

Package ‘shinybootstrap2’

August 29, 2016

Title Bootstrap 2 Web Components for Use with Shiny

Version 0.2.1

Description Provides Bootstrap 2 web components for use with the Shiny package. With versions of Shiny prior to 0.11, these Bootstrap 2 components were included as part of the package. Later versions of Shiny include Bootstrap 3, so the Bootstrap 2 components have been moved into this package for those uses who rely on features specific to Bootstrap 2.

Depends R (>= 3.0.0)

License GPL-3 | file LICENSE

LazyData true

Imports htmltools (>= 0.2.6), jsonlite (>= 0.9.12), shiny

Author Winston Chang [aut, cre],
RStudio [cph],
Mark Otto [ctb] (Bootstrap library),
Jacob Thornton [ctb] (Bootstrap library),
Bootstrap contributors [ctb] (Bootstrap library; authors listed at <https://github.com/twbs/bootstrap/graphs/contributors>),
Twitter, Inc [cph] (Bootstrap library),
Brian Reavis [ctb, cph] (selectize.js library),
Egor Khmelev [ctb, cph] (jslider library),
SpryMedia Limited [ctb, cph] (DataTables library)

Maintainer Winston Chang <winston@rstudio.com>

NeedsCompilation no

Repository CRAN

Date/Publication 2015-02-11 19:57:37

R topics documented:

shinybootstrap2-package	2
bootstrapPage	3
checkboxGroupInput	4
checkboxInput	5

column	5
dataTableOutput	6
dateInput	7
dateRangeInput	9
fixedPage	11
fluidPage	12
headerPanel	14
icon	14
mainPanel	15
navbarPage	16
navlistPanel	17
numericInput	18
pageWithSidebar	19
radioButtons	20
selectInput	21
sidebarLayout	22
sidebarPanel	23
sliderInput	24
submitButton	26
textInput	27
titlePanel	27
updateCheckboxGroupInput	28
updateRadioButtons	29
updateSliderInput	31
verticalLayout	32
withBootstrap2	33
Index	35

shinybootstrap2-package

Bootstrap 2 components for use with Shiny

Description

Bootstrap 2 components for use with Shiny

bootstrapPage *Create a Bootstrap page*

Description

Create a Shiny UI page that loads the CSS and JavaScript for **Bootstrap**, and has no content in the page body (other than what you provide).

Usage

```
bootstrapPage(..., title = NULL, responsive = TRUE, theme = NULL)
```

```
basicPage(...)
```

Arguments

...	The contents of the document body.
title	The browser window title (defaults to the host URL of the page)
responsive	TRUE to use responsive layout (automatically adapt and resize page elements based on the size of the viewing device)
theme	Alternative Bootstrap stylesheet (normally a css file within the www directory, e.g. www/bootstrap.css)

Details

This function is primarily intended for users who are proficient in HTML/CSS, and know how to lay out pages in Bootstrap. Most applications should use [fluidPage](#) along with layout functions like [fluidRow](#) and [sidebarLayout](#).

Value

A UI definition that can be passed to the [shinyUI](#) function.

Note

The `basicPage` function is deprecated, you should use the [fluidPage](#) function instead.

See Also

[fluidPage](#), [fixedPage](#)

checkboxGroupInput *Checkbox Group Input Control*

Description

Create a group of checkboxes that can be used to toggle multiple choices independently. The server will receive the input as a character vector of the selected values.

Usage

```
checkboxGroupInput(inputId, label, choices, selected = NULL, inline = FALSE)
```

Arguments

inputId	Input variable to assign the control's value to.
label	Display label for the control, or NULL.
choices	List of values to show checkboxes for. If elements of the list are named then that name rather than the value is displayed to the user.
selected	The values that should be initially selected, if any.
inline	If TRUE, render the choices inline (i.e. horizontally)

Value

A list of HTML elements that can be added to a UI definition.

See Also

[checkboxInput](#), [updateCheckboxGroupInput](#)

Other input.elements: [animationOptions](#), [sliderInput](#); [checkboxInput](#); [dateInput](#); [dateRangeInput](#); [numericInput](#); [radioButtons](#); [selectInput](#), [selectizeInput](#); [submitButton](#); [textInput](#)

Examples

```
checkboxGroupInput("variable", "Variable:",  
                c("Cylinders" = "cyl",  
                  "Transmission" = "am",  
                  "Gears" = "gear"))
```

checkboxInput	<i>Checkbox Input Control</i>
---------------	-------------------------------

Description

Create a checkbox that can be used to specify logical values.

Usage

```
checkboxInput(inputId, label, value = FALSE)
```

Arguments

inputId	Input variable to assign the control's value to.
label	Display label for the control.
value	Initial value (TRUE or FALSE).

Value

A checkbox control that can be added to a UI definition.

See Also

[checkboxGroupInput](#), [updateCheckboxInput](#)

Other input.elements: [animationOptions](#), [sliderInput](#); [checkboxGroupInput](#); [dateInput](#); [dateRangeInput](#); [numericInput](#); [radioButtons](#); [selectInput](#), [selectizeInput](#); [submitButton](#); [textInput](#)

Examples

```
checkboxInput("outliers", "Show outliers", FALSE)
```

column	<i>Create a column within a UI definition</i>
--------	---

Description

Create a column for use within a [fluidRow](#) or [fixedRow](#)

Usage

```
column(width, ..., offset = 0)
```

Arguments

width	The grid width of the column (must be between 1 and 12)
...	Elements to include within the column
offset	The number of columns to offset this column from the end of the previous column.

Value

A column that can be included within a `fluidRow` or `fixedRow`.

See Also

`fluidRow`, `fixedRow`.

