Package ‘shinyChakraUI’

October 14, 2022

Title A Wrapper of the 'React' Library 'Chakra UI' for 'Shiny'

Version 1.1.1

Description Makes the 'React' library 'Chakra UI' usable in 'Shiny' apps. 'Chakra UI' components include alert dialogs, drawers (sliding panels), menus, modals, popovers, sliders, and more.

License GPL (>= 3)

Encoding UTF-8

URL `https://github.com/stla/shinyChakraUI`

BugReports `https://github.com/stla/shinyChakraUI/issues`

RoxygenNote 7.1.2

Imports htmltools, reactR, shiny, jsonlite, rlang, stringr, grDevices, utils, formatR, fontawesome

Suggests testthat (>= 3.0.0), V8

Config/testthat/edition 3

NeedsCompilation no

Author Stéphane Laurent [aut, cre],
Segun Adebayo [cph] ('Chakra UI' library (https://chakra-ui.com/)),
David Kaye [ctb] ('json-normalize.js'),
RubyLouvre [cph] ('jsx-parser' library),
Terence Eden [cph] ('SuperTinyIcons' library
(https://github.com/edent/SuperTinyIcons/)),
Ionic (http://ionic.io/) [cph]

Maintainer Stéphane Laurent `<laurent_step@outlook.fr>`

Repository CRAN

Date/Publication 2022-01-05 15:30:06 UTC

R topics documented:

`chakraAlertDialog` *Alert dialog widget*

Description

An alert dialog widget.

Usage

```
chakraAlertDialog(
  inputId,
  options = chakraAlertDialogOptions(),
  openButton,
  header,
  body,
  footer
)
```

Arguments

`inputId`	widget id
`options`	named list of options created with `chakraAlertDialogOptions`
`openButton`	a Chakra button to open the alert dialog
`header`	an `AlertDialogHeader` element
`body`	an `AlertDialogBody` element
`footer`	an `AlertDialogFooter` element; usually it contains some Chakra buttons (that you can group with `Tag$ButtonGroup(...)`)

Details

You can use an `action` attribute and a `value` attribute to the Chakra buttons you put in the widget. For example, if you include the Chakra button `Tag$Button("Cancel", action = "cancel", value = "CANCEL")`, clicking this button will cancel the alert dialog and will set the Shiny value `"CANCEL"`. Other possible action attributes are `"close"` to close the alert dialog, `"disable"` to disable the alert dialog, and `"remove"` to entirely remove the widget.

Value

A widget that can be used in `chakraComponent`.

Examples

```
library(shiny)
library(shinyChakraUI)

ui <- chakraPage(

  br(),

  chakraComponent(
    "mycomponent",

    chakraAlertDialog(
      inputId = "alertDialog",
      openButton = Tag$Button(
        leftIcon = Tag$DeleteIcon(),
        colorScheme = "red",
        "Delete customer"
      ),
      header = Tag$AlertDialogHeader(
        fontSize = "lg",
        fontWeight = "bold",
        "Delete customer?"
      ),
      body = Tag$AlertDialogBody(
        "Are you sure? You can't undo this action afterwards."
      ),
      footer = Tag$AlertDialogFooter(
        Tag$ButtonGroup(
          spacing = "3",
          Tag$Button(
            action = "cancel",
            value = "CANCEL",
            "Cancel"
          ),
          Tag$Button(
            action = "disable",
            value = "DISABLE",
            colorScheme = "red",
            "Disable"
          ),
          Tag$Button(
            action = "remove",
            value = "REMOVE",
            "Remove"
          )
        )
      )
    )

  )

)

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

  observe({
    print(input[["alertDialog"]])
  })

}

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

chakraAlertDialogOptions

Alert dialog options

Description

Options for the alert dialog widget (chakraAlertDialog).

Usage

```
chakraAlertDialogOptions(
  closeOnEsc = TRUE,
  colorScheme = "red",
  isCentered = TRUE,
  motionPreset = "scale",
  size = "md",
  ...
)
```

Arguments

closeOnEsc	whether to close the modal on pressing the 'esc' key
colorScheme	a Chakra color scheme
isCentered	whether to center the modal on screen
motionPreset	transition that should be used for the modal; one of "scale", "none", "slideInBottom", or "slideInRight"
size	modal size, "sm", "md", "lg", "xl", "2xl", "full", "xs", "3xl", "4xl", "5xl", or "6xl"
...	other attributes of AlertDialog

Value

A named list, for usage in chakraAlertDialog.

chakraCheckboxWithChildren

Checkbox with child checkboxes

Description

A widget with a parent checkbox and child checkboxes.

Usage

