Package 'shiny.semantic'

March 13, 2020

Type Package

Title Semantic UI Support for Shiny

Version 0.3.0

Description Creating a great user interface for your Shiny apps can be a hassle, especially if you want to work purely in R and don't want to use, for instance HTML templates. This package adds support for a powerful UI library Semantic UI - <http://semantic-ui.com/>. It also supports universal UI input binding that works with various DOM elements.

BugReports https://github.com/Appsilon/shiny.semantic/issues

Encoding UTF-8

LazyData TRUE

License MIT + file LICENSE

Imports shiny (>= 0.12.1), htmltools (>= 0.2.6), htmlwidgets (>= 0.8), purrr (>= 0.2.2), magrittr, jsonlite

Suggests dplyr, gapminder, testthat, lintr, covr

RoxygenNote 7.0.2.9000

NeedsCompilation no

Author Filip Stachura [aut], Krystian Igras [aut], Adam Forys [aut], Dominik Krzeminski [cre], Appsilon Sp. z o.o. [cph]

Maintainer Dominik Krzeminski <dominik@appsilon.com>

Repository CRAN

Date/Publication 2020-03-13 07:10:02 UTC

R topics documented:

.onLoad 3

attach_rule . . . 3
checkbox_positions . . . 4
check_proper_color . . . 4
check_semantic_theme . . . 5
create_modal . . . 5
date_input . . . 6
define_selection_type . . . 7
dropdown . . . 7
get_cdn_path . . . 9
get_default_semantic_theme . . . 9
get_dependencies . . . 10
label . . . 10
list_element . . . 11
menu_divider . . . 11
menu_header . . . 12
menu_item . . . 12
modal . . . 13
multiple_checkbox . . . 15
parse_val . . . 17
register_search . . . 18
search_field . . . 18
search_selection_api . . . 19
search_selection_choices . . . 20
semanticPage . . . 22
semantic_palette . . . 22
set_tab_id . . . 23
shiny.semantic . . . 23
shiny_input . . . 24
shiny_text_input . . . 24
show_modal . . . 25
simple_checkbox . . . 25
SUPPORTED_THEMES . . . 26
tabset . . . 26
uibutton . . . 27
uicalendar . . . 28
uicard . . . 29
uicards . . . 30
uicheckbox . . . 30
uidropdown . . . 31
uifield . . . 31
uifields . . . 32
uiform . . . 32
uiheader . . . 33
uiicon . . . 33
uiinput . . . 34
uilabel . . . 34
uilist . . . 35
uimenu . . . 36

uimessage 37
uinumericinput 37
uirender 38
uisegment 39
uislider 39
uitextinput 40
update_dropdown 42
update_slider 42
%:::% 43

Index **44**

`.onLoad` *Internal function that expose javascript bindings to Shiny app.*

Description

Internal function that expose javascript bindings to Shiny app.

Usage

```
.onLoad(libname, pkgname)
```

Arguments

`libname`	library name
`pkgname`	package name

`attach_rule` *Internal function that creates the rule for a specific setting or behavior of the modal.*

Description

Internal function that creates the rule for a specific setting or behavior of the modal.

Usage

```
attach_rule(id, behavior, target, value)
```

Arguments

`id`	ID of the target modal.
`behavior`	What behavior is beging set i. e. setting or attach events.
`target`	First argument of the behavior. Usually a target or a setting name.
`value`	Second argument of the behavior. usually an action or a setting value.

checkbox_positions *Checkbox positions*

Description

Checkbox positions

Usage

```
checkbox_positions
```

Format

An object of class `character` of length 2.

check_proper_color *Check if color is set from Semanti-UI palette*

Description

Check if color is set from Semanti-UI palette

Usage

```
check_proper_color(color)
```

Arguments

`color`	character with color name

Value

Error when `color` does not belong to palette

Examples

```
check_proper_color("blue")
```

check_semantic_theme *Semantic theme path validator*

Description

Semantic theme path validator

Usage

```
check_semantic_theme(theme_css, full_url = TRUE)
```

Arguments

theme_css	it can be either NULL, character with css path, or theme name
full_url	boolean flag that defines what is returned, either filename, or full path. Default TRUE

Value

path to theme or filename

Examples

```
check_semantic_theme(NULL)
check_semantic_theme("darkly")
check_semantic_theme("darkly", full_url = FALSE)
```

create_modal *Allows for the creation of modals in the server side without being tied to a specific HTML element.*

Description

Allows for the creation of modals in the server side without being tied to a specific HTML element.

