Package 'shiny.blueprint'

May 21, 2024

Title Palantir's 'Blueprint' for 'Shiny' Apps

Version 0.3.0

Description Easily use 'Blueprint', the popular 'React' library from Palantir, in your 'Shiny' app. 'Blueprint' provides a rich set of UI components for creating visually appealing applications and is optimized for building complex, data-dense web interfaces. This package provides most components from the underlying library, as well as special wrappers for some components to make it easy to use them in 'R' without writing 'JavaScript' code.

License LGPL-3

Encoding UTF-8

RoxygenNote 7.3.1

VignetteBuilder knitr

Imports checkmate, htmltools, shiny, shiny.react (>= 0.4.0), utils

Suggests covr, knitr, lintr (>= 3.0.0), purrr, rcmdcheck, rmarkdown, shiny.router

NeedsCompilation no

Author Jakub Sobolewski [aut, cre], Kamil Żyła [aut], Filip Akkad [aut], Filip Stachura [aut], Paweł Chabros [aut], Appsilon Sp. z o.o. [cph]

Maintainer Jakub Sobolewski <opensource+jakub.sobolewski@appsilon.com>

Repository CRAN

Date/Publication 2024-05-21 11:20:11 UTC

R topics documented:

Alert 3
Breadcrumbs 4
Button 5

ButtonGroup . . . 6
Callout . . . 7
Card . . . 8
Checkbox . . . 9
Collapse . . . 10
ControlGroup . . . 11
Dialog . . . 12
Divider . . . 13
Drawer . . . 14
EditableText . . . 15
FileInput . . . 16
FormGroup . . . 17
htmlElements . . . 18
HTMLSelect . . . 20
HTMLTable . . . 21
Icon . . . 22
InputGroup . . . 23
Label . . . 25
Menu . . . 26
MultiSelect . . . 27
MultiSlider . . . 29
MultistepDialog . . . 31
Navbar . . . 33
NonIdealState . . . 34
NumericInput . . . 35
OverflowList . . . 36
Overlay . . . 37
PanelStack . . . 39
Popover . . . 41
ProgressBar . . . 42
Radio . . . 43
RangeSlider . . . 44
ResizeSensor . . . 45
runExample . . . 47
Select . . . 47
Slider . . . 49
Spinner . . . 50
Suggest . . . 51
Switch . . . 52
Tabs . . . 53
Tag . . . 54
TagInput . . . 55
Text . . . 56
TextArea . . . 57
Toaster . . . 58
Tree . . . 59

Index **63**

Alert *Alert*

Description

Documentation: https://blueprintjs.com/docs/#core/components/alert

Usage

```
Alert(...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  ns <- NS(id)
  tagList(
    Button.shinyInput(
      inputId = ns("showAlert"),
      "Show alert"
    ),
    reactOutput(ns("alert"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns

    isOpen <- reactiveVal(FALSE)
    observeEvent(input$showAlert, isOpen(TRUE))
    observeEvent(input$closeAlert, isOpen(FALSE))

    output$alert <- renderReact({
      Alert(
        usePortal = FALSE,
        confirmButtonText = "Got it",
        isOpen = isOpen(),
        onClose = triggerEvent(ns("closeAlert")),
        p("Hello, it's me, your alert")
```

```
      )
    })
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`Breadcrumbs` *Breadcrumbs*

Description

Documentation: https://blueprintjs.com/docs/#core/components/breadcrumbs

Usage

```
Breadcrumbs(...)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

items <- list(
  list(href = "/", icon = "folder-close", text = "Users"),
  list(href = "/", icon = "folder-close", text = "Janet"),
  list(icon = "document", text = "image.jpg")
)

ui <- function(id) {
  Breadcrumbs(items = items)
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

Button *Button*

Description

Documentation: https://blueprintjs.com/docs/#core/components/button

Usage

```
Button(...)

Button.shinyInput(inputId, ...)

AnchorButton(...)

AnchorButton.shinyInput(inputId, ...)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.
`inputId`	The `input` slot that will be used to access the value.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  ns <- NS(id)
  tagList(
    textOutput(ns("clicks")),
    Button(
      onClick = triggerEvent(ns("click1")),
      icon = "refresh",
      "Refresh"
    ),
    Button.shinyInput(
      inputId = ns("click2"),
      rightIcon = "share",
      "Export"
    ),
    AnchorButton(
      onClick = triggerEvent(ns("click3")),
      intent = "primary",
      "OK"
```

```
    ),
    AnchorButton.shinyInput(
      inputId = ns("click4"),
      intent = "success",
      "Next"
    )
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    clicks <- reactiveVal(0)
    output$clicks <- renderText(paste("Clicks:", clicks()))
    observeEvent(input$click1, clicks(clicks() + 1))
    observeEvent(input$click2, clicks(clicks() + 1))
    observeEvent(input$click3, clicks(clicks() + 1))
    observeEvent(input$click4, clicks(clicks() + 1))
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`ButtonGroup` *Button group*

Description

Documentation: https://blueprintjs.com/docs/#core/components/button-group

Usage

```
ButtonGroup(...)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  ButtonGroup(
    Button(icon = "database", "Queries"),
    Button(icon = "function", "Functions"),
```

```
    AnchorButton(rightIcon = "caret-down", "Options")
  )
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`Callout` *Callout*

Description

Documentation: https://blueprintjs.com/docs/#core/components/callout

Usage

```
Callout(...)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  Callout(
    title = "Visually important content",
    "The component is a simple wrapper around the CSS API",
    " that provides props for modifiers and optional title element.",
   " Any additional HTML props will be spread to the rendered ", Code("div"), " element."
  )
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

Card *Card*

Description

Documentation: https://blueprintjs.com/docs/#core/components/card

Usage

```
Card(...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  Card(
    interactive = TRUE,
    H5(tags$a(href = "#", "Analytical applications")),
    tags$p(
      "User interfaces that enable people to interact smoothly with data,",
      " ask better questions, and make better decisions."
    ),
    Button(text = "Explore products")
  )
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

Checkbox *Checkbox*

Description

Documentation: https://blueprintjs.com/docs/#core/components/checkbox

Usage

```
Checkbox(...)

Checkbox.shinyInput(inputId, ..., value = defaultValue)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.
`inputId`	The `input` slot that will be used to access the value.
`value`	Initial value.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

setInput <- function(inputId, accessor = NULL) {
  JS(paste0("x => Shiny.setInputValue('", inputId, "', x", accessor, ")"))
}

ui <- function(id) {
  ns <- NS(id)
  tagList(
    Checkbox(
      onChange = setInput(ns("apples"), ".target.checked"),
      defaultChecked = TRUE,
      label = "Apples"
    ),
    Checkbox.shinyInput(
      inputId = ns("bananas"),
      value = TRUE,
      label = "Bananas"
    ),
    textOutput(ns("applesEnabled")),
    textOutput(ns("bananasEnabled"))
  )
}
```

```
server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$applesEnabled <- renderText(paste("Apples:", deparse(input$apples)))
    output$bananasEnabled <- renderText(paste("Bananas:", deparse(input$bananas)))
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`Collapse` *Collapse*

Description

Documentation: https://blueprintjs.com/docs/#core/components/collapse

Usage