```
chakraCheckboxWithChildren(
  inputId,
  parentCheckbox,
  ...,
  stackAttributes = list(pl = 6, mt = 1, spacing = 1)
)
```

Arguments

`inputId`	widget id
`parentCheckbox`	the parent checkbox
`...`	the child checkboxes
`stackAttributes`	list of attributes which control the layout

Value

A widget to use in `chakraComponent`.

Examples

```
library(shiny)
library(shinyChakraUI)

ui <- chakraPage(

  br(),

  chakraComponent(
    "mycomponent",

    chakraCheckboxWithChildren(
      "cwc",
      Tag$Checkbox(
        "Parent checkbox"
      ),
      Tag$Checkbox(
        "Child checkbox 1"
      ),
      Tag$Checkbox(
        defaultChecked = TRUE,
        "Child checkbox 2"
      )
    )

  )

)
```

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

  observe({
    print(input[["cwc"]])
  })

}

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

---

chakraColorSchemes *Chakra color schemes*

Description

List of Chakra color schemes (to use as a `colorScheme` attribute in e.g. Chakra buttons).

Usage

```
chakraColorSchemes()
```

Value

The names of the Chakra color schemes in a vector.

Examples

```
chakraColorSchemes()
```

---

chakraCombinedSlider *Combined slider and number input*

Description

A widget combining a slider and a number input.

Usage

```
chakraCombinedSlider(
  id,
  value,
  min,
  max,
  step = NULL,
  maxWidth = "400px",
  numericInputOptions = numberInputOptions(),
  spacing = "2rem",
  keepWithinRange = TRUE,
  clampValueOnBlur = TRUE,
  focusThumbOnChange = FALSE,
  trackColor = NULL,
  filledTrackColor = NULL,
  tooltip = TRUE,
  tooltipOptions = sliderTooltipOptions(),
  thumbOptions = sliderThumbOptions(),
  ...
)
```

Arguments

`id`	widget id
`value`	initial value
`min`	minimal value
`max`	maximal value
`step`	increment step
`maxWidth`	slider width
`numericInputOptions`	list of options for the number input created with `numberInputOptions`
`spacing`	the space between the number input and the slider
`keepWithinRange`	whether to forbid the value to exceed the max or go lower than min
`clampValueOnBlur`	similar to `keepWithinRange`
`focusThumbOnChange`	whether to focus the thumb on change
`trackColor`	color of the slider track
`filledTrackColor`	color of the filled slider track
`tooltip`	whether to set a tooltip to the thumb, to show the value
`tooltipOptions`	options of the tooltip, a list created with `sliderTooltipOptions`
`thumbOptions`	list of options for the thumb created with `sliderThumbOptions`
`...`	other attributes passed to `Slider`

Value

A widget to use in `chakraComponent`.

Examples

```
library(shiny)
library(shinyChakraUI)

ui <- chakraPage(

  br(), br(),

  chakraComponent(
    "mycomponent",

    chakraCombinedSlider(
      "slider",
      value = 5,
      min = 0,
      max = 10,
      step = 0.5,
      maxWidth = "300px",
      tooltip = TRUE,
      trackColor = "green.300",
      thumbOptions = sliderThumbOptions(
        width = 20, height = 20,
        borderColor = "firebrick", borderWidth = "3px"
      )
    )

  )

)

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

  observe({
    print(input[["slider"]])
  })

}

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

`chakraComponent` *Chakra component*

Description

Create a Chakra component.

Usage

```
chakraComponent(componentId, ...)
```

Arguments

`componentId`	component id
`...`	elements to include within the component

Value

A Shiny widget to use in a UI definition, preferably in `chakraPage`.

`chakraDrawer` *Drawer widget*

Description

Create a drawer widget, a panel that slides out from the edge of the screen.

Usage

```
chakraDrawer(
  inputId,
  openButton,
  options = chakraDrawerOptions(),
  isOpen = FALSE,
  closeButton = TRUE,
  header,
  body,
  footer
)
```

Arguments

`inputId`	widget id
`openButton`	a Chakra button to open the drawer
`options`	list of options created with `chakraDrawerOptions`
`isOpen`	Boolean, whether the drawer is initially open
`closeButton`	Boolean, whether to include a closing button
`header`	a `DrawerHeader` element
`body`	a `DrawerBody` element
`footer`	a `DrawerFooter` element

Details

Similarly to `chakraAlertDialog`, you can set an action attribute and a value attribute to the Chakra buttons you include in the Chakra drawer.