Usage

```
create_modal(
  ui_modal,
  show = TRUE,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

ui_modal	HTML containing the modal.
show	If the modal should only be created or open when called (open by default).
session	Current session.

date_input *Define simple date input with semantic ui styling*

Description

Define simple date input with semantic ui styling

Usage

```
date_input(
  name,
  label = NULL,
  value = NULL,
  min = NULL,
  max = NULL,
  style = NULL,
  icon = uiicon("calendar")
)
```

Arguments

name	Input id.
label	Label to be displayed with date input.
value	Default date chosen for input.
min	Minimum date that can be selected.
max	Maximum date that can be selected.
style	Css style for widget.
icon	Icon that should be displayed on widget.

Examples

```
if (interactive()) {
# Below example shows how to imlement simple date range input using \code{date_input}

library(shiny)
library(shiny.semantic)

ui <- shinyUI(
  semanticPage(
    title = "Date range example",
    uiOutput("date_range"),
    p("Selected dates:"),
    textOutput("selected_dates")
  )
)

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

```
  output$date_range <- renderUI({
    tagList(
      tags$div(tags$div(HTML("From")),
               date_input("date_from", value = Sys.Date() - 30, style = "width: 10%;")),
      tags$div(tags$div(HTML("To")),
               date_input("date_to", value = Sys.Date(), style = "width: 10%;"))
    )
  })

  output$selected_dates <- renderPrint({
    c(input$date_from, input$date_to)
  })
})

shinyApp(ui = ui, server = server)
}
```

`define_selection_type` *Define search type if multiple*

Description

Define search type if multiple

Usage

```
define_selection_type(name, multiple)
```

Arguments

`name`	character with name
`multiple`	multiple flag

`dropdown` *Create dropdown Semantic UI component*

Description

This creates a default dropdown using Semantic UI styles with Shiny input. Dropdown is already initialized and available under input[[name]].

Usage

```
dropdown(
  name,
  choices,
  choices_value = choices,
  default_text = "Select",
  value = NULL,
  type = "selection fluid"
)
```

Arguments

name	Input name. Reactive value is available under input[[name]].
choices	All available options one can select from.
choices_value	What reactive value should be used for corresponding choice.
default_text	Text to be visible on dropdown when nothing is selected.
value	Pass value if you want to initialize selection for dropdown.
type	Change depending what type of dropdown is wanted.

Examples

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

  library(shiny)
  library(shiny.semantic)
  ui <- function() {
      shinyUI(
        semanticPage(
          title = "Dropdown example",
          suppressDependencies("bootstrap"),
          uiOutput("dropdown"),
          p("Selected letter:"),
          textOutput("selected_letter")
       )
     )
  }
  server <- shinyServer(function(input, output) {
     output$dropdown <- renderUI({
         dropdown("simple_dropdown", LETTERS, value = "A")
     })
     output$selected_letter <- renderText(input[["simple_dropdown"]])
  })

  shinyApp(ui = ui(), server = server)
}
```

`get_cdn_path` *Get CDN path semantic dependencies*

Description

Internal function that returns path string from ‘shiny.custom.semantic.cdn‘ options.

Usage

```
get_cdn_path()
```

Value

CDN path of semantic dependencies

Examples

```
## Load shiny.semantic dependencies from local domain.
options("shiny.custom.semantic.cdn" = "shiny.semantic")
```

`get_default_semantic_theme` *Get default semantic css*

Description

Get default semantic css

Usage

```
get_default_semantic_theme(full_url = TRUE)
```

Arguments

`full_url`	define return output filename or full path. Default TRUE

Value

path to default css semantic file or default filename

get_dependencies *Add dashboard dependencies to html*

Description

Internal function that adds dashboard dependencies to html.

Usage

```
get_dependencies(theme = NULL)
```

Arguments

theme	define theme

Value

Content with appended dependencies.

label *Create HTML label tag*

Description

This creates a HTML label tag.

Usage

```
label(...)
```

Arguments

...	Other arguments to be added as attributes of the tag (e.g. style, class or childrens etc.)

list_element *Helper function to render list element*

Description

Helper function to render list element

Usage

```
list_element(data, is_description, is_icon, row)
```

Arguments

data	data to list; data.frame with fields header, icon, description
is_description	description flag
is_icon	Icon logical to add icon from data
row	row character

menu_divider *Create Semantic UI Divider Item*

Description

This creates a menu divider item using Semantic UI.

Usage

```
menu_divider(...)
```

Arguments

...	Other attributes of the divider such as style.

menu_header *Create Semantic UI Header Item*

Description

This creates a dropdown header item using Semantic UI.

Usage