Examples

```
library(shiny)

fluidRow(
  column(4,
    sliderInput("obs", "Number of observations:",
      min = 1, max = 1000, value = 500)
  ),
  column(8,
    plotOutput("distPlot")
  )
)

fluidRow(
  column(width = 4,
    "4"
  ),
  column(width = 3, offset = 2,
    "3 offset 2"
  )
)
```

dataTableOutput	<i>Create a table output element</i>
-----------------	--------------------------------------

Description

Render a `renderDataTable` within an application page. `renderDataTable` uses the DataTables Javascript library to create an interactive table with more features.

Usage

```
dataTableOutput(outputId)
```

Arguments

outputId output variable to read the table from

Value

A table output element that can be included in a panel

Examples

```
## Only run this example in interactive R sessions
if (interactive()) {
  library(shiny)
  shinyApp(
    ui = bootstrapPage(
      dataTableOutput('table')
    ),
    server = function(input, output) {
      output$table <- renderDataTable(iris)
    }
  )
}
```

dateInput

Create date input

Description

Creates a text input which, when clicked on, brings up a calendar that the user can click on to select dates.

Usage

```
dateInput(inputId, label, value = NULL, min = NULL, max = NULL,
  format = "yyyy-mm-dd", startview = "month", weekstart = 0,
  language = "en")
```

Arguments

inputId	Input variable to assign the control's value to.
label	Display label for the control, or NULL.
value	The starting date. Either a Date object, or a string in yyyy-mm-dd format. If NULL (the default), will use the current date in the client's time zone.
min	The minimum allowed date. Either a Date object, or a string in yyyy-mm-dd format.
max	The maximum allowed date. Either a Date object, or a string in yyyy-mm-dd format.

format	The format of the date to display in the browser. Defaults to "yyyy-mm-dd".
startview	The date range shown when the input object is first clicked. Can be "month" (the default), "year", or "decade".
weekstart	Which day is the start of the week. Should be an integer from 0 (Sunday) to 6 (Saturday).
language	The language used for month and day names. Default is "en". Other valid values include "bg", "ca", "cs", "da", "de", "el", "es", "fi", "fr", "he", "hr", "hu", "id", "is", "it", "ja", "kr", "lt", "lv", "ms", "nb", "nl", "pl", "pt", "pt-BR", "ro", "rs", "rs-latin", "ru", "sk", "sl", "sv", "sw", "th", "tr", "uk", "zh-CN", and "zh-TW".

Details

The date format string specifies how the date will be displayed in the browser. It allows the following values:

- yy Year without century (12)
- yyyy Year with century (2012)
- mm Month number, with leading zero (01-12)
- m Month number, without leading zero (01-12)
- M Abbreviated month name
- MM Full month name
- dd Day of month with leading zero
- d Day of month without leading zero
- D Abbreviated weekday name
- DD Full weekday name

See Also

[dateRangeInput](#), [updateDateInput](#)

Other `input.elements`: [animationOptions](#), [sliderInput](#); [checkboxGroupInput](#); [checkboxInput](#); [dateRangeInput](#); [numericInput](#); [radioButtons](#); [selectInput](#), [selectizeInput](#); [submitButton](#); [textInput](#)

Examples

```
dateInput("date", "Date:", value = "2012-02-29")

# Default value is the date in client's time zone
dateInput("date", "Date:")

# value is always yyyy-mm-dd, even if the display format is different
dateInput("date", "Date:", value = "2012-02-29", format = "mm/dd/yy")

# Pass in a Date object
dateInput("date", "Date:", value = Sys.Date()-10)
```



```
# Use different language and different first day of week
dateInput("date", "Date:",
  language = "de",
  weekstart = 1)

# Start with decade view instead of default month view
dateInput("date", "Date:",
  startview = "decade")
```

dateRangeInput *Create date range input*

Description

Creates a pair of text inputs which, when clicked on, bring up calendars that the user can click on to select dates.

Usage

```
dateRangeInput(inputId, label, start = NULL, end = NULL, min = NULL,
  max = NULL, format = "yyyy-mm-dd", startview = "month", weekstart = 0,
  language = "en", separator = " to ")
```

Arguments

inputId	Input variable to assign the control's value to.
label	Display label for the control, or NULL.
start	The initial start date. Either a Date object, or a string in yyyy-mm-dd format. If NULL (the default), will use the current date in the client's time zone.
end	The initial end date. Either a Date object, or a string in yyyy-mm-dd format. If NULL (the default), will use the current date in the client's time zone.
min	The minimum allowed date. Either a Date object, or a string in yyyy-mm-dd format.
max	The maximum allowed date. Either a Date object, or a string in yyyy-mm-dd format.
format	The format of the date to display in the browser. Defaults to "yyyy-mm-dd".
startview	The date range shown when the input object is first clicked. Can be "month" (the default), "year", or "decade".
weekstart	Which day is the start of the week. Should be an integer from 0 (Sunday) to 6 (Saturday).
language	The language used for month and day names. Default is "en". Other valid values include "bg", "ca", "cs", "da", "de", "el", "es", "fi", "fr", "he", "hr", "hu", "id", "is", "it", "ja", "kr", "lt", "lv", "ms", "nb", "nl", "pl", "pt", "pt-BR", "ro", "rs", "rs-latin", "ru", "sk", "sl", "sv", "sw", "th", "tr", "uk", "zh-CN", and "zh-TW".
separator	String to display between the start and end input boxes.