Value

A widget to use in `chakraComponent`.

Examples

```
library(shiny)
library(shinyChakraUI)

ui <- chakraPage(

  br(),

  chakraComponent(
    "mycomponent",

    chakraDrawer(
      "drawer",
      openButton = Tag$Button("Open Drawer"),
      options = chakraDrawerOptions(placement = "right"),
      header = Tag$DrawerHeader("I'm the header"),
      body = Tag$DrawerBody(
        Tag$Box("I'm the body")
      ),
      footer = Tag$DrawerFooter(
        Tag$ButtonGroup(
          spacing = "6",
          Tag$Button(
            value = "try me",
            "Try me"
          ),
          Tag$Button(
            action = "close",
            variant = "outline",
            "Close"
          )
        )
      )
    )

  )

)

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

  observe({
    print(input[["drawer"]])
  })

}

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

---

chakraDrawerOptions *Drawer options*

Description

Options for the drawer widget (`chakraDrawer`).

Usage

```
chakraDrawerOptions(
  closeOnEsc = TRUE,
  closeOnOverlayClick = TRUE,
  colorScheme = NULL,
  isCentered = FALSE,
  isFullHeight = FALSE,
  motionPreset = "scale",
  placement = "right",
  size = "xs",
  ...
)
```

Arguments

`closeOnEsc`	whether to close the panel on pressing the 'esc' key
`closeOnOverlayClick`	whether to close the panel on clicking the overlay
`colorScheme`	a chakra color scheme
`isCentered`	whether to center the modal on screen
`isFullHeight`	if `TRUE` and drawer's placement is `"top"` or `"bottom"`, the drawer will occupy the viewport height
`motionPreset`	transition that should be used for the modal; one of `"scale"`, `"none"`, `"slideInBottom"`, or `"slideInRight"`
`placement`	placement of the drawer, `"top"`, `"right"`, `"bottom"`, or `"left"`
`size`	modal size, `"sm"`, `"md"`, `"lg"`, `"xl"`, `"2xl"`, `"full"`, `"xs"`, `"3xl"`, `"4xl"`, `"5xl"`, or `"6xl"`
`...`	other attributes of `Drawer`

Value

A named list, for usage in `chakraDrawer`.

`chakraExample` *Run a Chakra example*

Description

A function to run examples of Shiny apps with Chakra components.

Usage

```
chakraExample(example, display.mode = "showcase", ...)
```

Arguments

`example`	example name
`display.mode`	the display mode to use when running the example; see `runApp`
`...`	arguments passed to `runApp`

Value

No return value, just launches a Shiny app.

Examples

```
if(interactive()){
  chakraExample("Menu")
}
```

`chakraExamples` *Chakra examples*

Description

List of Chakra examples.

Usage

```
chakraExamples()
```

Value

No return value, only prints a message listing the Chakra examples.

Examples

```
chakraExamples()
if(interactive()){
  chakraExample("MenuWithGroups")
}
```

chakraIcons *Chakra icons*

Description

List of Chakra icons.

Usage

```
chakraIcons()
```

Details

See all chakra icons.

Value

The names of the Chakra icons in a vector.

Examples

```
chakraIcons()
```

chakraModal *Modal widget*

Description

A modal widget.

Usage

```
chakraModal(
  inputId,
  options = chakraModalOptions(),
  openButton,
  isOpen = FALSE,
  header,
  body,
  footer
)
```

Arguments

`inputId`	widget id
`options`	named list of options created with `chakraModalOptions`
`openButton`	a Chakra button to open the modal
`isOpen`	whether the modal is initially open
`header`	a `ModalHeader` element
`body`	a `ModalBody` element
`footer`	a `ModalFooter` element; usually it contains some Chakra buttons (that you can group with `Tag$ButtonGroup(...)`)

Details

You can use an `action` attribute and a `value` attribute to the Chakra buttons you put in the widget. For example, if you include the Chakra button `Tag$Button("Close", action = "close", value = "CLOSE")`, clicking this button will close the modal and will set the Shiny value `"CLOSE"`. Other possible action attributes are `"cancel"` to cancel, `"disable"` to disable the modal, and `"remove"` to entirely remove the modal.

Value

A widget that can be used in `chakraComponent`.

Examples