```
menu_header(..., is_item = TRUE)
```

Arguments

...	Content of the header: text, icons, etc.
is_item	If TRUE created header is item of Semantic UI Menu.

menu_item *Create Semantic UI Menu Item*

Description

This creates a menu item using Semantic UI

Usage

```
menu_item(..., item_feature = "", style = NULL, href = NULL)
```

Arguments

...	Content of the menu item: text, icons or labels to be displayed.
item_feature	If required, add additional item feature like 'active', 'header', etc.
style	Style of the item, e.g. "text-align: center".
href	If NULL (default) menu_item is created with 'div' tag. Otherwise it is created with 'a' tag, and parameeter defines its href attribute.

`modal` *Create Semantic UI modal*

Description

This creates a modal using Semantic UI styles.

Usage

```
modal(
  ...,
  id = "",
  class = "",
  header = "",
  content = NULL,
  footer = NULL,
  target = NULL,
  settings = NULL,
  modal_tags = NULL
)
```

Arguments

`...`	Content elements to be added to the modal body. To change attributes of the container please check the ‘content‘ argument.
`id`	ID to be added to the modal div. Default "".
`class`	Classes except "ui modal" to be added to the modal. Semantic UI classes can be used. Default "".
`header`	Content to be displayed in the modal header. If given in form of a list, HTML attributes for the container can also be changed. Default "".
`content`	Content to be displayed in the modal body. If given in form of a list, HTML attributes for the container can also be changed. Default NULL.
`footer`	Content to be displayed in the modal footer. Usually for buttons. If given in form of a list, HTML attributes for the container can also be changed. Default NULL.
`target`	Javascript selector for the element that will open the modal. Default NULL.
`settings`	List of vectors of Semantic UI settings to be added to the modal. Default NULL.
`modal_tags`	Other modal elements. Default NULL.

Examples

```
## Create a simple server modal
library(shiny)
library(shiny.semantic)
```

```
ui <- function() {
  shinyUI(
    semanticPage(
      actionButton("show", "Show modal dialog")
    )
  )
}

server = function(input, output) {
  observeEvent(input$show, {
    create_modal(modal(
      id = "simple-modal",
      title = "Important message",
      "This is an important message!"
    ))
  })
}

## Create a simple UI modal
library(shiny)
library(shiny.semantic)
ui <- function() {
  shinyUI(
    semanticPage(
      title = "Modal example - Static UI modal",
      div(id = "modal-open-button", class = "ui button", "Open Modal"),
      modal(
        div("Example content"),
        id = "example-modal",
        target = "modal-open-button"
      )
    )
  )
}

## Observe server side actions
library(shiny)
library(shiny.semantic)
ui <- function() {
  shinyUI(
    semanticPage(
      title = "Modal example - Server side actions",
      uiOutput("modalAction"),
      actionButton("show", "Show by calling show_modal")
    )
  )
}

server <- shinyServer(function(input, output) {
  observeEvent(input$show, {
    show_modal('action-example-modal')
  })
  observeEvent(input$hide, {
```

```
    hide_modal('action-example-modal')
  })

  output$modalAction <- renderUI({
    modal(
      actionButton("hide", "Hide by calling hide_modal"),
      id = "action-example-modal",
      header = "Modal example",
      footer = "",
      class = "tiny"
    )
  })
})

## Changing attributes of header and content.
library(shiny)
library(shiny.semantic)

ui <- function() {
  shinyUI(
    semanticPage(
      actionButton("show", "Show modal dialog")
    )
  )
}

server = function(input, output) {
  observeEvent(input$show, {
    create_modal(modal(
      id = "simple-modal",
      title = "Important message",
      header = list(style = "background: lightcoral"),
      content = list(style = "background: lightblue",
                     `data-custom` = "value", "This is an important message!"),
      p("This is also part of the content!")
    ))
  })
}
```

`multiple_checkbox` *Create Semantic UI multiple checkbox*

Description

This creates a multiple checkbox using Semantic UI styles.

Usage

```
multiple_checkbox(
```

```
  name,
  label,
  choices,
  choices_value = choices,
  selected = NULL,
  position = "grouped",
  type = NULL,
  ...
)

multiple_radio(
  name,
  label,
  choices,
  choices_value = choices,
  selected = choices_value[1],
  position = "grouped",
  type = "radio",
  ...
)
```

Arguments

name	Input name. Reactive value is available under input[[name]].
label	Text to be displayed with checkbox.
choices	Vector of labels to show checkboxes for.
choices_value	Vector of values that should be used for corresponding choice. If not specified, choices is used by default.
selected	The value(s) that should be chosen initially. If NULL the first one from choices is chosen.
position	Specified checkmarks setup. Can be grouped or inline.
type	Type of checkbox or radio.
...	Other arguments to be added as attributes of the tag (e.g. style, childrens etc.)

Details

The following types are allowed:

- NULLThe standard checkbox (default)
- toggleEach checkbox has a toggle form
- sliderEach checkbox has a simple slider form

Examples

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