Details

The date format string specifies how the date will be displayed in the browser. It allows the following values:

- yy Year without century (12)
- yyyy Year with century (2012)
- mm Month number, with leading zero (01-12)
- m Month number, without leading zero (01-12)
- M Abbreviated month name
- MM Full month name
- dd Day of month with leading zero
- d Day of month without leading zero
- D Abbreviated weekday name
- DD Full weekday name

See Also

[dateInput](#), [updateDateRangeInput](#)

Other input.elements: [animationOptions](#), [sliderInput](#); [checkboxGroupInput](#); [checkboxInput](#); [dateInput](#); [numericInput](#); [radioButtons](#); [selectInput](#), [selectizeInput](#); [submitButton](#); [textInput](#)

Examples

```
dateRangeInput("daterange", "Date range:",
               start = "2001-01-01",
               end   = "2010-12-31")

# Default start and end is the current date in the client's time zone
dateRangeInput("daterange", "Date range:")

# start and end are always specified in yyyy-mm-dd, even if the display
# format is different
dateRangeInput("daterange", "Date range:",
               start = "2001-01-01",
               end   = "2010-12-31",
               min   = "2001-01-01",
               max   = "2012-12-21",
               format = "mm/dd/yy",
               separator = " - ")

# Pass in Date objects
dateRangeInput("daterange", "Date range:",
               start = Sys.Date()-10,
               end   = Sys.Date()+10)

# Use different language and different first day of week
dateRangeInput("daterange", "Date range:",
```

```

      language = "de",
      weekstart = 1)

# Start with decade view instead of default month view
dateRangeInput("daterange", "Date range:",
               startview = "decade")

```

fixedPage

Create a page with a fixed layout

Description

Functions for creating fixed page layouts. A fixed page layout consists of rows which in turn include columns. Rows exist for the purpose of making sure their elements appear on the same line (if the browser has adequate width). Columns exist for the purpose of defining how much horizontal space within a 12-unit wide grid it's elements should occupy. Fixed pages limit their width to 940 pixels on a typical display, and 724px or 1170px on smaller and larger displays respectively.

Usage

```

fixedPage(..., title = NULL, responsive = TRUE, theme = NULL)

fixedRow(...)

```

Arguments

...	Elements to include within the container
title	The browser window title (defaults to the host URL of the page)
responsive	TRUE to use responsive layout (automatically adapt and resize page elements based on the size of the viewing device)
theme	Alternative Bootstrap stylesheet (normally a css file within the www directory). For example, to use the theme located at <code>www/bootstrap.css</code> you would use <code>theme = "bootstrap.css"</code> .

Details

To create a fixed page use the `fixedPage` function and include instances of `fixedRow` and `column` within it. Note that unlike `fluidPage`, fixed pages cannot make use of higher-level layout functions like `sidebarLayout`, rather, all layout must be done with `fixedRow` and `column`.

Value

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

Note

See the [Shiny Application Layout Guide](#) for additional details on laying out fixed pages.

See Also[column](#)**Examples**

```
library(shiny)

fixedPage(
  title = "Hello, Shiny!",
  fixedRow(
    column(width = 4,
           "4"
    ),
    column(width = 3, offset = 2,
           "3 offset 2"
    )
  )
)
```

`fluidPage`*Create a page with fluid layout*

Description

Functions for creating fluid page layouts. A fluid page layout consists of rows which in turn include columns. Rows exist for the purpose of making sure their elements appear on the same line (if the browser has adequate width). Columns exist for the purpose of defining how much horizontal space within a 12-unit wide grid it's elements should occupy. Fluid pages scale their components in realtime to fill all available browser width.

Usage

```
fluidPage(..., title = NULL, responsive = TRUE, theme = NULL)
```

```
fluidRow(...)
```

Arguments

<code>...</code>	Elements to include within the page
<code>title</code>	The browser window title (defaults to the host URL of the page). Can also be set as a side effect of the titlePanel function.
<code>responsive</code>	TRUE to use responsive layout (automatically adapt and resize page elements based on the size of the viewing device)
<code>theme</code>	Alternative Bootstrap stylesheet (normally a css file within the www directory). For example, to use the theme located at <code>www/bootstrap.css</code> you would use <code>theme = "bootstrap.css"</code> .

Details

To create a fluid page use the `fluidPage` function and include instances of `fluidRow` and `column` within it. As an alternative to low-level row and column functions you can also use higher-level layout functions like `sidebarLayout`.

Value

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

Note

See the [Shiny-Application-Layout-Guide](#) for additional details on laying out fluid pages.

See Also

[column](#), [sidebarLayout](#)

Examples

```
library(shiny)

fluidPage(

  # Application title
  titlePanel("Hello Shiny!"),

  sidebarLayout(

    # Sidebar with a slider input
    sidebarPanel(
      sliderInput("obs",
                  "Number of observations:",
                  min = 0,
                  max = 1000,
                  value = 500)
    ),

    # Show a plot of the generated distribution
    mainPanel(
      plotOutput("distPlot")
    )
  )
)

fluidPage(
  title = "Hello Shiny!",
  fluidRow(
    column(width = 4,
           "4"
    ),
    column(width = 3, offset = 2,
           "3 offset 2"
    )
  )
)
```

```

    )
  )
)
```

headerPanel *Create a header panel*

Description

Create a header panel containing an application title.

Usage

```
headerPanel(title, windowTitle = title)
```

Arguments

title	An application title to display
windowTitle	The title that should be displayed by the browser window. Useful if title is not a string.

Value

A headerPanel that can be passed to [pageWithSidebar](#)

Examples

```
headerPanel("Hello Shiny!")
```

icon *Create an icon*

Description

Create an icon for use within a page. Icons can appear on their own, inside of a button, or as an icon for a [tabPanel](#) within a [navbarPage](#).