```
library(shiny)
library(shinyChakraUI)

ui <- chakraPage(

  br(),

  chakraComponent(
    "mycomponent",

    chakraModal(
      inputId = "modal",
      openButton = Tag$Button(
        colorScheme = "orange",
        "Open Modal"
      ),
      header = Tag$ModalHeader(
        fontSize = "lg",
        fontWeight = "bold",
        "Modal title"
      ),
      body = Tag$ModalBody(
        "Sit nulla est ex deserunt exercitation anim occaecat."
      ),
      footer = Tag$ModalFooter(
            Tag$ButtonGroup(
              spacing = "3",
              Tag$Button(
                action = "close",
                value = "CLOSE",
                "Close"
              ),
              Tag$Button(
                action = "cancel",
                colorScheme = "red",
                "Cancel"
              )
            )
          )
        )

    )

  )

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

    observe({
      print(input[["modal"]])
    })

  }

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

---

chakraModalOptions *Modal options*

Description

Options for the modal widget (`chakraModal`).

Usage

```
chakraModalOptions(
  closeOnEsc = TRUE,
  isCentered = TRUE,
  motionPreset = "scale",
  size = "md",
  ...
)
```

Arguments

`closeOnEsc`	whether to close the modal on pressing the 'esc' key
`isCentered`	whether to center the modal on screen
`motionPreset`	transition that should be used for the modal; one of `"scale"`, `"none"`, `"slideInBottom"`, or `"slideInRight"`
`size`	modal size, `"sm"`, `"md"`, `"lg"`, `"xl"`, `"2xl"`, `"full"`, `"xs"`, `"3xl"`, `"4xl"`, `"5xl"`, or `"6xl"`
`...`	other attributes of `Modal`

Value

A named list, for usage in `chakraModal`.

`chakraPage` *Chakra page*

Description

Function to be used as the `ui` element of a Shiny app; it is intended to contain some `chakraComponent` elements.

Usage

```
chakraPage(...)
```

Arguments

`...`	elements to include within the page

Value

A UI definition that can be passed to the `shinyUI` function.

chakraPinInput *Pin input*

Description

Create a pin input widget.

Usage

```
chakraPinInput(
  id,
  label = NULL,
  nfields,
  type = "alphanumeric",
  size = "md",
  mask = FALSE,
  defaultValue = ""
)
```

Arguments

`id`	input id
`label`	optional label
`nfields`	number of fields
`type`	either `"alphanumeric"` or `"number"`
`size`	one of `"xs"`, `"sm"`, `"md"`, `"lg"`
`mask`	Boolean, whether to mask the user inputs (like a password input)
`defaultValue`	default value, can be partial

Value

A widget to use in `chakraComponent`.

Examples

```
library(shiny)
library(shinyChakraUI)

ui <- chakraPage(
  br(),
  chakraComponent(
    "mycomponent",
    chakraPinInput(
      "pininput", label = tags$h2("Enter password"),
      nfields = 3, mask = TRUE
    )
  )
)

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

  observe({
    print(input[["pininput"]])
  })

}

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

`chakraRangeSlider` *Chakra range slider*

Description

Create a Chakra range slider.

Usage

```
chakraRangeSlider(
  id,
  label = NULL,
  values,
  min,
  max,
  step = NULL,
  width = NULL,
  size = "md",
  colorScheme = "blue",
  orientation = "horizontal",
  focusThumbOnChange = TRUE,
  isDisabled = FALSE,
  isReadOnly = FALSE,
  isReversed = FALSE,
  trackColor = NULL,
  filledTrackColor = NULL,
  tooltip = TRUE,
  tooltipOptions = sliderTooltipOptions(),
  thumbOptionsLeft = sliderThumbOptions(),
  thumbOptionsRight = sliderThumbOptions(),
  shinyValueOn = "end",
  ...
)
```

Arguments

`id`	widget id
`label`	label (optional)
`values`	the two initial values
`min`	minimal value
`max`	maximal value
`step`	increment step
`width`	slider width
`size`	size, `"sm"`, `"md"`, or `"lg"`
`colorScheme`	a Chakra color scheme
`orientation`	slider orientation, `"horizontal"` or `"vertical"`
`focusThumbOnChange`	whether to focus the thumb on change
`isDisabled`	whether to disable the slider
`isReadOnly`	read only mode
`isReversed`	whether to reverse the slider
`trackColor`	color of the track
`filledTrackColor`	color of the filled track
`tooltip`	whether to set a tooltip to the thumb
`tooltipOptions`	options of the tooltip, a list created with `sliderTooltipOptions`
`thumbOptionsLeft`	list of options for the left thumb, created with `sliderThumbOptions`
`thumbOptionsRight`	list of options for the right thumb, created with `sliderThumbOptions`
`shinyValueOn`	either `"drag"` or `"end"`, the moment to get the Shiny value
`...`	other attributes passed to `RangeSlider`

Value

A widget to use in `chakraComponent`.

Examples

