Package ‘shinyglide’

June 11, 2021

Type Package
Title Glide Component for Shiny Applications
Version 0.1.3
Date 2021-06-11
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
BugReports https://github.com/juba/shinyglide/issues
Encoding UTF-8
Imports shiny (>= 1.2.0), htmltools
Suggests knitr, rmarkdown
RoxygenNote 7.1.1
NeedsCompilation no
Author Julien Barnier [aut, cre]
Repository CRAN
Date/Publication 2021-06-11 14:40:02 UTC

R topics documented:

  firstButton     2
  glide           2
  glideControls   4
  nextButton      4
  screen          5
  screenOutput    6

Index 8
**firstButton**  
Create a glide control only shown on first or last screen

**Description**
Create a glide control only shown on first or last screen

**Usage**

```r
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 `lastButton` only on the last screen.

**Examples**

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

---

**glide**  
Glide component creation

**Description**

Insert a glide component in the current shiny app UI

**Usage**

```r
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"),
)```
### 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.

**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

```r
## 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**  Code for the default controls

**Description**

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

**Usage**

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>control CSS classes. The needed class is automatically added.</td>
</tr>
</tbody>
</table>

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

---

Description

Insert a new screen into a glide component.

Usage

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>content of the screen.</td>
</tr>
<tr>
<td>next_label</td>
<td>specific label of the &quot;next&quot; control for this screen. If NULL, use the default one for the current glide.</td>
</tr>
<tr>
<td>previous_label</td>
<td>specific label of the &quot;back&quot; control for this screen. If NULL, use the default one for the current glide.</td>
</tr>
<tr>
<td>next_condition</td>
<td>condition for the &quot;next&quot; control to be enabled. Same syntax as shiny::conditionalPanel.</td>
</tr>
<tr>
<td>previous_condition</td>
<td>condition for the &quot;back&quot; control to be enabled. Same syntax as shiny::conditionalPanel.</td>
</tr>
<tr>
<td>class</td>
<td>screen CSS classes. glide__slide is automatically added.</td>
</tr>
</tbody>
</table>

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.
screenOutput

Create a screen output element

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,
  ...
)

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)
}
Arguments

- outputId
  output variable to read the value from

- next_label
  specific label of the "next" control for this screen. If NULL, use the default one for the current glide.

- prev_label
  specific label of the "back" control for this screen. If NULL, use the default one for the current glide.

- next_condition
  condition for the "next" control to be enabled. Same syntax as shiny::conditionalPanel.

- prev_condition
  condition for the "back" control to be enabled. Same syntax as shiny::conditionalPanel.

- class
  screen CSS classes. glide__slide is automatically added.

- ...
  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

```r
## 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

slide, 2
glideControls, 4

lastButton (firstButton), 2

nextButton, 4

prevButton (nextButton), 4

screen, 5

screenOutput, 6