bar"),
    actionButton("sleep4", "With different circle options)")
  )

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

    observeEvent(input$sleep1, {
      show_modal_progress_line()
      for (i in 1:100) {
        update_modal_progress(
          value = i / 100
        )
        Sys.sleep(0.1)
      }
```

```
      remove_modal_progress()
    })

    observeEvent(input$sleep2, {
      show_modal_progress_line(
        color = "#DF0101",
        duration = 900,
        easing = "easeOut",
        text = "Starting computation"
      )
      Sys.sleep(0.1)
      for (i in 1:100) {
        update_modal_progress(
          value = i / 100,
          text = paste("Process", trunc(i/10), sprintf("(%02d%%)", i))
        )
        Sys.sleep(0.15)
      }
      remove_modal_progress()
    })


    observeEvent(input$sleep3, {
      show_modal_progress_circle()
      for (i in 1:100) {
        update_modal_progress(
          value = i / 100
        )
        Sys.sleep(0.1)
      }
      remove_modal_progress()
    })

    observeEvent(input$sleep4, {
      show_modal_progress_circle(
        color = "#DF0101",
        duration = 900,
        easing = "easeOut",
        text = "Starting computation",
        height = "300px"
      )
      Sys.sleep(0.1)
      for (i in 1:100) {
        update_modal_progress(
          value = i / 100,
          text = paste("Process", trunc(i/10), sprintf("(%02d%%)", i))
        )
        Sys.sleep(0.15)
      }
      remove_modal_progress()
    })

  }
```

```
  shinyApp(ui, server)

}
```

---

modal-spinner   *Show a modal with a spinner*

Description

Make a pop-up window appear from the server with a spinner during long computation, remove it when finished.

Usage

```
show_modal_spinner(
  spin = "double-bounce",
  color = "#112446",
  text = NULL,
  session = shiny::getDefaultReactiveDomain()
)

remove_modal_spinner(session = getDefaultReactiveDomain())

update_modal_spinner(text, session = shiny::getDefaultReactiveDomain())
```

Arguments

spin	Style of the spinner, see [spin_epic()](#) or [spin_kit()](#) for possible choices.
color	Color for the spinner, in a valid CSS format.
text	Additional text to appear under the spinner.
session	The session object passed to function given to shinyServer.

Examples

```
if (interactive()) {

  library(shiny)
  library(shinybusy)

  ui <- fluidPage(

    tags$h1("Modal with spinner"),
    actionButton("sleep1", "Launch a long calculation"),
    actionButton("sleep2", "And another one")
  )

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

```
    observeEvent(input$sleep1, {
      show_modal_spinner()
      Sys.sleep(5)
      remove_modal_spinner()
    })

    observeEvent(input$sleep2, {
      show_modal_spinner(
        spin = "cube-grid",
        color = "firebrick",
        text = "Please wait..."
      )
      Sys.sleep(5)
      remove_modal_spinner()
    })

  }

  shinyApp(ui, server)

}
```

---

notify	*Notifications*

Description

Send notifications to the user.

Usage

```
notify(
  text,
  ...,
  timeout = 3000,
 position = c("right-top", "right-bottom", "left-top", "left-bottom", "center-top",
    "center-bottom", "center-center"),
  type = c("success", "failure", "info", "warning"),
  session = shiny::getDefaultReactiveDomain()
)

notify_success(text, ..., timeout = 3000, position = "right-top")

notify_failure(text, ..., timeout = 3000, position = "right-top")

notify_info(text, ..., timeout = 3000, position = "right-top")

notify_warning(text, ..., timeout = 3000, position = "right-top")
```

Arguments

text	Text to be displayed.
...	Options passed to JavaScript method, see `config_notify()`.
timeout	The delay in milliseconds to hide and remove the notifications.
position	Position where to display the notification.
type	Type of notification: `success`, `failure`, `info` or `warning`.
session	Default Shiny session.

Value

No value.

References

Notify module from Notiflix library. More documentation and examples are available on the official website: `https://notiflix.github.io/notify`.