```
# Run `chakraExample("RangeSlider")` to see a better example.
library(shiny)
library(shinyChakraUI)

ui <- chakraPage(

  br(),

  chakraComponent(
    "mycomponent",

    chakraRangeSlider(
      "slider",
      label = HTML("<span style='color:red'>Hello range slider!</span>"),
      values = c(2, 8),
      min = 0,
      max = 10,
      width = "50%",
      tooltip = TRUE,
      tooltipOptions = sliderTooltipOptions(placement = "bottom"),
      shinyValueOn = "end"
    )

  )

)

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

  observe({
    print(input[["slider"]])
  })

}

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

---

chakraSlider *Chakra slider*

Description

Create a Chakra slider.

Usage

```
chakraSlider(
  id,
  label = NULL,
  value,
  min,
  max,
  step = NULL,
  width = NULL,
  size = "md",
  colorScheme = "blue",
  orientation = "horizontal",
  focusThumbOnChange = TRUE,
  isDisabled = FALSE,
  isReadOnly = FALSE,
  isReversed = FALSE,
  trackColor = NULL,
  filledTrackColor = NULL,
  mark = FALSE,
  markOptions = sliderMarkOptions(),
  tooltip = TRUE,
  tooltipOptions = sliderTooltipOptions(),
  thumbOptions = sliderThumbOptions(),
  shinyValueOn = "end",
  ...
)
```

Arguments

`id`	widget id
`label`	label (optional)
`value`	initial value
`min`	minimal value
`max`	maximal value
`step`	increment step
`width`	slider width
`size`	size, `"sm"`, `"md"`, or `"lg"`
`colorScheme`	a Chakra color scheme
`orientation`	slider orientation, `"horizontal"` or `"vertical"`
`focusThumbOnChange`	whether to focus the thumb on change
`isDisabled`	whether to disable the slider
`isReadOnly`	read only mode
`isReversed`	whether to reverse the slider
`trackColor`	color of the track
`filledTrackColor`	color of the filled track
`mark`	whether to set a mark to the thumb (I personally prefer the tooltip)
`markOptions`	options of the mark, a list created with `sliderMarkOptions`
`tooltip`	whether to set a tooltip to the thumb
`tooltipOptions`	options of the tooltip, a list created with `sliderTooltipOptions`
`thumbOptions`	list of options for the thumb created with `sliderThumbOptions`
`shinyValueOn`	either `"drag"` or `"end"`, the moment to get the Shiny value
`...`	other attributes passed to `Slider`

Value

A widget to use in `chakraComponent`.

Examples

```
library(shiny)
library(shinyChakraUI)

ui <- chakraPage(

  br(),

  chakraComponent(
    "mycomponent",

    chakraSlider(
      "slider",
      label = HTML("<span style='color:red'>Hello slider!</span>"),
      value = 5,
      min = 0,
      max = 10,
      width = "50%",
      tooltip = TRUE,
      shinyValueOn = "end"
    )

  )

)

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

  observe({
    print(input[["slider"]])
  })

}

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

`createStandaloneToast` *The ‘createStandaloneToast‘ hook*

Description

The ‘createStandaloneToast‘ hook.

Usage

```
createStandaloneToast()
```

Details

See Standalone toasts.

Value

A list containing some URL-encoded JavaScript code.

Examples

```
library(shiny)
library(shinyChakraUI)

ui <- chakraPage(

  br(),

  chakraComponent(
    "mycomponent",

    withStates(

      Tag$Button(
        colorScheme = "orange",
        size = "lg",
        onClick = jseval(paste(
          '() => {',
          '  const toast = getState("toast");',
          '  toast({',
          '    position: "bottom",',
          '    title: "Account created.",',
          '    description: "We have created your account for you.",',
          '    status: "success",',
          '    duration: 3000,',
          '    isClosable: true',
          '  });',
          '}',
          sep = "\n")),

        "Show toast"
      ),

      states = list(toast = createStandaloneToast())

    )

  )

)
```

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

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

getHookProperty *Get hook property*

Description

Chakra hooks are JavaScript objects; this function allows to get a property (key) of a hook. See `useDisclosure` for an example.

Usage