```
Collapse(...)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

logs <- Pre(
  "[11:53:30] Finished 'typescript-bundle-blueprint' after 769 ms\n",
  "[11:53:30] Starting 'typescript-typings-blueprint'...\n",
  "[11:53:30] Finished 'typescript-typings-blueprint' after 198 ms\n",
  "[11:53:30] write ./blueprint.css\n",
  "[11:53:30] Finished 'sass-compile-blueprint' after 2.84 s\n"
)

ui <- function(id) {
  ns <- NS(id)
  tagList(
    Button.shinyInput(ns("toggle"), "Toggle logs"),
    reactOutput(ns("ui"))
  )
}
```

```
server <- function(id) {
  moduleServer(id, function(input, output, session) {
    show <- reactiveVal(FALSE)
    observeEvent(input$toggle, show(!show()))
    output$ui <- renderReact({
      Collapse(isOpen = show(), logs)
    })
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`ControlGroup` *Control group*

Description

Documentation: https://blueprintjs.com/docs/#core/components/control-group

Usage

```
ControlGroup(...)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  ControlGroup(
    HTMLSelect(options = rownames(mtcars)),
    InputGroup(placeholder = "Find car..."),
    Button(icon = "arrow-right"),
  )
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

Dialog *Dialog*

Description

Documentation: https://blueprintjs.com/docs/#core/components/dialog.dialog

Usage

```
Dialog(...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  ns <- NS(id)
  tagList(
    Button.shinyInput(
      inputId = ns("showDialog"),
      "Show dialog"
    ),
    reactOutput(ns("dialog"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns

    isOpen <- reactiveVal(FALSE)
    observeEvent(input$showDialog, isOpen(TRUE))
    observeEvent(input$closeDialog, isOpen(FALSE))

    output$dialog <- renderReact({
      Dialog(
        usePortal = FALSE,
        isOpen = isOpen(),
        onClose = triggerEvent(ns("closeDialog")),
        div(
          className = "bp5-dialog-body",
```

```
            H5("Analytical applications"),
            tags$p(
              "User interfaces that enable people to interact smoothly with data,",
              " ask better questions, and make better decisions."
            ),
            Button.shinyInput(
              inputId = ns("closeDialog"),
              "Close"
            )
          )
        )
      })
    })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`Divider` *Divider*

Description

Documentation: https://blueprintjs.com/docs/#core/components/divider

Usage

```
Divider(...)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  ButtonGroup(
    minimal = TRUE,
    Button(text = "File"),
    Button(text = "Edit"),
    Divider(),
    Button(text = "Create"),
    Button(text = "Delete"),
    Divider(),
```

```
    Button(icon = "add"),
    Button(icon = "remove")
  )
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

Drawer *Drawer*

Description

Documentation: https://blueprintjs.com/docs/#core/components/drawer

Usage

```
Drawer(...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  ns <- NS(id)
  tagList(
    Button.shinyInput(ns("hello"), "Say Hello", intent = "primary"),
    reactOutput(ns("ui"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns

    isOpen <- reactiveVal(FALSE)
    observeEvent(input$hello, isOpen(!isOpen()))
    observeEvent(input$dismissDrawer, isOpen(FALSE))
```

```
    output$ui <- renderReact({
      Drawer(
        isOpen = isOpen(),
        onClose = triggerEvent(ns("dismissDrawer")),
        usePortal = FALSE,
        title = "Hello",
        icon = "info-sign",
        div(
          class = "bp5-dialog-body",
          p("Lorem Ipsum is simply dummy text of the printing and typesetting industry.")
        )
      )
    })
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`EditableText` *Editable text*

Description

Documentation: https://blueprintjs.com/docs/#core/components/editable-text

Usage

```
EditableText(...)

EditableText.shinyInput(inputId, ..., value = defaultValue)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.
`inputId`	The `input` slot that will be used to access the value.
`value`	Initial value.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  ns <- NS(id)
```

```
  tagList(
    H2(EditableText(onChange = setInput(ns("header")))),
    EditableText.shinyInput(
      inputId = ns("body"),
      multiline = TRUE,
      minLines = 3, maxLines = 12
    ),
    textOutput(ns("headerValue")),
    textOutput(ns("bodyValue"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$headerValue <- renderText(paste("Header:", deparse(input$header)))
    output$bodyValue <- renderText(paste("Body:", deparse(input$body)))
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`FileInput` *FileInput*

Description

Documentation: https://blueprintjs.com/docs/#core/components/file-input

Usage

```
FileInput(...)

FileInput.shinyInput(inputId, ..., value = defaultValue)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.
`inputId`	The `input` slot that will be used to access the value.
`value`	Initial value.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

setInput <- function(inputId, accessor = NULL) {
  JS(paste0("x => Shiny.setInputValue('", inputId, "', x", accessor, ")"))
}

ui <- function(id) {
  ns <- NS(id)
  tagList(
    Switch(
      onChange = setInput(ns("apples"), ".target.checked"),
      defaultChecked = TRUE,
      label = "Apples"
    ),
    Switch.shinyInput(
      inputId = ns("bananas"),
      value = TRUE,
      label = "Bananas"
    ),
    textOutput(ns("applesEnabled")),
    textOutput(ns("bananasEnabled"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$applesEnabled <- renderText(paste("Apples:", deparse(input$apples)))
    output$bananasEnabled <- renderText(paste("Bananas:", deparse(input$bananas)))
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`FormGroup` *Form group*

Description

Documentation: https://blueprintjs.com/docs/#core/components/form-group

Usage

```
FormGroup(...)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  FormGroup(
    helperText = "Helper text with details...",
    label = "Label A",
    labelFor = "my-button",
    labelInfo = "(required)",
    inline = TRUE,
    Switch(
      defaultChecked = TRUE,
      label = "Apples"
    ),
    Switch(
      defaultChecked = TRUE,
      label = "Bananas"
    )
  )
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`htmlElements` *HTML elements*

Description

Documentation: https://blueprintjs.com/docs/#core/components/html

Usage

```
H1(...)

H2(...)

H3(...)