Examples

```
library(shiny)
library(shinybusy)

ui <- fluidPage(
  tags$h2("notify examples"),
  tags$p(
    "More examples available on the official website:",
    tags$a("https://notiflix.github.io/notify")
  ),
  actionButton("success", "Success"),
  actionButton("failure", "Failure"),
  actionButton("info", "Info"),
  actionButton("warning", "Warning")
)

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

  observeEvent(input$success, {
    notify_success("Well done!")
  })

  observeEvent(input$failure, {
    notify_failure("Oups...")
  })

  observeEvent(input$info, {
    notify_info("For your information")
  })

  observeEvent(input$warning, {
    notify_warning("Be careful!")
```

```
  })

}

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

---

progress   *Create progress indicator*

Description

Bar, circle or semicircle to show progress. Can be used outside Shiny. In Shiny you can set progress value server-side.

Usage

```
progress_line(
  value = 0,
  color = "#112446",
  stroke_width = 4,
  easing = "linear",
  duration = 1000,
  trail_color = "#eee",
  trail_width = 1,
  text = "auto",
  text_color = "#000",
  width = "100%",
  height = "15px",
  shiny_id = NULL
)

progress_circle(
  value = 0,
  color = "#112446",
  stroke_width = 4,
  easing = "easeInOut",
  duration = 1400,
  trail_color = "#eee",
  trail_width = 1,
  text = "auto",
  text_color = "#000",
  width = "200px",
  height = "200px",
  shiny_id = NULL
)
```

```
progress_semicircle(
  value = 0,
  color = "#112446",
  stroke_width = 4,
  easing = "easeInOut",
  duration = 1400,
  trail_color = "#eee",
  trail_width = 1,
  text = "auto",
  text_color = "#000",
  width = "200px",
  height = "100px",
  shiny_id = NULL
)

update_progress(
  shiny_id,
  value,
  text = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

value	Initial value or new value to set.
color	Main color.
stroke_width	Main width.
easing	CSS animation to use, ex.: "linear", "easeIn", "easeOut", "easeInOut".
duration	Animation duration (in milliseconds).
trail_color	Color of shape behind the main bar.
trail_width	Width of shape behind the main bar.
text	Text to display.
text_color	Text color.
width	Container width.
height	Container height.
shiny_id	Id to use in Shiny application.
session	Shiny session.

Value

an `htmlwidget` object.

Examples

```
# Default usage
progress_line(value = 0.5)

# change color
progress_line(value = 0.5, color = "firebrick")

# Circle
progress_circle(value = 0.5)


# Shiny usage
if (interactive()) {
  library(shiny)
  library(shinybusy)

  ui <- fluidPage(
    tags$h2("Progress bars examples"),
    fluidRow(
      column(
        width = 4,
        tags$p("Default bar:"),
        progress_line(value = 0, shiny_id = "bar"),
        sliderInput(
          inputId = "update_bar",
          label = "Update:",
          min = 0, max = 1,
          value = 0, step = 0.1
        ),
        tags$p("Set custom text:"),
        progress_line(
          value = 0.5,
          text = "To update",
          shiny_id = "text"
        ),
        textInput(
          inputId = "update_text",
          label = "Update:"
        )
      ),
      column(
        width = 4,
        tags$p("Default circle:"),
        progress_circle(value = 0, shiny_id = "circle"),
        sliderInput(
          inputId = "update_circle",
          label = "Update:",
          min = 0, max = 1,
          value = 0, step = 0.1,
          width = "100%"
        )
      ),
```

```
      column(
        width = 4,
        tags$p("Default semi-circle:"),
        progress_semicircle(value = 0, shiny_id = "semicircle"),
        sliderInput(
          inputId = "update_semicircle",
          label = "Update:",
          min = 0, max = 1,
          value = 0, step = 0.1,
          width = "100%"
        )
      )
    )
  )

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

    observe({
      update_progress("bar", input$update_bar)
    })

    observe({
      update_progress("circle", input$update_circle)
    })

    observe({
      update_progress("semicircle", input$update_semicircle)
    })

    observe({
      req(input$update_text)
      update_progress("text", 0.5, input$update_text)
    })

  }

  shinyApp(ui, server)
}
```

---

report	*Reports*

Description

show extended notifications that contain a title, description to the user.