Usage

```
icon(name, class = NULL, lib = "font-awesome")
```

Arguments

name	Name of icon. Icons are drawn from the Font Awesome library. Note that the "fa-" prefix should not be used in icon names (i.e. the "fa-calendar" icon should be referred to as "calendar")
class	Additional classes to customize the style of the icon (see the usage examples for details on supported styles).
lib	Icon library to use ("font-awesome" is only one currently supported)

Value

An icon element

Examples

```
library(shiny)
icon("calendar")      # standard icon
icon("calendar", "fa-3x") # 3x normal size

# add an icon to a submit button
submitButton("Update View", icon = icon("refresh"))

navbarPage("App Title",
  tabPanel("Plot", icon = icon("bar-chart-o")),
  tabPanel("Summary", icon = icon("list-alt")),
  tabPanel("Table", icon = icon("table"))
)
```

mainPanel

Create a main panel

Description

Create a main panel containing output elements that can in turn be passed to [sidebarLayout](#).

Usage

```
mainPanel(..., width = 8)
```

Arguments

...	Output elements to include in the main panel
width	The width of the main panel. For fluid layouts this is out of 12 total units; for fixed layouts it is out of whatever the width of the main panel's parent column is.

Value

A main panel that can be passed to [sidebarLayout](#).

Examples

```
library(shiny)
# Show the caption and plot of the requested variable against mpg
mainPanel(
  h3(textOutput("caption")),
  plotOutput("mpgPlot")
)
```

 navbarPage

Create a page with a top level navigation bar

Description

Create a page that contains a top level navigation bar that can be used to toggle a set of [tabPanel](#) elements.

Usage

```
navbarPage(title, ..., id = NULL, header = NULL, footer = NULL,
  inverse = FALSE, collapsable = FALSE, fluid = TRUE, responsive = TRUE,
  theme = NULL, windowTitle = title)
```

Arguments

title	The title to display in the navbar
...	tabPanel elements to include in the page
id	If provided, you can use <code>input\$id</code> in your server logic to determine which of the current tabs is active. The value will correspond to the value argument that is passed to tabPanel .
header	Tag or list of tags to display as a common header above all <code>tabPanels</code> .
footer	Tag or list of tags to display as a common footer below all <code>tabPanels</code>
inverse	TRUE to use a dark background and light text for the navigation bar
collapsable	TRUE to automatically collapse the navigation elements into a menu when the width of the browser is less than 940 pixels (useful for viewing on smaller touch-screen device)
fluid	TRUE to use a fluid layout. FALSE to use a fixed layout.
responsive	TRUE to use responsive layout (automatically adapt and resize page elements based on the size of the viewing device)
theme	Alternative Bootstrap stylesheet (normally a css file within the www directory). For example, to use the theme located at <code>www/bootstrap.css</code> you would use <code>theme = "bootstrap.css"</code> .
windowTitle	The title that should be displayed by the browser window. Useful if <code>title</code> is not a string.

Details

The navbarMenu function can be used to create an embedded menu within the navbar that in turns includes additional tabPanels (see example below).

Value

A UI defintion that can be passed to the [shinyUI](#) function.

See Also

[tabPanel](#), [tabsetPanel](#)

Examples

```
## Not run:
navbarPage("App Title",
  tabPanel("Plot"),
  tabPanel("Summary"),
  tabPanel("Table")
)

navbarPage("App Title",
  tabPanel("Plot"),
  navbarMenu("More",
    tabPanel("Summary"),
    tabPanel("Table")
  )
)

## End(Not run)
```

navlistPanel

Create a navigation list panel

Description

Create a navigation list panel that provides a list of links on the left which navigate to a set of tabPanels displayed to the right.

Usage

```
navlistPanel(..., id = NULL, selected = NULL, well = TRUE, fluid = TRUE,
  widths = c(4, 8))
```

Arguments

...	<code>tabPanel</code> elements to include in the navlist
<code>id</code>	If provided, you can use <code>input\$id</code> in your server logic to determine which of the current navlist items is active. The value will correspond to the <code>value</code> argument that is passed to <code>tabPanel</code> .
<code>selected</code>	The <code>value</code> (or, if none was supplied, the <code>title</code>) of the navigation item that should be selected by default. If <code>NULL</code> , the first navigation will be selected.
<code>well</code>	<code>TRUE</code> to place a well (gray rounded rectangle) around the navigation list.
<code>fluid</code>	<code>TRUE</code> to use fluid layout; <code>FALSE</code> to use fixed layout.
<code>widths</code>	Column widths of the navigation list and tabset content areas respectively.

Details

You can include headers within the `navlistPanel` by including plain text elements in the list; you can include separators by including "—" (any number of dashes works).

Examples

```
library(shiny)

fluidPage(
  titlePanel("Application Title"),

  navlistPanel(
    "Header",
    tabPanel("First"),
    tabPanel("Second"),
    "-----",
    tabPanel("Third")
  )
)
```

 numericInput

Create a numeric input control

Description

Create an input control for entry of numeric values

Usage

```
numericInput(inputId, label, value, min = NA, max = NA, step = NA)
```

Arguments

inputId	Input variable to assign the control's value to
label	Display label for the control
value	Initial value
min	Minimum allowed value
max	Maximum allowed value
step	Interval to use when stepping between min and max

Value

A numeric input control that can be added to a UI definition.

See Also

[updateNumericInput](#)

Other input.elements: [animationOptions](#), [sliderInput](#); [checkboxGroupInput](#); [checkboxInput](#); [dateInput](#); [dateRangeInput](#); [radioButtons](#); [selectInput](#), [selectizeInput](#); [submitButton](#); [textInput](#)

Examples

```
numericInput("obs", "Observations:", 10,  
            min = 1, max = 100)
```

pageWithSidebar	<i>Create a page with a sidebar</i>
-----------------	-------------------------------------

Description

Create a Shiny UI that contains a header with the application title, a sidebar for input controls, and a main area for output.