H4(...)
```

`H5(...)`

`H6(...)`

`Blockquote(...)`

`Code(...)`

`Pre(...)`

`OL(...)`

`UL(...)`

Arguments

`...`	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

See Also

Other HTML elements: `HTMLTable()`, `Label()`

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  tagList(
    H1("H1"),
    H2("H2"),
    H3("H3"),
    H4("H4"),
    H5("H5"),
    H6("H6"),
    Blockquote("Blockquote"),
    Code("Code"),
    Label("Label"),
    Pre("Pre"),
    OL(tags$li("OL")),
    UL(tags$li("UL"))
  )
}

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

```
if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`HTMLSelect` *HTML select*

Description

Documentation: https://blueprintjs.com/docs/#core/components/html-select

Usage

```
HTMLSelect(...)

HTMLSelect.shinyInput(inputId, ..., value = defaultValue)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.
`inputId`	The `input` slot that will be used to access the value.
`value`	Initial value.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

setInput <- function(inputId, accessor = NULL) {
  JS(paste0("x => Shiny.setInputValue('", inputId, "', x", accessor, ")"))
}

options <- list(
  list(value = "a", label = "Apples"),
  list(value = "b", label = "Bananas"),
  list(value = "c", label = "Cherries")
)

ui <- function(id) {
  ns <- NS(id)
  tagList(
    HTMLSelect(
      onChange = setInput(ns("choice1"), ".target.value"),
      options = options
    ),
    textOutput(ns("text1")),
```

```
    br(),
    HTMLSelect.shinyInput(
      inputId = ns("choice2"),
      value = "b",
      options = options
    ),
    textOutput(ns("text2"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$text1 <- renderText(deparse(input$choice1))
    output$text2 <- renderText(deparse(input$choice2))
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`HTMLTable` *HTML table*

Description

Documentation: https://blueprintjs.com/docs/#core/components/html-table

Usage

```
HTMLTable(...)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

See Also

Other HTML elements: `Label()`, `htmlElements`

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  HTMLTable(
    tags$thead(
```

```
      tags$tr(tags$th("Project"), tags$th("Stack"), tags$th("Contributors"))
    ),
    tags$tbody(
      tags$tr(tags$td("Blueprint"), tags$td("JS React"), tags$td("268")),
      tags$tr(tags$td("TS"), tags$td("JSX"), tags$td("68")),
      tags$tr(tags$td("shiny.blueprint"), tags$td("R JS"), tags$td("2"))
    ),
    tags$tfoot(
      tags$tr(tags$td("Total", colSpan = 2), tags$td("1508"))
    )
  )
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`Icon` *Icon*

Description

Documentation: https://blueprintjs.com/docs/#core/components/icon

Usage

```
Icon(...)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.

Details

A list of available icons: https://blueprintjs.com/docs/#icons

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  tagList(
    Icon(icon = "cross"),
```

```
    Icon(icon = "globe", size = 20),
  )
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`InputGroup` *Input group*

Description

Documentation: https://blueprintjs.com/docs/#core/components/text-inputs.input-group

Usage

```
InputGroup(...)

InputGroup.shinyInput(inputId, ..., value = defaultValue)

TextArea.shinyInput(inputId, ..., value = defaultValue)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.
`inputId`	The `input` slot that will be used to access the value.
`value`	Initial value.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

setInput <- function(inputId, accessor = NULL) {
  JS(paste0("x => Shiny.setInputValue('", inputId, "', x", accessor, ")"))
}

ui <- function(id) {
  ns <- NS(id)
  div(
    style = "width: 20rem; display: grid; row-gap: 0.5rem",
    H4("Uncontrolled"),
```

```
    InputGroup(
      onChange = setInput(ns("uncontrolledInputGroup"), ".target.value"),
      disabled = FALSE,
      large = TRUE,
      leftIcon = "filter",
      placeholder = "Filter histogram...",
      rightElement = Spinner(intent = "primary", size = 20)
    ),
    textOutput(ns("uncontrolledInputGroupOutput")),
    H4("Controlled"),
    InputGroup.shinyInput(
      inputId = ns("controlledInputGroup"),
      disabled = FALSE,
      large = FALSE,
      leftIcon = "home",
      placeholder = "Type something..."
    ),
    textOutput(ns("controlledInputGroupOutput")),
    reactOutput(ns("passwordExample")),
    textOutput(ns("passwordOutput"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns

    output$uncontrolledInputGroupOutput <- renderText(input$uncontrolledInputGroup)
    output$controlledInputGroupOutput <- renderText(input$controlledInputGroup)

    isLocked <- reactiveVal(TRUE)

    observeEvent(input$toggleLock, isLocked(!isLocked()))
    output$passwordOutput <- renderText(input$passwordInput)

    output$passwordExample <- renderReact({
      lockButton <- Button.shinyInput(
        inputId = ns("toggleLock"),
        icon = ifelse(isLocked(), "lock", "unlock"),
        minimal = TRUE,
        intent = "warning"
      )
      InputGroup.shinyInput(
        inputId = ns("passwordInput"),
        disabled = FALSE,
        large = FALSE,
        rightElement = lockButton,
        placeholder = "Enter your password...",
        type = ifelse(isLocked(), "password", "text")
      )
    })
  })
}
```

```
if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

Label *Label*

Description

Documentation: https://blueprintjs.com/docs/#core/components/label

Usage

```
Label(...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

See Also

Other HTML elements: `HTMLTable()`, `htmlElements`

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  Label(
    "Label",
    tags$input(class = "bp5-input")
  )
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

Menu *Menu*

Description

Documentation: https://blueprintjs.com/docs/#core/components/menu

Usage

```
Menu(...)

MenuItem(...)

MenuDivider(...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  Menu(
    style = "max-width: 200px",
    className = "bp5-elevation-1",
    MenuDivider(title = "Edit"),
    MenuItem(icon = "cut", text = "Cut", label = "^X"),
    MenuItem(icon = "duplicate", text = "Copy", label = "^C"),
    MenuItem(icon = "clipboard", text = "Paste", label = "^V", disabled = TRUE),
    MenuDivider(title = "Text"),
    MenuItem(
      icon = "style", text = "Style",
      MenuItem(icon = "bold", text = "Bold"),
      MenuItem(icon = "italic", text = "Italic"),
      MenuItem(icon = "underline", text = "Underline")
    ),
    MenuItem(
      icon = "asterisk", text = "Miscellaneous",
      MenuItem(icon = "badge", text = "Badge"),
     MenuItem(icon = "book", text = "Long items will truncate when they reach max-width"),
      MenuItem(
        icon = "more", text = "Look in here for even more items",
        MenuItem(icon = "briefcase", text = "Briefcase"),
```

```
          MenuItem(icon = "calculator", text = "Calculator"),
          MenuItem(icon = "dollar", text = "Dollar"),
          MenuItem(
            icon = "dot", text = "Shapes",
            MenuItem(icon = "full-circle", text = "Full circle"),
            MenuItem(icon = "heart", text = "Heart"),
            MenuItem(icon = "ring", text = "Ring"),
            MenuItem(icon = "square", text = "Square")
          )
        )
      ),
      MenuDivider(),
      MenuItem(
        icon = "cog", labelElement = Icon(icon = "share"),
        text = "Settings...", intent = "primary"
      )
    )
  )
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`MultiSelect` *MultiSelect*