```
getHookProperty(hook, property)
```

Arguments

`hook`	the name of the hook, usually created in the `states` list of the `withStates` function
`property`	the hook property you want to get

Value

A list like the return value of `jseval`.

getState *Get React state*

Description

Get the value of a React state.

Usage

```
getState(state)
```

Arguments

`state`	name of the state

Value

A list like the return value of `jseval`.

See Also

withStates

Examples

```
library(shiny)
library(shinyChakraUI)

ui <- chakraPage(

  br(),

  chakraComponent(
    "mycomponent",

    withStates(
      Tag$Fragment(

        Tag$Box(
          bg = "yellow.100",
          fontSize = "30px",
          width = "50%",
          getState("boxtext")
        ),

        br(),
        Tag$Divider(),
        br(),

        Tag$Button(
          colorScheme = "telegram",
          size = "lg",
          onClick = jseval('() => setState("boxtext", "Hello Chakra")'),
          "Change box text"
        )
      ),

      states = list(boxtext = "I am the box text")
    )

  )

)

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

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

`ionIcons` *Ionicons*

Description

List of ionicons.

Usage

```
ionIcons()
```

Details

See ionicons website.

Value

The names of the ionicons in a vector.

Examples

```
ionIcons()
```

`jseval` *Evaluate JS code*

Description

Evaluate JavaScript code in the application.

Usage

```
jseval(code)
```

Arguments

`code`	JavaScript code given as a string

Value

A list containing the URL-encoded JavaScript code.

Examples

```
library(shiny)
library(shinyChakraUI)

ui <- chakraPage(

  br(),

  chakraComponent(
    "mycomponent",

    Tag$Button(
      colorScheme = "pink",
      size = "lg",
      onClick = jseval('() => alert("Hello Chakra")'),
      "Trigger alert"
    )

  )

)

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

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

`jsx` *JSX element*

Description

Create a JSX element.

Usage

```
jsx(element, preamble = "")
```

Arguments

`element`	the JSX element given as a string
`preamble`	JavaScript code to run before, given as a string

Value

A list containing the URL-encoded strings `element` and `preamble`.

Examples

```
library(shiny)
library(shinyChakraUI)

ui <- chakraPage(

  chakraComponent(
    "mycomponent",

    jsx(paste(
      '<>',
      '  <Button onClick={onOpen}>Open Modal</Button>',
      '  <Modal isOpen={isOpen} onClose={onClose}>',
      '    <ModalOverlay />',
      '    <ModalContent>',
      '      <ModalHeader>Modal Title</ModalHeader>',
      '      <ModalCloseButton />',
      '      <ModalBody>',
      '        Sit nulla est ex deserunt exercitation anim occaecat.',
      '      </ModalBody>',
      '      <ModalFooter>',
      '        <Button colorScheme="blue" mr={3} onClick={onClose}>',
      '          Close',
      '        </Button>',
      '        <Button variant="ghost" onClick={setShinyValue}>',
      '          Secondary Action',
      '        </Button>',
      '      </ModalFooter>',
      '    </ModalContent>',
      '  </Modal>',
      '</>',
      sep = "\n"
    ),

    preamble = paste(
      'const { isOpen, onOpen, onClose } = useDisclosure();',
      'const setShinyValue = () => Shiny.setInputValue("modal", "action");',
      sep = "\n"
    )

  ))

)

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

  observe({
    print(input[["modal"]])
  })

}
```

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

jsxString2code *JSX string to React component code*

Description

Given a JSX string, this function prints the code of the corresponding React component that can be used in `chakraComponent`.

Usage

```
jsxString2code(jsxString, clipboard = TRUE)
```

Arguments

`jsxString`	JSX code given as a string
`clipboard`	whether to copy the output to the clipboard

Value

No return value, only prints the code in the console and copy it to the clipboard if `clipboard = TRUE`.

Note

Instead of using this function, rather use the RStudio addin provided by the package. Simply copy some JSX code to your clipboard, and select the 'JSX parser' addin in the RStudio Addins menu.

Examples

```
jsxString <- '<Input type="email" id="myinput" />'
jsxString2code(jsxString)
jsxString <- '<Button onClick={() => alert("hello")}>Hello</Button>'
jsxString2code(jsxString)
```

numberInputOptions *Options for the number input of the combined Chakra slider*

Description

Create a list of options to be passed to the numericInputOptions argument in chakraCombinedSlider.

Usage