```
  library(shiny)
  library(shiny.semantic)

  ui <- function() {
      shinyUI(
        semanticPage(
          title = "Checkbox example",
          suppressDependencies("bootstrap"),
          h1("Checkboxes"),
          multiple_checkbox("checkboxes", "Select Letters", LETTERS[1:6], value = "A"),
          p("Selected letters:"),
          textOutput("selected_letters"),
          tags$br(),
          h1("Radioboxes"),
          multiple_radio("radioboxes", "Select Letter", LETTERS[1:6], value = "A"),
          p("Selected letter:"),
          textOutput("selected_letter")
       )
     )
  }

  server <- shinyServer(function(input, output) {
     output$selected_letters <- renderText(paste(input$checkboxes, collapse = ", "))
     output$selected_letter <- renderText(input$radioboxes)
  })

  shinyApp(ui = ui(), server = server)
}
```

`parse_val` *Parse the 'shiny_input' value from JSON*

Description

Parse the 'shiny_input' value from JSON

Usage

```
parse_val(val)
```

Arguments

`val`	value to get from JSON

Value

Value of type defined in 'shiny_input'

register_search *Register search api url*

Description

Calls Shiny session's registerDataObj to create REST API. Publishes any R object as a URL endpoint that is unique to Shiny session. search_query must be a function that takes two arguments: data (the value that was passed into registerDataObj) and req (an environment that implements the Rook specification for HTTP requests). search_query will be called with these values whenever an HTTP request is made to the URL endpoint. The return value of search_query should be a list of list or a dataframe. Note that different semantic components expect specific JSON fields to be present in order to work correctly. Check components documentation for details.

Usage

```
register_search(session, data, search_query)
```

Arguments

session	Shiny server session
data	Data (the value that is passed into registerDataObj)
search_query	Function providing a response as a list of lists or dataframe of search results.

search_field *Create search field Semantic UI component*

Description

This creates a default search field using Semantic UI styles with Shiny input. Search field is already initialized and available under input[[name]]. Search will automatically route to the named API endpoint provided as parameter. API response is expected to be a JSON with property fields 'title' and 'description'. See https://semantic-ui.com/modules/search.html#behaviors for more details.

Usage

```
search_field(name, search_api_url, default_text = "Search", value = "")
```

Arguments

name	Input name. Reactive value is available under input[[name]].
search_api_url	Register custom API url with server JSON Response containing fields 'title' and 'description'.
default_text	Text to be visible on serach field when nothing is selected.
value	Pass value if you want to initialize selection for search field.

Examples

```
## Only run examples in interactive R sessions
## Not run:
if (interactive()) {
library(shiny)
library(shiny.semantic)
library(gapminder)
library(dplyr)

ui <- function() {
  shinyUI(
    semanticPage(
      title = "Dropdown example",
      uiOutput("search_letters"),
      p("Selected letter:"),
      textOutput("selected_letters")
   )
  )
}

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

 search_api <- function(gapminder, q) {
   has_matching <- function(field) {
     startsWith(field, q)
   }
   gapminder %>%
     mutate(country = as.character(country)) %>%
     select(country) %>%
     unique %>%
     filter(has_matching(country)) %>%
     head(5) %>%
     transmute(title = country,
               description = country)
 }

 search_api_url <- register_search(session, gapminder, search_api)
 output$search_letters <- shiny::renderUI(
   search_field("search_result", search_api_url)
 )
 output$selected_letters <- renderText(input[["search_result"]])
})
}

shinyApp(ui = ui(), server = server)

## End(Not run)
```

`search_selection_api` *Add Semantic UI search selection dropdown based on REST API*

Description

Define the (multiple) search selection dropdown input for retrieving remote selection menu content from an API endpoint. API response is expected to be a JSON with property fields ‘name‘ and ‘value‘. Using a search selection dropdown allows to search more easily through large lists.

Usage