Usage

```
pageWithSidebar(headerPanel, sidebarPanel, mainPanel)
```

Arguments

headerPanel	The headerPanel with the application title
sidebarPanel	The sidebarPanel containing input controls
mainPanel	The mainPanel containing outputs

Value

A UI definition that can be passed to the [shinyUI](#) function

Note

This function is deprecated. You should use `fluidPage` along with `sidebarLayout` to implement a page with a sidebar.

Examples

```
library(shiny)

pageWithSidebar(

  # Application title
  headerPanel("Hello Shiny!"),

  # Sidebar with a slider input
  sidebarPanel(
    sliderInput("obs",
               "Number of observations:",
               min = 0,
               max = 1000,
               value = 500)
  ),

  # Show a plot of the generated distribution
  mainPanel(
    plotOutput("distPlot")
  )
)
```

radioButtons

Create radio buttons

Description

Create a set of radio buttons used to select an item from a list.

Usage

```
radioButtons(inputId, label, choices, selected = NULL, inline = FALSE)
```

Arguments

<code>inputId</code>	Input variable to assign the control's value to
<code>label</code>	Display label for the control, or NULL
<code>choices</code>	List of values to select from (if elements of the list are named then that name rather than the value is displayed to the user)
<code>selected</code>	The initially selected value (if not specified then defaults to the first value)
<code>inline</code>	If TRUE, render the choices inline (i.e. horizontally)

Value

A set of radio buttons that can be added to a UI definition.

See Also

[updateRadioButtons](#)

Other `input.elements`: [animationOptions](#), [sliderInput](#); [checkboxGroupInput](#); [checkboxInput](#); [dateInput](#); [dateRangeInput](#); [numericInput](#); [selectInput](#), [selectizeInput](#); [submitButton](#); [textInput](#)

Examples

```
radioButtons("dist", "Distribution type:",
  c("Normal" = "norm",
    "Uniform" = "unif",
    "Log-normal" = "lnorm",
    "Exponential" = "exp"))
```

selectInput

Create a select list input control

Description

Create a select list that can be used to choose a single or multiple items from a list of values.

Usage

```
selectInput(inputId, label, choices, selected = NULL, multiple = FALSE,
  selectize = TRUE, width = NULL)
```

```
selectizeInput(inputId, ..., options = NULL, width = NULL)
```

Arguments

<code>inputId</code>	Input variable to assign the control's value to
<code>label</code>	Display label for the control, or <code>NULL</code>
<code>choices</code>	List of values to select from. If elements of the list are named then that name rather than the value is displayed to the user.
<code>selected</code>	The initially selected value (or multiple values if <code>multiple = TRUE</code>). If not specified then defaults to the first value for single-select lists and no values for multiple select lists.
<code>multiple</code>	Is selection of multiple items allowed?
<code>selectize</code>	Whether to use <code>selectize.js</code> or not.
<code>width</code>	The width of the input, e.g. <code>'400px'</code> , or <code>'100%'</code> ; see validateCssUnit .
<code>...</code>	Arguments passed to <code>selectInput()</code> .

options A list of options. See the documentation of **selectize.js** for possible options (character option values inside `I()` will be treated as literal JavaScript code; see `renderDataTable()` for details).

Details

By default, `selectInput()` and `selectizeInput()` use the JavaScript library **selectize.js** (<https://github.com/brianreavis/selectize.js>) instead of the basic select input element. To use the standard HTML select input element, use `selectInput()` with `selectize=FALSE`.

Value

A select list control that can be added to a UI definition.

Note

The selectize input created from `selectizeInput()` allows deletion of the selected option even in a single select input, which will return an empty string as its value. This is the default behavior of **selectize.js**. However, the selectize input created from `selectInput(..., selectize = TRUE)` will ignore the empty string value when it is a single choice input and the empty string is not in the choices argument. This is to keep compatibility with `selectInput(..., selectize = FALSE)`.

See Also

[updateSelectInput](#)

Other input.elements: [animationOptions](#), [sliderInput](#); [checkboxGroupInput](#); [checkboxInput](#); [dateInput](#); [dateRangeInput](#); [numericInput](#); [radioButtons](#); [submitButton](#); [textInput](#)

Examples

```
selectInput("variable", "Variable:",
  c("Cylinders" = "cyl",
    "Transmission" = "am",
    "Gears" = "gear"))
```

sidebarLayout

Layout a sidebar and main area

Description

Create a layout with a sidebar and main area. The sidebar is displayed with a distinct background color and typically contains input controls. The main area occupies 2/3 of the horizontal width and typically contains outputs.

Usage

```
sidebarLayout(sidebarPanel, mainPanel, position = c("left", "right"),
  fluid = TRUE)
```

Arguments

sidebarPanel	The sidebarPanel containing input controls
mainPanel	The mainPanel containing outputs
position	The position of the sidebar relative to the main area ("left" or "right")
fluid	TRUE to use fluid layout; FALSE to use fixed layout.

Examples

```
library(shiny)

# Define UI
fluidPage(

  # Application title
  titlePanel("Hello Shiny!"),

  sidebarLayout(

    # Sidebar with a slider input
    sidebarPanel(
      sliderInput("obs",
                  "Number of observations:",
                  min = 0,
                  max = 1000,
                  value = 500)
    ),

    # Show a plot of the generated distribution
    mainPanel(
      plotOutput("distPlot")
    )
  )
)
```

sidebarPanel

Create a sidebar panel

Description

Create a sidebar panel containing input controls that can in turn be passed to [sidebarLayout](#).