Usage

```
report(
  title,
```

```
  text,
  ...,
  button = "Ok",
  type = c("success", "failure", "info", "warning"),
  session = shiny::getDefaultReactiveDomain()
)

report_success(title, text, ..., button = "Ok")

report_failure(title, text, ..., button = "Ok")

report_info(title, text, ..., button = "Ok")

report_warning(title, text, ..., button = "Ok")
```

Arguments

title	Title of the report.
text	Text to be displayed.
...	Options passed to JavaScript method, see `config_report()`.
button	Label for the button.
type	Type of notification: `success`, `failure`, `info` or `warning`.
session	Default Shiny session.

Value

No value.

References

Report module from Notiflix library. More documentation and examples are available on the official website: `https://notiflix.github.io/report`.

Examples

```
library(shiny)
library(shinybusy)

ui <- fluidPage(
  tags$h2("Report examples"),
  tags$p(
    "More examples available on the official website:",
    tags$a("https://notiflix.github.io/report")
  ),
  actionButton("success", "Success"),
  actionButton("failure", "Failure"),
  actionButton("info", "Info"),
  actionButton("warning", "Warning")
)
```

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

  observeEvent(input$success, {
    report_success(
      "Well done!",
      "All in order"
    )
  })

  observeEvent(input$failure, {
    report_failure(
      "Oups...",
      "Something went wrong"
    )
  })

  observeEvent(input$info, {
    report_info(
      "For your information",
      tags$p(
        style = "font-style: italic;",
        "Lorem ipsum dolor sit amet"
      )
    )
  })

  observeEvent(input$warning, {
    report_warning(
      "Be careful!",
      "There were 30 warnings (use warnings() to see them)"
    )
  })

}

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

---

spin_epic   *Epic spinners*

Description

Via <https://epic-spinners.epicmax.co/>.

Usage

```
spin_epic(
  spin = c("flower", "pixel", "hollow-dots", "intersecting-circles", "orbit", "radar",
```

```
    "scaling-squares", "half-circle", "trinity-rings", "fulfilling-square",
  "circles-to-rhombuses", "semipolar", "self-building-square", "swapping-squares",
  "fulfilling-bouncing-circle", "fingerprint", "spring", "atom", "looping-rhombuses",
    "breeding-rhombus"),
  color = "#112446"
)
```

Arguments

spin	Name of the spinner.
color	Color of the spinner.

Value

an HTML tag.

Examples

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

  ui <- fluidPage(
    tags$h2("Epic spinner demo"),
    lapply(
      X = c(
        "flower", "pixel", "hollow-dots",
        "intersecting-circles", "orbit", "radar",
        "scaling-squares", "half-circle",
        "fulfilling-square", "circles-to-rhombuses"
      ),
      FUN = function(x) {
        tags$div(
          style = "display: table-cell; width: 150px; height: 100px; margin: 10px;",
          tags$b(x),
          spin_epic(x, color = "#08298A")
        )
      }
    ),
    tags$hr(),
    lapply(
      X = c(
        "semipolar", "self-building-square", "swapping-squares",
        "fulfilling-bouncing-circle", "fingerprint", "spring",
        "atom", "looping-rhombuses", "breeding-rhombus", "trinity-rings"
      ),
      FUN = function(x) {
        tags$div(
          style = "display: table-cell; width: 150px; height: 100px; margin: 10px;",
          tags$b(x),
          spin_epic(x, color = "#08298A")
        )
```

```
      }
    )
  )

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

  }

  shinyApp(ui, server)
}
```

---

spin_kit   *SpinKit spinners*

Description

Via <https://tobiasahlin.com/spinkit/>.

Usage

```
spin_kit(
  spin = c("double-bounce", "circle", "bounce", "folding-cube", "rotating-plane",
    "cube-grid", "fading-circle", "dots", "cube"),
  color = "#112446",
  style = NULL
)
```

Arguments

spin	Name of the spinner.
color	Color of the spinner.
style	If not NULL, add a div container with specified style.

Value

an HTML tag.

Examples

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

  ui <- fluidPage(
    tags$h2("SpinKit demo"),
    fluidRow(lapply(
      X = c(
        "circle", "bounce", "folding-cube", "rotating-plane", "cube-grid",
        "fading-circle", "double-bounce", "dots", "cube"
```

```
      ),
      FUN = function(x) {
        column(
          width = 2,
          tags$b(x),
          tags$div(
            style = "width: 60px; height: 60px; position: relative;",
            spin_kit(spin = x)
          )
        )
      }
    ))
  )

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

  }

  shinyApp(ui, server)
}
```

Index