```
search_selection_api(
  name,
  search_api_url,
  multiple = FALSE,
  default_text = "Select"
)
```

Arguments

name	Input name. Reactive value is available under input[[name]].
search_api_url	Register API url with server JSON Response containing fields ‘name‘ and ‘value‘.
multiple	TRUE if the dropdown should allow multiple selections, FALSE otherwise (default FALSE).
default_text	Text to be visible on dropdown when nothing is selected.
	#’ @examples ## Only run examples in interactive R sessions if (interactive()) library(shiny) library(shiny.semantic) library(gapminder) library(dplyr) ui <- function() shinyUI(semanticPage(title = "Dropdown example", uiOutput("search_letters"), p("Selected letter:"), textOutput("selected_letters"))) server <- shinyServer(function(input, output, session) search_api <- function(gapminder, q) has_matching <- function(field) startsWith(field, q) gapminder mutate(country = as.character(country)) select(country) unique filter(has_matching(country)) head(5) transmute(name = country, value = country) search_api_url <- shiny.semantic::register_search(session, gapminder, search_api) output$search_letters <- shiny::renderUI(search_selection_api("search_result", search_api_url, multiple = TRUE)) output$selected_letters <- renderText(input[["search_result"]])) shinyApp(ui = ui(), server = server)

---

search_selection_choices

Add Semantic UI search selection dropdown based on provided choices

---

Description

Define the (multiple) search selection dropdown input component serving search options using provided choices.

Usage

```
search_selection_choices(
  name,
  choices,
  value = NULL,
  multiple = FALSE,
  default_text = "Select",
  groups = NULL,
  dropdown_settings = list(forceSelection = FALSE)
)
```

Arguments

name	Input name. Reactive value is available under input[[name]].
choices	Vector or a list of choices to search through.
value	String with default values to set when initialize the component. Values should be delimited with a comma when multiple to set. Default NULL.
multiple	TRUE if the dropdown should allow multiple selections, FALSE otherwise (default FALSE).
default_text	Text to be visible on dropdown when nothing is selected.
groups	Vector of length equal to choices, specifying to which group the choice belongs. Specifying the parameter enables group dropdown search implementation.
dropdown_settings	Settings passed to dropdown() semantic-ui method. See https://semantic-ui.com/modules/dropdown.html#

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {
  library(shiny)
  library(shiny.semantic)

  ui <- function() {
    shinyUI(
      semanticPage(
        title = "Dropdown example",
        uiOutput("search_letters"),
        p("Selected letter:"),
        textOutput("selected_letters")
      )
    )
  }

  server <- shinyServer(function(input, output, session) {
    choices <- LETTERS
    output$search_letters <- shiny::renderUI(
      search_selection_choices("search_result", choices, multiple = TRUE)
    )
```

```
    output$selected_letters <- renderText(input[["search_result"]])
  })

  shinyApp(ui = ui(), server = server)
}
```

`semanticPage` *Semantic UI page*

Description

This creates a Semantic page for use in a Shiny app.

Usage

```
semanticPage(..., title = "", theme = NULL)
```

Arguments

`...`	Other arguments to be added as attributes of the main div tag wrapper (e.g. style, class etc.)
`title`	A title to display in the browser's title bar.
`theme`	Theme name or path. Full list of supported themes you will find in `SUPPORTED_THEMES` or at http://semantic-ui-forest.com/themes.

Details

Inside, it uses two crucial options:

(1) `shiny.minified` with a logical value, tells whether it should attach min or full semnatic css or js (TRUE by default). (2) `shiny.custom.semantic` if this option has not NULL character `semanticPage` takes dependencies from custom css and js files specified in this path (NULL by default). Depending on `shiny.minified` value the folder should contain either "min" or standard version. The folder should contain: `semantic.css` and `semantic.js` files, or `semantic.min.css` and `semantic.min.js` in `shiny.minified = TRUE` mode.

`semantic_palette` *Semantic colors https://github.com/Semantic-Org/Semantic-UI/blob/master/src/themes/default/globals/site.variables*

Description

Semantic colors https://github.com/Semantic-Org/Semantic-UI/blob/master/src/themes/default/globals/site.variables

Usage

```
semantic_palette
```

Format

An object of class `character` of length 13.

`set_tab_id` *Sets tab id if not provided*

Description

Sets tab id if it wasn't provided

Usage

```
set_tab_id(tab)
```

Arguments

`tab`	A tab. Tab is a list of three elements - first element defines menu item, second element defines tab content, third optional element defines tab id.

`shiny.semantic` *Semantic UI wrapper for Shiny*

Description

With this library it's easy to wrap Shiny with Semantic UI components. Add a few simple lines of code and some CSS classes to give your UI a fresh, modern and highly interactive look.