Description

Documentation: https://blueprintjs.com/docs/#select/multi-select2

Usage

```
MultiSelect(...)

MultiSelect.shinyInput(
  inputId,
  items,
  selected = NULL,
  ...,
  noResults = "No results."
)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.
`inputId`	The `input` slot that will be used to access the value.
`items`	A list of options (character vector or list containing `text` and `label` entries)

selected	Initialy selected item
noResults	Message when no results were found

Value

Object with shiny.tag class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny)
library(shiny.blueprint)

top5Films <- list(
  list(text = "The Shawshank Redemption", label = 1994),
  list(text = "The Godfather", label = 1972),
  list(text = "The Godfather: Part II", label = 1974),
  list(text = "The Dark Knight", label = 2008),
  list(text = "12 Angry Men", label = 1957)
)

ui <- function(id) {
  ns <- NS(id)
  tagList(
    H3("Multiselect"),
    MultiSelect.shinyInput(
      inputId = ns("multiselect"),
      items = paste("Option", LETTERS),
      selected = c("Option B", "Option E"),
      tagInputProps = list(
        tagProps = list(
          intent = "danger"
        )
      )
    ),
    uiOutput(ns("multiselect_output")),
    H3("Multiselect with labels"),
    MultiSelect.shinyInput(
      inputId = ns("multiselect_lab"),
      items = top5Films,
      selected = c("12 Angry Men", "The Godfather")
    ),
    uiOutput(ns("multiselect_lab_output"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$multiselect_output <- renderText({
      paste(
        purrr::map_chr(input$multiselect[[1]], ~ .x$text),
        collapse = ", "
      )
```

```
    })

    output$multiselect_lab_output <- renderText({
      paste(
        purrr::map_chr(input$multiselect_lab[[1]], ~ .x$text),
        collapse = ", "
      )
    })
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`MultiSlider` *Multi slider*

Description

Documentation: https://blueprintjs.com/docs/#core/components/sliders.multi-slider

Usage

```
MultiSlider(...)

MultiSlider.shinyInput(inputId, values, min = NULL, max = NULL, ...)

MultiSliderHandle(...)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.
`inputId`	The `input` slot that will be used to access the value.
`values`	Numeric vector or list containing `value` and other params passed to `MultiSliderHandle`
`min`	Minimal value of the slider
`max`	Maximum value of the slider

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny)
library(shiny.blueprint)

ui <- function(id) {
  ns <- NS(id)
  tagList(
```

```
    reactOutput(ns("multiSlider")),
    textOutput(ns("multiSliderOutput")),
    MultiSlider.shinyInput(
      inputId = ns("multiSliderShiny"),
      values = c(3, 6, 9)
    ),
    textOutput(ns("multiSliderShinyOutput")),
    MultiSlider.shinyInput(
      inputId = ns("multiSliderShiny2"),
      values = list(
        list(value = 3, type = "start", intentBefore = "danger"),
        list(value = 8, type = "start", intentBefore = "warning"),
        list(value = 14, type = "end", intentAfter = "warning"),
        list(value = 17, type = "end", intentAfter = "warning")
      ),
      min = 0,
      max = 20,
      defaultTrackIntent = "success"
    ),
    textOutput(ns("multiSliderShinyOutput2")),
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns

    thresholds <- reactiveValues(
      dangerStart = 3,
      warningStart = 8,
      warningEnd = 14,
      dangerEnd = 17
    )

    observeEvent(input$mutliSliderInput, {
      sliderValues <- sort(input$mutliSliderInput)
      thresholds$dangerStart <- sliderValues[1]
      thresholds$warningStart <- sliderValues[2]
      thresholds$warningEnd <- sliderValues[3]
      thresholds$dangerEnd <- sliderValues[4]
    })

    output$multiSlider <- renderReact({
      MultiSlider(
        defaultTrackIntent = "success",
        onChange = setInput(ns("mutliSliderInput")),
        stepSize = 1,
        min = 0,
        max = 20,
        MultiSliderHandle(
          type = "start",
          intentBefore = "danger",
          value = thresholds$dangerStart,
```

```
            interactionKind = "push"
          ),
          MultiSliderHandle(
            type = "start",
            intentBefore = "warning",
            value = thresholds$warningStart,
            interactionKind = "push"
          ),
          MultiSliderHandle(
            type = "end",
            intentAfter = "warning",
            value = thresholds$warningEnd,
            interactionKind = "push"
          ),
          MultiSliderHandle(
            type = "end",
            intentAfter = "danger",
            value = thresholds$dangerEnd,
            interactionKind = "push"
          )
        )
      })
      output$multiSliderOutput <- renderText(
        paste(
          thresholds$dangerStart,
          thresholds$warningStart,
          thresholds$warningEnd,
          thresholds$dangerEnd,
          sep = ", "
        )
      )
      output$multiSliderShinyOutput <- renderText(
        paste(input$multiSliderShiny, collapse = ", ")
      )
      output$multiSliderShinyOutput2 <- renderText(
        paste(input$multiSliderShiny2, collapse = ", ")
      )
    })
  }

  if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`MultistepDialog` *Multistep dialog*

Description

Documentation: https://blueprintjs.com/docs/#core/components/dialog.multistep-dialog

Usage

```
MultistepDialog(...)

DialogStep(...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  ns <- NS(id)
  tagList(
    Button.shinyInput(
      inputId = ns("showMultistepDialog"),
      "Show multistep dialog"
    ),
    reactOutput(ns("multistepDialog"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns

    isOpen <- reactiveVal(FALSE)
    observeEvent(input$showMultistepDialog, isOpen(TRUE))
    observeEvent(input$closeMultistepDialog, isOpen(FALSE))

    output$multistepDialog <- renderReact({
      MultistepDialog(
        usePortal = FALSE,
        isOpen = isOpen(),
        title = "Multistep dialog",
        onClose = triggerEvent(ns("closeMultistepDialog")),
        DialogStep(
          id = "step1",
          panel = div(
            className = "bp5-dialog-body",
            p("This is a step 1")
          ),
          title = "Step 1"
        ),
        DialogStep(
```

```
            id = "step2",
            panel = div(
              className = "bp5-dialog-body",
              p("This is a step 2")
            ),
            title = "Step 2"
          ),
          DialogStep(
            id = "step3",
            panel = div(
              className = "bp5-dialog-body",
              p("This is a step 3")
            ),
            title = "Step 3"
          )
        )
      })
    })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

Navbar *Navbar*

Description

Documentation: https://blueprintjs.com/docs/#core/components/navbar

Usage

```
Navbar(...)

NavbarGroup(...)

NavbarHeading(...)

