

Package: shinyglide (via r-universe)

August 30, 2024

Type Package

Title Glide Component for Shiny Applications

Version 0.1.4.9000

Date 2023-02-16

Maintainer Julien Barnier <julien.barnier@cnrs.fr>

Description Insert Glide JavaScript component into Shiny applications
for carousel or assistant-like user interfaces.

License GPL (>= 3)

VignetteBuilder knitr

URL <https://juba.github.io/shinyglide/>,
<https://github.com/juba/shinyglide>

BugReports <https://github.com/juba/shinyglide/issues>

Encoding UTF-8

Imports shiny (>= 1.2.0), htmltools

Suggests knitr, rmarkdown

Roxygen list(markdown = TRUE)

LazyData true

RoxygenNote 7.2.3

Repository <https://juba.r-universe.dev>

RemoteUrl <https://github.com/juba/shinyglide>

RemoteRef HEAD

RemoteSha b0cc29fea5fc90d20a711d4617d474c449c05ff2

Contents

firstButton	2
glide	2
glideControls	4
nextButton	5
screen	5
screenOutput	7

firstButton	<i>Create a glide control only shown on first or last screen</i>
-------------	--

Description

Create a glide control only shown on first or last screen

Usage

```
firstButton(class = c("btn", "btn-default"), ...)
```

```
lastButton(class = c("btn", "btn-success"), ...)
```

Arguments

class CSS classes of the control. The needed class is automatically added.

... content of the control

Details

These controls generate an `<a>` tag, so you can use href attributes.

firstButton is only shown on the first screen of the app, and finalButton only on the last screen.

Examples

```
firstButton("Go to website", href = "https://example.com", class = "btn btn-primary")
```

glide	<i>Glide component creation</i>
-------	---------------------------------

Description

Insert a glide component in the current shiny app UI

Usage

```
glide(
  ...,
  id = NULL,
  next_label = paste("Next", shiny::icon("chevron-right", lib = "glyphicon")),
  previous_label = paste(shiny::icon("chevron-left", lib = "glyphicon"), "Back"),
  loading_label = span(span(class = "shinyglide-spinner"), span("Loading")),
  loading_class = "loading",
  disable_type = c("disable", "hide"),
  height = "100%",
  keyboard = TRUE,
  swipe = TRUE,
  start_at = 0,
  custom_controls = NULL,
  controls_position = c("bottom", "top")
)
```

Arguments

...	content of the glide.
id	optional HTML id of the glide root element.
next_label	label to be used in the "next" control.
previous_label	label to be used in the "back" control.
loading_label	label to be used in the "next" control when the next screen is still loading.
loading_class	class to add to the "next" control when the next screen is still loading.
disable_type	either to "disable" or "hide" the next or back control when it is disabled by a condition.
height	height of the glide (something like "400px" or "100%").
keyboard	set this to FALSE to disable keyboard navigation.
swipe	set this to FALSE to disable swipe navigation.
start_at	start at specific slide number defined with zero-based index (default 0).
custom_controls	custom HTML or shiny tags to be used for the controls. If 'NULL', use the default ones.
controls_position	either to place the default or custom controls on "top" or "bottom" of the glide.

See Also

screen nextButton prevButton firstButton lastButton

Examples

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

```
ui <- fixedPage(  
  h3("Simple shinyglide app"),  
  glide(  
    screen(  
      p("First screen.")  
    ),  
    screen(  
      p("Second screen.")  
    )  
  )  
)  
  
server <- function(input, output, session) {  
}  
  
shinyApp(ui, server)  
}
```

glideControls

Default controls layout

Description

Creates an horizontal layout with both "previous" and "next" contents side by side.

Usage

```
glideControls(previous_content = prevButton(), next_content = nextButton())
```

Arguments

`previous_content` Content of the "previous" (left) zone.
`next_content` Content of the "next" (right) zone.