Options

There are a number of global options that affect shiny.semantic as well as Shiny behavior.The options can be set globally with 'options()'

shiny.custom.semantic.cdn (defaults is internal CDN) This controls from where the css and javascripts will be downloaded.

shiny.semantic.local (defaults to 'FALSE') This allows to use only local dependency.

shiny.custom.semantic (defaults to 'NULL') This allows to set custom local path to semantic dependencies.

shiny.minified (defaults to 'TRUE') Defines including JavaScript as a minified or un-minified file.

shiny_input *Create universal Shiny input binding*

Description

Universal binding for Shiny input on custom user interface. Using this function one can create various inputs ranging from text, numerical, date, dropdowns, etc. Value of this input is extracted via jQuery using $().val() function and default exposed as serialized JSON to the Shiny server. If you want to change type of exposed input value specify it via type param. Currently list of supported types is "JSON" (default) and "text".

Usage

```
shiny_input(input_id, shiny_ui, value = NULL, type = "JSON")
```

Arguments

input_id	String with name of this input. Access to this input within server code is normal with input[[input_id]].
shiny_ui	UI of HTML component presenting this input to the users. This UI should allow to extract its value with jQuery $().val() function.
value	An optional argument with value that should be set for this input. Can be used to store persisten input valus in dynamic UIs.
type	Type of input value (could be "JSON" or "text").

shiny_text_input *Create universal Shiny text input binding*

Description

Universal binding for Shiny text input on custom user interface. Value of this input is extracted via jQuery using $().val() function. This function is just a simple binding over shiny_input. Please take a look at shiny_input documentation for more information.

Usage

```
shiny_text_input(...)
```

Arguments

...	Possible arguments are the same as in shiny_input() method: input_id, shiny_ui, value. Type is already predefined as "text"

show_modal *Show, Hide or Remove Semantic UI modal*

Description

This displays a hidden Semantic UI modal.

Usage

```
show_modal(id, session = shiny::getDefaultReactiveDomain())

remove_modal(id, session = shiny::getDefaultReactiveDomain())

hide_modal(id, session = shiny::getDefaultReactiveDomain())
```

Arguments

id	ID of the modal that will be displayed.
session	The session object passed to function given to shinyServer.

simple_checkbox *Create Semantic UI checkbox*

Description

This creates a checkbox using Semantic UI styles.

Usage

```
simple_checkbox(id, label, type = NULL, is_marked = TRUE, style = NULL)
```

Arguments

id	Input name. Reactive value is available under input[[name]].
label	Text to be displayed with checkbox.
type	Type of checkbox.
is_marked	Defines if checkbox should be marked. Default TRUE.
style	Style of the widget.

Details

The inputs are updateable by using updateCheckboxInput.

The following types are allowed:

- NULLThe standard checkbox (default)
- toggleEach checkbox has a toggle form
- sliderEach checkbox has a simple slider form

Examples

```
simple_checkbox("example", "Check me", is_marked = FALSE)
```

SUPPORTED_THEMES *Supported semantic themes*

Description

Supported semantic themes

Usage

```
SUPPORTED_THEMES
```

Format

An object of class `character` of length 17.

---

tabset *Create Semantic UI tabs*

Description

This creates tabs with content using Semantic UI styles.

Usage

```
tabset(
  tabs,
  active = NULL,
  id = generate_random_id("menu"),
  menu_class = "top attached tabular",
  tab_content_class = "bottom attached segment"
)
```

Arguments

`tabs`	A list of tabs. Each tab is a list of three elements - first element defines menu item, second element defines tab content, third optional element defines tab id.
`active`	Id of the active tab. If NULL first tab will be active.
`id`	Id of the menu element (default: randomly generated id)
`menu_class`	Class for the menu element (default: "top attached tabular")
`tab_content_class`	Class for the tab content (default: "bottom attached segment")

Details

You may access active tab id with `input$<id>_tab`.

Examples

```
tabset(list(
list(menu = shiny::div("First link"),
     content = shiny::div("First content")),
list(menu = shiny::div("Second link"),
     content = shiny::div("Second content"))
))
```

---

uibutton *Create Semantic UI Button*

Description

Create Semantic UI Button

Usage

```
uibutton(name, label, icon = NULL, type = NULL, ...)
```

Arguments

`name`	The `input` slot that will be used to access the value.
`label`	The contents of the button or link
`icon`	An optional `uiicon()` to appear on the button.
`type`	An optional attribute to be added to the button's class.
`...`	Named attributes to be applied to the button

Examples

```
uibutton("simple_button", "Press Me!")
```

`uicalendar` *Create Semantic UI Calendar*

Description

This creates a default calendar input using Semantic UI. The input is available under `input[[name]]`.