Usage

```
sidebarPanel(..., width = 4)
```

Arguments

...	UI elements to include on the sidebar
width	The width of the sidebar. For fluid layouts this is out of 12 total units; for fixed layouts it is out of whatever the width of the sidebar's parent column is.

Value

A sidebar that can be passed to [sidebarLayout](#)

Examples

```
# Sidebar with controls to select a dataset and specify
# the number of observations to view
sidebarPanel(
  selectInput("dataset", "Choose a dataset:",
             choices = c("rock", "pressure", "cars")),

  numericInput("obs", "Observations:", 10)
)
```

 sliderInput

Slider Input Widget

Description

Constructs a slider widget to select a numeric value from a range.

Usage

```
sliderInput(inputId, label, min, max, value, step = NULL, round = FALSE,
            format = "#,##0.#####", locale = "us", ticks = TRUE, animate = FALSE,
            width = NULL)
```

```
animationOptions(interval = 1000, loop = FALSE, playButton = NULL,
                 pauseButton = NULL)
```

Arguments

inputId	Specifies the input slot that will be used to access the value.
label	A descriptive label to be displayed with the widget, or NULL.
min	The minimum value (inclusive) that can be selected.
max	The maximum value (inclusive) that can be selected.
value	The initial value of the slider. A numeric vector of length one will create a regular slider; a numeric vector of length two will create a double-ended range slider. A warning will be issued if the value doesn't fit between min and max.

step	Specifies the interval between each selectable value on the slider (NULL means no restriction).
round	TRUE to round all values to the nearest integer; FALSE if no rounding is desired; or an integer to round to that number of digits (for example, 1 will round to the nearest 10, and -2 will round to the nearest .01). Any rounding will be applied after snapping to the nearest step.
format	Customize format values in slider labels. See https://code.google.com/p/jquery-numberformatter/ for syntax details.
locale	The locale to be used when applying format. See details.
ticks	FALSE to hide tick marks, TRUE to show them according to some simple heuristics.
animate	TRUE to show simple animation controls with default settings; FALSE not to; or a custom settings list, such as those created using <code>animationOptions</code> .
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit .
interval	The interval, in milliseconds, between each animation step.
loop	TRUE to automatically restart the animation when it reaches the end.
playButton	Specifies the appearance of the play button. Valid values are a one-element character vector (for a simple text label), an HTML tag or list of tags (using tag and friends), or raw HTML (using HTML).
pauseButton	Similar to <code>playButton</code> , but for the pause button.

Details

Valid values for `locale` are:

Arab Emirates	"ae"
Australia	"au"
Austria	"at"
Brazil	"br"
Canada	"ca"
China	"cn"
Czech	"cz"
Denmark	"dk"
Egypt	"eg"
Finland	"fi"
France	"fr"
Germany	"de"
Greece	"gr"
Great Britain	"gb"
Hong Kong	"hk"
India	"in"
Israel	"il"
Japan	"jp"
Russia	"ru"
South Korea	"kr"
Spain	"es"

Sweden	"se"
Switzerland	"ch"
Taiwan	"tw"
Thailand	"th"
United States	"us"
Vietnam	"vn"

See Also

[updateSliderInput](#)

Other `input.elements`: [checkboxGroupInput](#); [checkboxInput](#); [dateInput](#); [dateRangeInput](#); [numericInput](#); [radioButtons](#); [selectInput](#), [selectizeInput](#); [submitButton](#); [textInput](#)

submitButton

Create a submit button

Description

Create a submit button for an input form. Forms that include a submit button do not automatically update their outputs when inputs change, rather they wait until the user explicitly clicks the submit button.

Usage

```
submitButton(text = "Apply Changes", icon = NULL)
```

Arguments

<code>text</code>	Button caption
<code>icon</code>	Optional <code>icon</code> to appear on the button

Value

A submit button that can be added to a UI definition.

See Also

Other `input.elements`: [animationOptions](#), [sliderInput](#); [checkboxGroupInput](#); [checkboxInput](#); [dateInput](#); [dateRangeInput](#); [numericInput](#); [radioButtons](#); [selectInput](#), [selectizeInput](#); [textInput](#)

Examples

```
library(shiny)
submitButton("Update View")
submitButton("Update View", icon("refresh"))
```

textInput	<i>Create a text input control</i>
-----------	------------------------------------

Description

Create an input control for entry of unstructured text values

Usage

```
textInput(inputId, label, value = "")
```

Arguments

inputId	Input variable to assign the control's value to
label	Display label for the control
value	Initial value

Value

A text input control that can be added to a UI definition.

See Also

[updateTextInput](#)

Other input.elements: [animationOptions](#), [sliderInput](#); [checkboxGroupInput](#); [checkboxInput](#); [dateInput](#); [dateRangeInput](#); [numericInput](#); [radioButtons](#); [selectInput](#), [selectizeInput](#); [submitButton](#)

Examples

```
textInput("caption", "Caption:", "Data Summary")
```

titlePanel	<i>Create a panel containing an application title.</i>
------------	--

Description

Create a panel containing an application title.

Usage

```
titlePanel(title, windowTitle = title)
```

Arguments

title	An application title to display
windowTitle	The title that should be displayed by the browser window.

Details

Calling this function has the side effect of including a `title` tag within the head. You can also specify a page title explicitly using the `'title'` parameter of the top-level page function.