Examples

```
glideControls(  
  prevButton("Back"),  
  list(  
    lastButton(href = "https://example.com", "Go to website"),  
    nextButton("Next")  
  )  
)
```

nextButton	<i>Code for the default controls</i>
------------	--------------------------------------

Description

This generates the code of the default controls, and can be used in custom controls.

Usage

```
nextButton(class = c("btn", "btn-primary"))  
prevButton(class = c("btn", "btn-default"))
```

Arguments

class control CSS classes. The needed class is automatically added.

Details

prevButton is hidden on the first screen, while nextButton is hidden on the last one. The buttons labels are set with the next_label and previous_label arguments of glide().

See Also

glide

screen	<i>Screen creation</i>
--------	------------------------

Description

Insert a new screen into a glide component.

Usage

```
screen(  
  ...,  
  next_label = NULL,  
  previous_label = NULL,  
  next_condition = NULL,  
  previous_condition = NULL,  
  class = NULL  
)
```

Arguments

<code>...</code>	content of the screen.
<code>next_label</code>	specific label of the "next" control for this screen. If NULL, use the default one for the current glide.
<code>previous_label</code>	specific label of the "back" control for this screen. If NULL, use the default one for the current glide.
<code>next_condition</code>	condition for the "next" control to be enabled. Same syntax as <code>shiny::conditionalPanel</code> .
<code>previous_condition</code>	condition for the "back" control to be enabled. Same syntax as <code>shiny::conditionalPanel</code> .
<code>class</code>	screen CSS classes. <code>glide__slide</code> is automatically added.

Details

This function inserts a new "screen" into an existing glide component. It can only be used inside a `glide()` call, in a shiny app UI.

See Also

`glide`

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {
  ui <- fixedPage(
    h3("Simple shinyglide app"),
    glide(
      screen(
        next_label = "Go next",
        next_condition = "input.x > 0",
        p("First screen."),
        numericInput("x", "x", value = 0)
      ),
      screen(
        p("Final screen."),
      )
    )
  )
}

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

shinyApp(ui, server)
}
```

screenOutput	<i>Create a screen output element</i>
--------------	---------------------------------------

Description

Insert a screen output element in a shiny app UI. This must be used with a `renderUI` reactive expression in the app server.

Usage

```
screenOutput(  
  outputId,  
  next_label = NULL,  
  prev_label = NULL,  
  next_condition = NULL,  
  prev_condition = NULL,  
  class = NULL,  
  ...  
)
```

Arguments

<code>outputId</code>	output variable to read the value from
<code>next_label</code>	specific label of the "next" control for this screen. If <code>NULL</code> , use the default one for the current glide.
<code>prev_label</code>	specific label of the "back" control for this screen. If <code>NULL</code> , use the default one for the current glide.
<code>next_condition</code>	condition for the "next" control to be enabled. Same syntax as <code>shiny::conditionalPanel</code> .
<code>prev_condition</code>	condition for the "back" control to be enabled. Same syntax as <code>shiny::conditionalPanel</code> .
<code>class</code>	screen CSS classes. <code>glide__slide</code> is automatically added.
<code>...</code>	other arguments to pass to the container tag function.

Details

Important : for this to work, you have to add a `outputOptions(output, id, suspendWhenHidden = FALSE)` in your app server. See example.

Examples

```
## Only run examples in interactive R sessions  
if (interactive()) {  
  
  ui <- fixedPage(  
    h3("Simple shinyglide app"),  
    glide(  
      screen(  

```

```
      p("First screen."),
    ),
    screenOutput("screen"),
    screen(
      p("Final screen."),
    )
  )
)

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

  output$screen <- renderUI({
    p("Second screen.")
  })
  outputOptions(output, "screen", suspendWhenHidden = FALSE)

}

shinyApp(ui, server)

}
```


Index

`firstButton`, [2](#)

`glide`, [2](#)

`glideControls`, [4](#)

`lastButton (firstButton)`, [2](#)

`nextButton`, [5](#)

`prevButton (nextButton)`, [5](#)

`screen`, [5](#)

`screenOutput`, [7](#)