This function updates the date on a calendar

Usage

```
uicalendar(
  name,
  value = NULL,
  placeholder = NULL,
  type = "date",
  min = NA,
  max = NA
)

update_calendar(session, id, value = NULL, min = NULL, max = NULL)
```

Arguments

`name`	Input name. Reactive value is available under `input[[name]]`.
`value`	The date to be set, default NULL
`placeholder`	Text visible in the input when nothing is inputted.
`type`	Select from `'year'`, `'month'`, `'date'` and `'time'`
`min`	Minimum date that will be possible to set, default NULL
`max`	Maximum date that will be possible to set, default NULL
`session`	The `session` object passed to function given to `shinyServer`.
`id`	ID of the calendar that will be updated

Examples

```
# Basic calendar
if (interactive()) {

  library(shiny)
  library(shiny.semantic)

  ui <- shinyUI(
    semanticPage(
      title = "Calendar example",
      uicalendar("date"),
      p("Selected date:"),
      textOutput("selected_date")
```

```
    )
  )

  server <- shinyServer(function(input, output, session) {
    output$selected_date <- renderText(
      as.character(input$date)
    )
  })

  shinyApp(ui = ui, server = server)
}

## Not run:
# Calendar with max and min
uicalendar(
  name = "date_finish",
  placeholder = "Select End Date",
  min = "2019-01-01",
  max = "2020-01-01"
)

# Selecting month
uicalendar(
  name = "month",
  type = "month"
)

## End(Not run)
```

`uicard` *Create Semantic UI card tag*

Description

This creates a card tag using Semantic UI styles.

Usage

```
uicard(..., class = "")
```

Arguments

`...`	Other arguments to be added as attributes of the tag (e.g. style, class or childrens etc.)
`class`	Additional classes to add to html tag.

uicards *Create Semantic UI cards tag*

Description

This creates a cards tag using Semantic UI styles.

Usage

```
uicards(..., class = "")
```

Arguments

...	Other arguments to be added as attributes of the tag (e.g. style, class or childrens etc.)
class	Additional classes to add to html tag.

uicheckbox *Create Semantic UI checkox*

Description

This creates a checkbox using Semantic UI styles.

Usage

```
uicheckbox(..., type = "")
```

Arguments

...	Other arguments to be added as attributes of the tag (e.g. style, childrens etc.)
type	Type of checkbox to be used. The following types are allowed: • NULLThe standard checkbox (default) • toggleCheckbox with toggle form • sliderCheckbox with slider form

uidropdown *Create Semantic UI Dropdown*

Description

This creates a dropdown using Semantic UI.

Usage

```
uidropdown(..., type = "", name, is_menu_item = FALSE, dropdown_specs = list())
```

Arguments

...	Dropdown content.
type	Type of the dropdown. Look at https://semantic-ui.com/modules/dropdown.html for all possibilities.
name	Unique name of the created dropdown.
is_menu_item	TRUE if the dropdown is a menu item. Default is FALSE.
dropdown_specs	A list of dropdown functionalities. Look at https://semantic-ui.com/modules/dropdown.html#/settings for all possibilities.

Examples

```
uidropdown(
  "Dropdown menu",
  uiicon(type = "dropdown"),
  uimenu(
    menu_header("Header"),
    menu_divider(),
    menu_item("Option 1"),
    menu_item("Option 2")
  ),
  name = "dropdown_menu",
  dropdown_specs = list("duration: 500")
)
```

uifield *Create Semantic UI field tag*

Description

This creates a field tag using Semantic UI styles.

Usage

```
uifield(..., class = "")
```

Arguments

...	Other arguments to be added as attributes of the tag (e.g. style, class or childrens etc.)
class	Additional classes to add to html tag.

---

uifields *Create Semantic UI fields tag*

Description

This creates a fields tag using Semantic UI styles.

Usage

```
uifields(..., class = "")
```

Arguments

...	Other arguments to be added as attributes of the tag (e.g. style, class or childrens etc.)
class	Additional classes to add to html tag.

---

uiform *Create Semantic UI form tag*

Description

This creates a form tag using Semantic UI styles.

Usage

```
uiform(..., class = "")
```

Arguments

...	Other arguments to be added as attributes of the tag (e.g. style, class or childrens etc.)
class	Additional classes to add to html tag.

uiheader *Create Semantic UI header*

Description

This creates a header with optional icon using Semantic UI styles.

Usage

```
uiheader(title, description, icon = NULL)
```

Arguments

title	Header title
description	Subheader text
icon	Optional icon name

uiicon *Create Semantic UI icon tag*

Description

This creates an icon tag using Semantic UI styles.