NavbarDivider(...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  Navbar(
    NavbarGroup(
      NavbarHeading("Blueprint"),
      NavbarDivider(),
      Button(minimal = TRUE, icon = "home", text = "Home"),
      Button(minimal = TRUE, icon = "document", text = "Files")
    ),
    NavbarGroup(
      align = "right",
      Button(minimal = TRUE, icon = "user"),
      Button(minimal = TRUE, icon = "refresh")
    )
  )
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

NonIdealState *Non-ideal state*

Description

Documentation: https://blueprintjs.com/docs/#core/components/non-ideal-state

Usage

```
NonIdealState(...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  NonIdealState(
    icon = "search",
    title = "No search results",
    description = Card(
      "Your search didn't match any files.",
      tags$br(),
      "Try searching for something else, or create a new file."
    ),
    action = Button(icon = "plus", text = "New file", intent = "primary", outlined = TRUE)
  )
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`NumericInput` *NumericInput*

Description

Documentation: https://blueprintjs.com/docs/#core/components/numeric-input

Usage

```
NumericInput(...)

NumericInput.shinyInput(inputId, ..., value = defaultValue)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.
`inputId`	The `input` slot that will be used to access the value.
`value`	Initial value.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny)
library(shiny.blueprint)

ui <- function(id) {
  ns <- NS(id)
  tagList(
    NumericInput(
      onValueChange = setInput(ns("value1")),
      intent = "primary"
    ),
    textOutput(ns("value1Output")),
    NumericInput.shinyInput(
      inputId = ns("value2"),
      intent = "primary"
    ),
    textOutput(ns("value2Output"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$value1Output <- renderText(input$value1)
    output$value2Output <- renderText(input$value2)
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

OverflowList *Overflow list*

Description

Documentation: https://blueprintjs.com/docs/#core/components/overflow-list

Usage

```
OverflowList(...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

boxStyle <- tags$style("
  .box {
    margin: 0.5em;
    padding: 0.5em;
    background: silver;
    font-size: 4em;
  }
")

items <- lapply(
  list("Too", "many", "words", "to", "fit", "on", "your", "screen!"),
  function(text) div(text, class = "box")
)

ui <- function(id) {
  tagList(
    boxStyle,
    OverflowList(
      items = items,
      visibleItemRenderer = JS("item => item"),
      overflowRenderer = JS("items => null"),
      collapseFrom = "end"
    )
  )
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`Overlay` *Overlay*

Description

Documentation: https://blueprintjs.com/docs/#core/components/overlay

Usage

```
Overlay(...)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  ns <- NS(id)
  tagList(
    Button.shinyInput(
      inputId = ns("showOverlay"),
      "Show overlay"
    ),
    reactOutput(ns("overlay"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns

    isOpen <- reactiveVal(FALSE)
    observeEvent(input$showOverlay, isOpen(TRUE))
    observeEvent(input$closeOverlay, isOpen(FALSE))

    output$overlay <- renderReact({
      Overlay(
        usePortal = FALSE,
        isOpen = isOpen(),
        onClose = triggerEvent(ns("closeOverlay")),
        Card(
          className = "bp5-elevation-4 bp5-dark bp5-overlay-content",
          interactive = TRUE,
          H5("Analytical applications"),
          tags$p(
            "User interfaces that enable people to interact smoothly with data,",
            " ask better questions, and make better decisions."
          ),
          Button.shinyInput(
            inputId = ns("closeOverlay"),
            "Close"
          )
        )
      )
    })
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

PanelStack *Panel stack*

Description

Documentation: https://blueprintjs.com/docs/#core/components/panel-stack2

Usage

```
PanelStack(...)

PanelStack.shinyWrapper(panels, ns = "ps", size = c(300, 250), ...)

openPanel(panelId, ns = "ps")

closePanel(ns = "ps")
```

Arguments

...	Component props and children. See the official Blueprint docs for details.
panels	List of lists - each list contains `title` (string) and `content` (HTML)
ns	Namespace of given panel stack (required if there's more than 1 panel stack)
size	Numeric vector of length 2 - `c(width, height)`
panelId	Id of the panel to be closed

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

customComponents <- tagList(
  tags$style("
    .panel-stack {
      border: 1px solid lightgrey;
      width: 300px;
      height: 240px;
    }
    .panel {
      position: absolute;
      top: 50%;
      left: 50%;
      transform: translate(-50%, -50%);
    }
  "),
```

```
  tags$script(HTML("(() => {
    const React = jsmodule['react'];
    const Blueprint = jsmodule['@blueprintjs/core'];

    function createPanel(num) {
      return {
        title: `Panel ${num}`,
        renderPanel: Panel,
        props: { num },
      };
    }

    function Panel({ num, openPanel }) {
      const button = React.createElement(
        Blueprint.Button,
        {
          onClick: () => openPanel(createPanel(num + 1)),
          intent: Blueprint.Intent.PRIMARY,
        },
        'Open Panel'
      )
      return React.createElement('div', { className: 'panel' }, button);
    }

    window.createPanel = createPanel;
  })()"))
)

ui <- function(id) {
  tagList(
    customComponents,
    PanelStack(
      className = "panel-stack",
      initialPanel = JS("createPanel(1)")
    ),
    PanelStack.shinyWrapper(
      panels = list(
        list(id = "panel1", title = "Panel 1", content = div(
          class = "panel",
          Button(text = "Open 2", onClick = openPanel("panel2")),
          Button(text = "Open 4", onClick = openPanel("panel4"))
        )),
        list(id = "panel2", title = "Panel 2", content = div(
          class = "panel",
          Button(text = "Open 3", onClick = openPanel("panel3")),
          Button(text = "Close", onClick = closePanel())
        )),
        list(id = "panel3", title = "Panel 3", content = div(
          class = "panel",
          Button(text = "Open 4", onClick = openPanel("panel4")),
          Button(text = "Close", onClick = closePanel())
        )),
        list(id = "panel4", title = "Panel 4", content = div(
```

```
          class = "panel",
          Button(text = "Close", onClick = closePanel())
        ))
      )
    )
  )
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

Popover *Popover*

Description

Documentation: https://blueprintjs.com/docs/#core/components/popover

Usage

```
Popover(...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  ns <- NS(id)
  reactOutput(ns("ui"))
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns

    isOpen <- reactiveVal(FALSE)
    observeEvent(input$hello, isOpen(TRUE))
    observeEvent(input$dismiss, isOpen(FALSE))
```

```
    output$ui <- renderReact({
      Popover(
        isOpen = isOpen(),
        Button.shinyInput(ns("hello"), "Say Hello", intent = "primary"),
        usePortal = FALSE,
        content = tags$div(
          style = "padding: 1em",
          H5("Hello!"),
          tags$p("Please read this message."),
          Button.shinyInput(ns("dismiss"), "Dismiss")
        )
      )
    })
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`ProgressBar` *Progress bar*

Description

Documentation: https://blueprintjs.com/docs/#core/components/progress-bar

Usage

```
ProgressBar(...)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  ProgressBar(animate = TRUE)
}

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

```
if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

Radio *Radio*

Description

Documentation: https://blueprintjs.com/docs/#core/components/radio

Usage

```
Radio(...)