Examples

```
titlePanel("Hello Shiny!")
```

```
updateCheckboxGroupInput
```

Change the value of a checkbox group input on the client

Description

Change the value of a checkbox group input on the client

Usage

```
updateCheckboxGroupInput(session, inputId, label = NULL, choices = NULL,
  selected = NULL, inline = FALSE)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The id of the input object.
label	The label to set for the input object.
choices	List of values to show checkboxes for. If elements of the list are named then that name rather than the value is displayed to the user.
selected	The values that should be initially selected, if any.
inline	If TRUE, render the choices inline (i.e. horizontally)

Details

The input updater functions send a message to the client, telling it to change the settings of an input object. The messages are collected and sent after all the observers (including outputs) have finished running.

The syntax of these functions is similar to the functions that created the inputs in the first place. For example, `numericInput()` and `updateNumericInput()` take a similar set of arguments.

Any arguments with NULL values will be ignored; they will not result in any changes to the input object on the client.

See Also[checkboxGroupInput](#)**Examples**

```
## Not run:
shinyServer(function(input, output, session) {

  observe({
    # We'll use the input$controller variable multiple times, so save it as x
    # for convenience.
    x <- input$controller

    # Create a list of new options, where the name of the items is something
    # like 'option label x 1', and the values are 'option-x-1'.
    cb_options <- list()
    cb_options[[sprintf("option label %d 1", x)]] <- sprintf("option-%d-1", x)
    cb_options[[sprintf("option label %d 2", x)]] <- sprintf("option-%d-2", x)

    # Change values for input$inCheckboxGroup
    updateCheckboxGroupInput(session, "inCheckboxGroup", choices = cb_options)

    # Can also set the label and select items
    updateCheckboxGroupInput(session, "inCheckboxGroup2",
      label = paste("checkboxgroup label", x),
      choices = cb_options,
      selected = sprintf("option-%d-2", x)
    )
  })
})

## End(Not run)
```

updateRadioButtons *Change the value of a radio input on the client*

Description

Change the value of a radio input on the client

Usage

```
updateRadioButtons(session, inputId, label = NULL, choices = NULL,
  selected = NULL, inline = FALSE)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The id of the input object.
label	The label to set for the input object.
choices	List of values to select from (if elements of the list are named then that name rather than the value is displayed to the user)
selected	The initially selected value (if not specified then defaults to the first value)
inline	If TRUE, render the choices inline (i.e. horizontally)

Details

The input updater functions send a message to the client, telling it to change the settings of an input object. The messages are collected and sent after all the observers (including outputs) have finished running.

The syntax of these functions is similar to the functions that created the inputs in the first place. For example, `numericInput()` and `updateNumericInput()` take a similar set of arguments.

Any arguments with NULL values will be ignored; they will not result in any changes to the input object on the client.

See Also

[radioButtons](#)

Examples

```
## Not run:
shinyServer(function(input, output, session) {

  observe({
    # We'll use the input$controller variable multiple times, so save it as x
    # for convenience.
    x <- input$controller

    r_options <- list()
    r_options[[sprintf("option label %d 1", x)]] <- sprintf("option-%d-1", x)
    r_options[[sprintf("option label %d 2", x)]] <- sprintf("option-%d-2", x)

    # Change values for input$inRadio
    updateRadioButtons(session, "inRadio", choices = r_options)

    # Can also set the label and select an item
    updateRadioButtons(session, "inRadio2",
      label = paste("Radio label", x),
      choices = r_options,
      selected = sprintf("option-%d-2", x)
    )
  })
})
```

```
## End(Not run)
```

updateSliderInput *Change the value of a slider input on the client*

Description

Change the value of a slider input on the client

Usage

```
updateSliderInput(session, inputId, label = NULL, value = NULL)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The id of the input object.
label	The label to set for the input object.
value	The value to set for the input object.

Details

The input updater functions send a message to the client, telling it to change the settings of an input object. The messages are collected and sent after all the observers (including outputs) have finished running.

The syntax of these functions is similar to the functions that created the inputs in the first place. For example, `numericInput()` and `updateNumericInput()` take a similar set of arguments.

Any arguments with NULL values will be ignored; they will not result in any changes to the input object on the client.

See Also

[sliderInput](#)

Examples

```
## Not run:
shinyServer(function(input, output, session) {

  observe({
    # We'll use the input$controller variable multiple times, so save it as x
    # for convenience.
    x <- input$controller

    # Similar to number and text. only label and value can be set for slider
    updateSliderInput(session, "inSlider",
```

```
    label = paste("Slider label", x),
    value = x)

# For sliders that pick out a range, pass in a vector of 2 values.
updateSliderInput(session, "inSlider2", value = c(x-1, x+1))

# An NA means to not change that value (the low or high one)
updateSliderInput(session, "inSlider3", value = c(NA, x+2))
  })
})

## End(Not run)
```

verticalLayout	<i>Lay out UI elements vertically</i>
----------------	---------------------------------------

Description

Create a container that includes one or more rows of content (each element passed to the container will appear on it's own line in the UI)

Usage

```
verticalLayout(..., fluid = TRUE)
```

Arguments

...	Elements to include within the container
fluid	TRUE to use fluid layout; FALSE to use fixed layout.

See Also

[fluidPage](#), [flowLayout](#)

Examples

```
library(shiny)

fluidPage(
  verticalLayout(
    a(href="http://example.com/link1", "Link One"),
    a(href="http://example.com/link2", "Link Two"),
    a(href="http://example.com/link3", "Link Three")
  )
)
```

withBootstrap2 *Run Shiny UI code with Bootstrap 2 elements.*

Description

This function takes an expression containing calls to functions in the shinybootstrap2 package, and evaluates it in an environment where these functions will be found, even when shinybootstrap2 is not attached.

Usage