```
numberInputOptions(
  precision = NULL,
  maxWidth = "80px",
  fontSize = NULL,
  fontColor = NULL,
  borderColor = NULL,
  focusBorderColor = NULL,
  borderWidth = NULL,
  incrementStepperColor = NULL,
  decrementStepperColor = NULL,
  ...
)
```

Arguments

precision	number of decimal points
maxWidth	width of the number input, e.g. "100px" or "20%"
fontSize	font size of the displayed value, e.g. "15px"
fontColor	color of the displayed value
borderColor	color of the border of the number input
focusBorderColor	color of the border of the number input on focus
borderWidth	width of the border of the number input, e.g. "3px" or "medium"
incrementStepperColor	color of the increment stepper
decrementStepperColor	color of the decrement stepper
...	other attributes of NumberInput

Value

A list of options to be passed to the numericInputOptions argument in chakraCombinedSlider.

setReactState *Set a React state*

Description

Set a React state from the Shiny server.

Usage

```
setReactState(session, componentId, stateName, value)
```

Arguments

`session`	Shiny session object
`componentId`	the id of the `chakraComponent` which contains the state to be changed
`stateName`	the name of the state to be set
`value`	the new value of the state; it can be an R object serializable to JSON, a React component, a JSX element created with the `jsx` function, a Shiny widget, or some HTML code created with the `HTML` function

Value

No return value, called for side effect.

See Also

`withStates`

Examples

```
library(shiny)
library(shinyChakraUI)

ui <- chakraPage(

  br(),

  chakraComponent(
    "mycomponent",

    Tag$Button(
      id = "button",
      className = "action-button",
      colorScheme = "facebook",
      display = "block",
      onClick = jseval("(event) => {event.target.disabled = true}"),
      "Click me to change the content of the container"
    ),
    br(),
    Tag$Divider(),
    br(),

    withStates(

      Tag$Container(
        maxW = "xl",
        centerContent = TRUE,
        bg = "yellow.100",
        getState("containerContent")
      ),

      states = list(containerContent = "I am the container content.")

    )

  )

)

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

  observeEvent(input[["button"]], {

    setReactState(
      session = session,
      componentId = "mycomponent",
      stateName = "containerContent",
      value = Tag$Box(
        padding = "4",
        maxW = "3xl",
        fontStyle = "italic",
        fontWeight = "bold",
        borderWidth = "2px",
        "I am the new container content, included in a Box."
      )
    )

  })

}

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

`sliderMarkOptions` *Slider mark options.*

Description

Define the options for the slider mark.

Usage

```
sliderMarkOptions(
  textAlign = "center",
  backgroundColor = "blue.500",
  textColor = "white",
  margin = "-35px 0 0 -25px",
  padding = "0 10px",
  width = "50px",
  ...
)
```

Arguments

`textAlign`	text alignment
`backgroundColor`	background color
`textColor`	text color
`margin`	margin (CSS property)
`padding`	padding (CSS property)
`width`	width
`...`	other attributes passed to `SliderMark`

Value

A list of attributes for usage in `chakraSlider`.

`sliderThumbOptions` *Slider thumb options*

Description

Define the Chakra slider thumb options.

Usage

```
sliderThumbOptions(
  width = NULL,
  height = NULL,
  color = NULL,
  borderColor = NULL,
  borderWidth = NULL,
  ...
)
```

Arguments

`width`	thumb width
`height`	thumb height
`color`	thumb color
`borderColor`	thumb border color
`borderWidth`	thumb border width
`...`	other attributes passed to `SliderThumb`

Value

A list of attributes for usage in `chakraSlider`, `chakraCombinedSlider`, or `chakraRangeSlider`.

`sliderTooltipOptions` *Slider tooltip options*

Description

Define the slider tooltip options.

Usage

```
sliderTooltipOptions(
  hasArrow = TRUE,
  backgroundColor = "red.600",
  color = "white",
  placement = "top",
  closeOnClick = FALSE,
  isOpen = TRUE,
  ...
)
```

Arguments

`hasArrow`	whether to include an arrow
`backgroundColor`	background color
`color`	content color
`placement`	tooltip placement; see tooltip placement
`closeOnClick`	whether to close the tooltip on click
`isOpen`	whether the tooltip is open
`...`	other attributes passed to `Tooltip`

Value

A list of attributes for usage in `chakraSlider`, `chakraCombinedSlider`, or `chakraRangeSlider`.

`superTinyIcons` *Super tiny icons*

Description

List of super tiny icons.

Usage

```
superTinyIcons()
```

Details

See all super tiny icons.

Value

The names of the super tiny icons in a vector.

Examples

```
superTinyIcons()
```

`Tag` *React component builder*

Description

Create a React component. This is similar to `React`.