RadioGroup(...)

RadioGroup.shinyInput(inputId, ..., value = defaultValue)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.
`inputId`	The `input` slot that will be used to access the value.
`value`	Initial value.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

setInput <- function(inputId, accessor = NULL) {
  JS(paste0("x => Shiny.setInputValue('", inputId, "', x", accessor, ")"))
}

ui <- function(id) {
  ns <- NS(id)
  tagList(
    H3("Favorite animal"),
    RadioGroup.shinyInput(
      inputId = ns("animal"),
      value = "dog",
      Radio(label = "Cat", value = "cat"),
      Radio(label = "Dog", value = "dog")
    ),
    textOutput(ns("favoriteAnimal")),
    H3("Favorite fruit"),
```

```
    reactOutput(ns("fruitRadio")),
    textOutput(ns("favoriteFruit"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns

    output$favoriteAnimal <- renderText(deparse(input$animal))

    fruit <- reactiveVal()
    observeEvent(input$fruit, fruit(input$fruit))
    output$fruitRadio <- renderReact({
      RadioGroup(
        onChange = setInput(ns("fruit"), ".currentTarget.value"),
        selectedValue = fruit(),
        Radio(label = "Apple", value = "a"),
        Radio(label = "Banana", value = "b"),
        Radio(label = "Cherry", value = "c")
      )
    })
    output$favoriteFruit <- renderText(deparse(fruit()))
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`RangeSlider` *Range slider*

Description

Documentation: https://blueprintjs.com/docs/#core/components/sliders.range-slider

Usage

```
RangeSlider(...)

RangeSlider.shinyInput(inputId, ..., value = defaultValue)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.
`inputId`	The `input` slot that will be used to access the value.
`value`	Initial value.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny)
library(shiny.blueprint)

ui <- function(id) {
  ns <- NS(id)
  tagList(
    Slider.shinyInput(
      inputId = ns("value"),
      min = 0,
      max = 10,
      stepSize = 0.1,
      labelStepSize = 10
    ),
    textOutput(ns("valueOutput"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$valueOutput <- renderText(input$value)
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

ResizeSensor *Resize sensor*

Description

Documentation: https://blueprintjs.com/docs/#core/components/resize-sensor

Usage

```
ResizeSensor(...)

ResizeSensor.shinyInput(inputId, ...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.
inputId	The input slot that will be used to access the value.

Value

Object with shiny.tag class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

setInput <- function(inputId, accessor = NULL) {
  JS(paste0(
    "x => Shiny.setInputValue('", inputId, "', x", accessor, ")"
  ))
}

printSize <- function(content) {
  paste0(content$width, "x", content$height)
}

ui <- function(id) {
  ns <- NS(id)
  tagList(
    tags$style("
      .resizable {
        overflow: auto;
        resize: both;
        width: 100px;
        height: 100px;
        background: silver;
      }
    "),
    ResizeSensor(
      onResize = setInput(ns("resize"), "[0].contentRect"),
      div(
        class = "resizable",
        textOutput(ns("size"))
      )
    ),
    ResizeSensor.shinyInput(
      inputId = ns("resizeSensor"),
      content = div(
        textOutput(ns("resizeSensorInput")),
        style = "
          border: 1px solid black;
          width: 100px;
        "
      )
    )
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$size <- renderText({
      content <- req(input$resize)
      printSize(content)
    })
```

```
    output$resizeSensorInput <- renderText({
      content <- req(input$resizeSensor)
      printSize(content)
    })
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`runExample` *Run example*

Description

Launch a Shiny example app or list the available examples. Use `shiny.blueprint::runExample("showcase")` to run a showcase app with all the components.

Usage

```
runExample(example = NULL, ...)
```

Arguments

`example`	The name of the example to run, or `NULL` to retrieve the list of examples.
`...`	Additional arguments to pass to `shiny::runApp()`.

Value

This function normally does not return; interrupt R to stop the application (usually by pressing Ctrl+C or Esc).

`Select` *Select*

Description

Documentation: https://blueprintjs.com/docs/#select/select2

Usage

```
Select(...)

Select.shinyInput(
  inputId,
  items,
  selected = NULL,
  ...,
  noResults = "No results."
)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.
inputId	The input slot that will be used to access the value.
items	A list of options (character vector or list containing text and label entries)
selected	Initialy selected item
noResults	Message when no results were found

Value

Object with shiny.tag class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny)
library(shiny.blueprint)

top5Films <- list(
  list(text = "The Shawshank Redemption", label = 1994),
  list(text = "The Godfather", label = 1972),
  list(text = "The Godfather: Part II", label = 1974),
  list(text = "The Dark Knight", label = 2008),
  list(text = "12 Angry Men", label = 1957)
)

ui <- function(id) {
  ns <- NS(id)
  tagList(
    H3("Select"),
    Select.shinyInput(
      inputId = ns("select"),
      items = paste("Option", LETTERS),
      selected = "Option C",
      noResults = "No options."
    ),
    uiOutput(ns("select_output")),
    H3("Select with labels"),
    Select.shinyInput(
      inputId = ns("select_lab"),
      items = top5Films,
      selected = "The Dark Knight"
    ),
    uiOutput(ns("select_lab_output"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$select_output <- renderText(input$select$text)
    output$select_lab_output <- renderText(input$select_lab$text)
  })
```

```
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`Slider` *Slider*

Description

Documentation: https://blueprintjs.com/docs/#core/components/sliders.slider

Usage

```
Slider(...)

Slider.shinyInput(inputId, ..., value = defaultValue)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.
`inputId`	The `input` slot that will be used to access the value.
`value`	Initial value.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny)
library(shiny.blueprint)

ui <- function(id) {
  ns <- NS(id)
  tagList(
    Slider.shinyInput(
      inputId = ns("value"),
      min = 0,
      max = 10,
      stepSize = 0.1,
      labelStepSize = 10
    ),
    textOutput(ns("valueOutput"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$valueOutput <- renderText(input$value)
```

```
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

Spinner *Spinner*

Description

Documentation: https://blueprintjs.com/docs/#core/components/spinner

Usage

```
Spinner(...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  Spinner(intent = "primary", size = 100)
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`Suggest` *Suggest*

Description

Documentation: https://blueprintjs.com/docs/#select/suggest2

Usage

```
Suggest(...)