```
withBootstrap2(x, env = parent.frame(), quoted = FALSE)
```

Arguments

x	An expression to evaluate with Bootstrap 2 components.
env	The environment in which to evaluate x.
quoted	Treat x as a quoted expression. If FALSE (the default) x will be treated as an unquoted expression. If TRUE, the code should be the output of a <code>quote()</code> .

Details

Shiny version 0.11 and above uses Bootstrap 3 instead of Bootstrap 2. The purpose of the shinybootstrap2 package is to provide backward compatibility when needed. Almost all of the functions in shinybootstrap2 have the same name as functions in shiny, but they generate HTML that works with Bootstrap 2 instead of 3.

This function should almost always be called using `shinybootstrap2::withBootstrap2()`, without attaching the package. In other words, `library(shinybootstrap2)`, shouldn't appear in your code. This is because attaching the package will result in functions from shinybootstrap2 masking functions from shiny, even outside of `withBootstrap2()`.

Examples

```
## Not run:
library(shiny)

## Single-file app using Bootstrap 2 =====
shinybootstrap2::withBootstrap2({
  shinyApp(
    ui = fluidPage(
      numericInput("n", "n", 1),
      plotOutput("plot")
    ),
    server = function(input, output) {
      output$plot <- renderPlot( plot(head(cars, input$n)) )
    }
  )
})
```

```
## App with server.R and UI. R =====
## ui.R
shinybootstrap2::withBootstrap2({
  fluidPage(
    selectInput("ui", "Input type", choices = c("numeric", "slider")),
    uiOutput("n_ui"),
    plotOutput("plot")
  )
})

## server.R
# In server.R, it's only necessary to wrap code in withBootstrap2()
# when renderUI() is used.
shinybootstrap2::withBootstrap2({
  function(input, output) {
    output$n_ui <- renderUI({
      if (input$ui == "numeric")
        numericInput("n", "n", 1)
      else if (input$ui == "slider")
        sliderInput("n", "n", 1, 10, value = 1)
    })
    output$plot <- renderPlot( plot(head(cars, input$n)) )
  }
})

## End(Not run)
```

Index

- animationOptions, [4](#), [5](#), [8](#), [10](#), [19](#), [21](#), [22](#), [26](#), [27](#)
- animationOptions (sliderInput), [24](#)
- basicPage (bootstrapPage), [3](#)
- bootstrapPage, [3](#)
- checkboxGroupInput, [4](#), [5](#), [8](#), [10](#), [19](#), [21](#), [22](#), [26](#), [27](#), [29](#)
- checkboxInput, [4](#), [5](#), [8](#), [10](#), [19](#), [21](#), [22](#), [26](#), [27](#)
- column, [5](#), [11–13](#)
- dataTableOutput, [6](#)
- dateInput, [4](#), [5](#), [7](#), [10](#), [19](#), [21](#), [22](#), [26](#), [27](#)
- dateRangeInput, [4](#), [5](#), [8](#), [9](#), [19](#), [21](#), [22](#), [26](#), [27](#)
- fixedPage, [3](#), [11](#)
- fixedRow, [5](#), [6](#)
- fixedRow (fixedPage), [11](#)
- flowLayout, [32](#)
- fluidPage, [3](#), [11](#), [12](#), [20](#), [32](#)
- fluidRow, [3](#), [5](#), [6](#)
- fluidRow (fluidPage), [12](#)
- headerPanel, [14](#), [19](#)
- HTML, [25](#)
- I, [22](#)
- icon, [14](#), [26](#)
- mainPanel, [15](#), [19](#), [23](#)
- navbarPage, [14](#), [16](#)
- navlistPanel, [17](#)
- numericInput, [4](#), [5](#), [8](#), [10](#), [18](#), [21](#), [22](#), [26–28](#), [30](#), [31](#)
- pageWithSidebar, [14](#), [19](#)
- quote, [33](#)
- radioButtons, [4](#), [5](#), [8](#), [10](#), [19](#), [20](#), [22](#), [26](#), [27](#), [30](#)
- renderDataTable, [6](#), [22](#)
- selectInput, [4](#), [5](#), [8](#), [10](#), [19](#), [21](#), [21](#), [26](#), [27](#)
- selectizeInput, [4](#), [5](#), [8](#), [10](#), [19](#), [21](#), [26](#), [27](#)
- selectizeInput (selectInput), [21](#)
- shinybootstrap2
 - (shinybootstrap2-package), [2](#)
- shinybootstrap2-package, [2](#)
- shinyUI, [3](#), [11](#), [13](#), [17](#), [19](#)
- sidebarLayout, [3](#), [13](#), [15](#), [20](#), [22](#), [23](#), [24](#)
- sidebarPanel, [19](#), [23](#), [23](#)
- sliderInput, [4](#), [5](#), [8](#), [10](#), [19](#), [21](#), [22](#), [24](#), [26](#), [27](#), [31](#)
- submitButton, [4](#), [5](#), [8](#), [10](#), [19](#), [21](#), [22](#), [26](#), [26](#), [27](#)
- tabpanel, [14](#), [16–18](#)
- tabsetPanel, [17](#)
- tag, [25](#)
- textInput, [4](#), [5](#), [8](#), [10](#), [19](#), [21](#), [22](#), [26](#), [27](#)
- titlePanel, [12](#), [27](#)
- updateCheckboxGroupInput, [4](#), [28](#)
- updateCheckboxInput, [5](#)
- updateDateInput, [8](#)
- updateDateRangeInput, [10](#)
- updateNumericInput, [19](#)
- updateRadioButtons, [21](#), [29](#)
- updateSelectInput, [22](#)
- updateSliderInput, [26](#), [31](#)
- updateTextInput, [27](#)
- validateCssUnit, [21](#), [25](#)
- verticalLayout, [32](#)
- withBootstrap2, [33](#)