Usage

```
Tag
```

Format

An object of class `ReactTagBuilder` of length 0.

Examples

```
Tag$Box(
  bg = "tomato",
  Tag$ButtonGroup(
    spacing = "4",
    Tag$Button(
      "I'm a button"
    ),
    Tag$Button(
      "I'm another button"
    )
  )
)
```

`useClipboard` *The 'useClipboard' hook*

Description

The 'useClipboard' hook.

Usage

```
useClipboard(value)
```

Arguments

`value`	a string

Details

See useClipboard.

Value

A list containing some URL-encoded JavaScript code.

See Also

`getHookProperty`

Examples

```
library(shiny)
library(shinyChakraUI)

ui <- chakraPage(

  br(),
  chakraComponent(
    "mycomponent",

    withStates(
      Tag$Box(
        width = "50%",

        Tag$Flex(
          mb = 2,
          Tag$Input(
            isReadOnly = TRUE,
            value = getHookProperty("clipboard", "value")
          ),
          Tag$Button(
            ml = 2,
            onClick = getHookProperty("clipboard", "onCopy"),
            jseval('getState("hasCopied") ? "Copied" : "Copy"')
          )
        ),

        br(),
        Tag$Divider(),
        br(),

        Tag$Editable(
          bg = "yellow.100",
          placeholder = "Paste here",
          Tag$EditablePreview(),
          Tag$EditableInput()
        )

      ),

      states = list(
        clipboard = useClipboard("Hello Chakra"),
        hasCopied = getHookProperty("clipboard", "hasCopied")
      )
    )

  )

)

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

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

`useDisclosure` *The 'useDisclosure' hook*

Description

The 'useDisclosure' hook.

Usage

```
useDisclosure(defaultIsOpen = FALSE)
```

Arguments

`defaultIsOpen`	Boolean, the initial value of the `isOpen` property

Details

See useDisclosure.

Value

A list containing some URL-encoded JavaScript code.

See Also

`getHookProperty`

Examples

```
library(shiny)
library(shinyChakraUI)

ui <- chakraPage(

  br(),

  chakraComponent(
    "mycomponent",

    withStates(
      Tag$Fragment(

        Tag$Button(
          colorScheme = "teal",
          variant = "outline",
          onClick = getHookProperty("disclosure", "onToggle"),
          "Click me!"
        ),

        Tag$Fade(
          "in" = getHookProperty("disclosure", "isOpen"),
          Tag$Box(
            p = "40px",
            color = "white",
            mt = "4",
            bg = "teal.500",
            rounded = "md",
            shadow = "md",
            "Fade"
          )
        )

      ),

      states = list(disclosure = useDisclosure())
    )

  )

)

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

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

useRef — *The ‘useRef‘ hook*

Description

The React ‘useRef‘ hook.

Usage

```
useRef(initialValue = NA)
```

Arguments

`initialValue`	optional initial value

Value

A list like the return value of `jseval`.

`useToast` *The ‘useToast‘ hook*

Description

The ‘useToast‘ hook.

Usage

```
useToast()
```

Value

A list containing some URL-encoded JavaScript code.

Note

It does not work well. Use `createStandaloneToast` instead.

`withStates` *Chakra component with states or hooks*

Description

Create a Chakra component with React states and/or hooks.

Usage

```
withStates(component, states)
```

Arguments

`component`	a React component
`states`	named list of states; a state value can be an R object serializable to JSON, a React component (`Tag$Component(...)`), a Shiny widget, some HTML code defined by the `HTML` function, a JSX element defined by the `jsx` function, a JavaScript value defined by the `jseval` function, or a hook such as `useDisclosure()` (see `useDisclosure`).

Value

A component to use in `chakraComponent`.

Examples

```
library(shiny)
library(shinyChakraUI)

ui <- chakraPage(

  br(),

  chakraComponent(
    "mycomponent",

    withStates(

      Tag$Fragment(

        Tag$Container(
          maxW = "xl",
          centerContent = TRUE,
          bg = "orange.50",
          Tag$Heading(
            getState("heading")
          ),
          Tag$Text(
            "I'm just some text."
          )
        ),

        br(),
        Tag$Divider(),
        br(),

        Tag$Button(
          colorScheme = "twitter",
          display = "block",
          onClick = jseval(
            "() => setState('heading', 'I am the new heading.')"
          ),
          "Click me to change the heading"
        )

      ),

      states = list(heading = "I am the heading.")

    )

  )

)

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

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

Index