Suggest.shinyInput(
  inputId,
  items,
  selected = NULL,
  ...,
  noResults = "No results."
)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.
`inputId`	The `input` slot that will be used to access the value.
`items`	A list of options (character vector or list containing `text` and `label` entries)
`selected`	Initialy selected item
`noResults`	Message when no results were found

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny)
library(shiny.blueprint)

top5Films <- list(
  list(text = "The Shawshank Redemption", label = 1994),
  list(text = "The Godfather", label = 1972),
  list(text = "The Godfather: Part II", label = 1974),
  list(text = "The Dark Knight", label = 2008),
  list(text = "12 Angry Men", label = 1957)
)

ui <- function(id) {
  ns <- NS(id)
  tagList(
```

```
    H3("Suggest"),
    Suggest.shinyInput(
      inputId = ns("suggest"),
      items = paste("Option", LETTERS),
      inputProps = list(
        placeholder = "Search with Suggest..."
      )
    ),
    uiOutput(ns("suggest_output")),
    H3("Suggest with labels"),
    Suggest.shinyInput(
      inputId = ns("suggest_lab"),
      items = top5Films,
      noResults = "No suggestions."
    ),
    uiOutput(ns("suggest_lab_output"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$suggest_output <- renderText(input$suggest$text)
    output$suggest_lab_output <- renderText(input$suggest_lab$text)
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`Switch` *Switch*

Description

Documentation: https://blueprintjs.com/docs/#core/components/switch

Usage

```
Switch(...)

Switch.shinyInput(inputId, ..., value = defaultValue)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.
`inputId`	The `input` slot that will be used to access the value.
`value`	Initial value.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

setInput <- function(inputId, accessor = NULL) {
  JS(paste0("x => Shiny.setInputValue('", inputId, "', x", accessor, ")"))
}

ui <- function(id) {
  ns <- NS(id)
  tagList(
    Switch(
      onChange = setInput(ns("apples"), ".target.checked"),
      defaultChecked = TRUE,
      label = "Apples"
    ),
    Switch.shinyInput(
      inputId = ns("bananas"),
      value = TRUE,
      label = "Bananas"
    ),
    textOutput(ns("applesEnabled")),
    textOutput(ns("bananasEnabled"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$applesEnabled <- renderText(paste("Apples:", deparse(input$apples)))
    output$bananasEnabled <- renderText(paste("Bananas:", deparse(input$bananas)))
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

Tabs *Tabs*

Description

Documentation: https://blueprintjs.com/docs/#core/components/tabs

Usage

```
Tabs(...)

Tab(...)

TabsExpander(...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  ns <- NS(id)
  reactOutput(ns("tabs"))
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns

    currentTab <- reactiveVal("react")
    observeEvent(input$selectTab, currentTab(input$selectTab))
    output$tabs <- renderReact(
      Tabs(
        selectedTabId = currentTab(),
        onChange = setInput(ns("selectTab")),
        Tab(id = "angular", title = "Angular", panel = "Angular"),
        Tab(id = "ember", title = "Ember", panel = "Ember"),
        Tab(id = "react", title = "React", panel = "React"),
        TabsExpander(),
        tags$input(class = "bp5-input", type = "text", placeholder = "Search...")
      )
    )
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

Tag *Tag*

Description

Documentation: https://blueprintjs.com/docs/#core/components/tag

Usage

```
Tag(...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  tagList(
    Tag(active = TRUE, "Hello"),
    Tag(active = TRUE, large = TRUE, "Hello"),
    Tag(active = TRUE, round = TRUE, "Hello"),
    Tag(active = FALSE, icon = "home", round = TRUE, large = TRUE, "Hello"),
    Tag(active = TRUE, rightIcon = "home", "Hello"),
    Tag(active = TRUE, round = TRUE, intent = "primary", interactive = TRUE, "Hello"),
    Tag(active = TRUE, round = TRUE, intent = "warning", interactive = TRUE, "Hello"),
    Tag(active = TRUE, round = TRUE, intent = "success", interactive = TRUE, "Hello"),
    Tag(active = TRUE, round = TRUE, intent = "danger", interactive = TRUE, "Hello")
  )
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`TagInput` *TagInput*

Description

Documentation: https://blueprintjs.com/docs/#core/components/tag-input

Usage

```
TagInput(...)

TagInput.shinyInput(inputId, ..., value = defaultValue)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.
inputId	The `input` slot that will be used to access the value.
value	Initial value.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny)
library(shiny.blueprint)


ui <- function(id) {
  ns <- NS(id)
  tagList(
    TagInput.shinyInput(
      inputId = ns("value"),
      value = c("one", "two", "three")
    ),
    textOutput(ns("valueOutput"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$valueOutput <- renderText(input$value)
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

Text *Text*

Description

Documentation: https://blueprintjs.com/docs/#core/components/text

Usage

```
Text(...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

ui <- function(id) {
  Text(
    "Lorem ipsum dolor sit amet,
    consectetur adipiscing elit,
    sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
    Ut enim ad minim veniam,
    quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
    Duis aute irure dolor in reprehenderit
    in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
    Excepteur sint occaecat cupidatat non proident,
    sunt in culpa qui officia deserunt mollit anim id est laborum."
  )
}

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

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

TextArea *Text area*

Description

Documentation: https://blueprintjs.com/docs/#core/components/text-inputs.text-area

Usage

```
TextArea(...)
```

Arguments

`...`	Component props and children. See the official Blueprint docs for details.

Value

Object with `shiny.tag` class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(shiny)

setInput <- function(inputId, accessor = NULL) {
```

```
  JS(paste0("x => Shiny.setInputValue('", inputId, "', x", accessor, ")"))
}

ui <- function(id) {
  ns <- NS(id)
  tagList(
    H4("Uncontrolled"),
    TextArea(
      growVertically = TRUE,
      onChange = setInput(ns("uncontrolledTextarea"), ".target.value"),
      large = TRUE,
      intent = "primary"
    ),
    textOutput(ns("uncontrolledTextareaOutput")),
    H4("Controlled"),
    TextArea.shinyInput(
      inputId = ns("controlledTextarea"),
      growVertically = TRUE,
      large = TRUE,
      intent = "primary"
    ),
    textOutput(ns("controlledTextareaOutput"))
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$uncontrolledTextareaOutput <- renderText(input$uncontrolledTextarea)
    output$controlledTextareaOutput <- renderText(input$controlledTextarea)
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

`Toaster` *Toaster*

Description

Documentation: https://blueprintjs.com/docs/#core/components/toast

Methods

Public methods:

- `Toaster$new()`
- `Toaster$show()`
- `Toaster$clear()`
- `Toaster$dismiss()`

Method `new()`:

Usage:

```
Toaster$new(
  toasterId = incrementToasterId(),
  session = shiny::getDefaultReactiveDomain(),
  ...
)
```

Arguments:

`toasterId` Unique number - needed to use more than one toaster

`session` Shiny session object

`...` Parameters passed to `Toaster` component

Returns: A new `Toaster` instance.

Method `show()`**:** Shows a new toast to the user, or updates an existing toast corresponding to the provided key

Usage:

```
Toaster$show(..., key = NULL)
```

Arguments:

`...` Parameters passed to `Toaster` component

`key` A key of toast to be shown/dismissed

Returns: Nothing. This method is called for side effects.

Method `clear()`**:** Dismiss all toasts instantly

Usage:

```
Toaster$clear()
```

Returns: Nothing. This method is called for side effects.

Method `dismiss()`**:** Dismiss the given toast instantly

Usage:

```
Toaster$dismiss(key)
```

Arguments:

`key` A key of toast to be shown/dismissed

Returns: Nothing. This method is called for side effects.

`Tree` *Tree*

Description

Documentation: https://blueprintjs.com/docs/#core/components/tree

Usage

```
Tree(...)

Tree.shinyInput(inputId, data, ...)
```

Arguments

...	Component props and children. See the official Blueprint docs for details.
inputId	The input slot that will be used to access the value.
data	A list of nodes parameters: • required: label • optional: childNodes, icon, hasCaret, isExpanded, disabled, secondaryLabel

Value

Object with shiny.tag class suitable for use in the UI of a Shiny app.

Examples

```
library(shiny.blueprint)
library(purrr)
library(shiny)

treeList <- list(
  list(
    id = "0",
    hasCaret = TRUE,
    icon = "folder-close",
    label = "Tree"
  ),
  list(
    id = "1",
    icon = "folder-close",
    isExpanded = TRUE,
    label = "Hello here",
    childNodes = list(
      list(
        id = "2",
        icon = "document",
        label = "Item 0",
        secondaryLabel = Icon(icon = "eye-open")
      ),
      list(
        id = "3",
        icon = "tag",
       label = "Organic meditation gluten-free, sriracha VHS drinking vinegar beard man.",
        childNodes = list(
          list(
            id = "4",
            icon = "document",
```

```
            label = "Item 0",
            secondaryLabel = Icon(icon = "eye-open")
          ),
          list(
            id = "5",
            icon = "tag",
            label = "Some other stuff"
          )
        )
      )
    )
  ),
  list(
    id = "10",
    hasCaret = TRUE,
    icon = "folder-close",
    label = "Super secret files",
    disabled = TRUE
  )
)

modifyTree <- function(tree, ids, props) {
  if (!is.null(tree)) purrr::map(tree, function(node) {
    if (node$id %in% ids) {
      node <- purrr::list_modify(node, !!!props)
    }
    node$childNodes <- modifyTree(node$childNodes, ids, props)
    node
  })
}

ui <- function(id) {
  ns <- NS(id)
  tagList(
    reactOutput(ns("tree")),
    Divider(),
    reactOutput(ns("info")),
    Divider(),
    Tree.shinyInput(
      inputId = ns("selected_nodes"),
      data = list(
        list(
          label = "1",
          id = "1",
          isExpanded = TRUE,
          childNodes = list(
            list(
              label = "1.1",
              id = "1.1",
              childNodes = list(list(label = "1.1.1", id = "1.1.1"))
            ),
            list(label = "1.2", id = "1.2")
          )
```

```
        ),
        list(
          label = "2",
          id = "2",
          childNodes = list(
            list(label = "2.1", id = "2.1")
          )
        ),
        list(label = "3", id = "3", hasCaret = TRUE)
      )
    ),
    Divider(),
    tags$span("Hold ", tags$b("shift"), " to select multiple nodes."),
    reactOutput(ns("selected_nodes_list")),
  )
}

server <- function(id) {
  moduleServer(id, function(input, output, session) {
    ns <- session$ns

    treeReactive <- reactiveVal(treeList)
    observeEvent(input$expand, {
      treeReactive(
        modifyTree(treeReactive(), ids = input$expand, props = list(isExpanded = TRUE))
      )
    })
    observeEvent(input$collapse, {
      treeReactive(
       modifyTree(treeReactive(), ids = input$collapse, props = list(isExpanded = FALSE))
      )
    })

    output$tree <- renderReact({
      Tree(
        contents = treeReactive(),
        onNodeExpand = setInput(ns("expand"), jsAccessor = "[0].id"),
        onNodeCollapse = setInput(ns("collapse"), jsAccessor = "[0].id"),
        onNodeClick = setInput(ns("click"), jsAccessor = "[0].id")
      )
    })

    output$info <- renderReact({
      tags$div("Clicked (id): ", input$click)
    })

    output$selected_nodes_list <- renderReact({
      UL(lapply(input$selected_nodes, function(node) tags$li(node)))
    })
  })
}

if (interactive()) shinyApp(ui("app"), function(input, output) server("app"))
```

Index

∗ **HTML elements**
htmlElements, 18
HTMLTable, 21
Label, 25

Alert, 3
AnchorButton *(*Button*)*, 5

Blockquote *(*htmlElements*)*, 18
Breadcrumbs, 4
Button, 5
ButtonGroup, 6

Callout, 7
Card, 8
Checkbox, 9
closePanel *(*PanelStack*)*, 39
Code *(*htmlElements*)*, 18
Collapse, 10
ControlGroup, 11

Dialog, 12
DialogStep *(*MultistepDialog*)*, 31
Divider, 13
Drawer, 14

EditableText, 15

FileInput, 16
FormGroup, 17

H1 *(*htmlElements*)*, 18
H2 *(*htmlElements*)*, 18
H3 *(*htmlElements*)*, 18
H4 *(*htmlElements*)*, 18
H5 *(*htmlElements*)*, 18
H6 *(*htmlElements*)*, 18
htmlElements, 18, *21*, *25*
HTMLSelect, 20
HTMLTable, *19*, 21, *25*

Icon, 22
InputGroup, 23

Label, *19*, *21*, 25

Menu, 26
MenuDivider *(*Menu*)*, 26
MenuItem *(*Menu*)*, 26
MultiSelect, 27
MultiSlider, 29
MultiSliderHandle *(*MultiSlider*)*, 29
MultistepDialog, 31

Navbar, 33
NavbarDivider *(*Navbar*)*, 33
NavbarGroup *(*Navbar*)*, 33
NavbarHeading *(*Navbar*)*, 33
NonIdealState, 34
NumericInput, 35

OL *(*htmlElements*)*, 18
openPanel *(*PanelStack*)*, 39
OverflowList, 36
Overlay, 37

PanelStack, 39
Popover, 41
Pre *(*htmlElements*)*, 18
ProgressBar, 42

Radio, 43
RadioGroup *(*Radio*)*, 43
RangeSlider, 44
ResizeSensor, 45
runExample, 47

Select, 47
Slider, 49
Spinner, 50
Suggest, 51
Switch, 52

Tab *(*Tabs*)*, 53
Tabs, 53
TabsExpander *(*Tabs*)*, 53
Tag, 54
TagInput, 55
Text, 56
TextArea, 57
TextArea.shinyInput *(*InputGroup*)*, 23
Toaster, 58
Tree, 59

UL *(*htmlElements*)*, 18