Usage

```
uiicon(type = "", ...)
```

Arguments

type	A name of an icon. Look at http://semantic-ui.com/elements/icon.html for all possibilities.
...	Other arguments to be added as attributes of the tag (e.g. style, class etc.)

uiinput *Create Semantic UI Input*

Description

This creates an input shell for the actual input

Usage

```
uiinput(..., class = "")
```

Arguments

...	Other arguments to be added as attributes of the tag (e.g. style, class or childrens etc.)
class	Additional classes to add to html tag.

See Also

uitextinput

Examples

```
library(shiny)
library(shiny.semantic)

# Text input
uiinput(
  tags$label("Text input"),
  uitextinput("ex", type = "text", placeholder = "Enter Text")
)
```

uilabel *Create Semantic UI label tag*

Description

This creates a label tag using Semantic UI.

Usage

```
uilabel(..., type = "", is_link = TRUE)
```

Arguments

...	Other arguments to be added such as content of the tag (text, icons) and/or attributes (style)
type	Type of the label. Look at https://semantic-ui.com/elements/label.html for all possibilities.
is_link	If TRUE creates label with 'a' tag, otherwise with 'div' tag. #'

---

uilist *Create Semantic UI list with header, description and icons*

Description

This creates a list with icons using Semantic UI

Usage

```
uilist(data, is_icon = FALSE, is_divided = FALSE, is_description = FALSE)
```

Arguments

data	A dataframe with columns 'header' and/or 'description', 'icon' containing the list items headers, descriptions and icons. 'description' column is optional and should be provided if 'is_description' parameter TRUE. 'icon' column is optional and should be provided if 'is_icon' parameter TRUE. Icon column should contain strings with icon names available here: https://semantic-ui.com/elements/icon.html
is_icon	IF TRUE created list has icons
is_divided	If TRUE created list elements are divided
is_description	If TRUE created list will have a description

Examples

```
list_content <- data.frame(
  header = paste("Header", 1:5),
  description = paste("Description", 1:5),
  icon = paste("home", 1:5),
  stringsAsFactors = FALSE
)

# Create a 5 element divided list with alert icons and description
uilist(list_content, is_icon = TRUE, is_divided = TRUE, is_description = TRUE)
```

uimenu *Create Semantic UI Menu*

Description

This creates a menu using Semantic UI.

Usage

```
uimenu(..., type = "")
```

Arguments

...	Menu items to be created. Use menu_item function to create new menu item. Use uidropdown(is_menu_item = TRUE, ...) function to create new dropdown menu item. Use menu_header and menu_divider functions to customize menu format.
type	Type of the menu. Look at https://semantic-ui.com/collections/menu.html for all possiblities.

Examples

```
if (interactive()) {
library(shiny)
library(shiny.semantic)

ui <- function() {
  shinyUI(
    semanticPage(
      title = "My page",
      suppressDependencies("bootstrap"),
      uimenu(menu_item("Menu"),
             uidropdown(
               "Action",
               uimenu(
                 menu_header(uiicon("file"), "File", is_item = FALSE),
                 menu_item(uiicon("wrench"), "Open"),
                 menu_item(uiicon("upload"), "Upload"),
                menu_item(uiicon("remove"), "Upload"),
                 menu_divider(),
                 menu_header(uiicon("user"), "User", is_item = FALSE),
                 menu_item(uiicon("add user"), "Add"),
                 menu_item(uiicon("remove user"), "Remove")),
               type = "",
               name = "unique_name",
               is_menu_item = TRUE),
           menu_item(uiicon("user"), "Profile", href = "#index", item_feature = "active"),
             menu_item("Projects", href = "#projects"),
```

```
              menu_item(uiicon("users"), "Team"),
              uimenu(menu_item(uiicon("add icon"), "New tab"), type = "right"))
    )
  )
}

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

shinyApp(ui = ui(), server = server)
}
```

`uimessage` *Create Semantic UI Message*

Description

Create Semantic UI Message

Usage

```
uimessage(header, content, type = "", icon, closable = FALSE)
```

Arguments

`header`	Header of the message
`content`	Content of the message. If it is a vector, creates a list of vector's elements
`type`	Type of the message. Look at https://semantic-ui.com/collections/message.html for all possibilities.
`icon`	If the message is of the type 'icon', specify the icon. Look at http://semantic-ui.com/elements/icon.html for all possibilities.
`closable`	Determines whether the message should be closable. Default is FALSE - not closable

---

`uinumericinput` *Create Semantic UI Numeric Input*

Description

This creates a default numeric input using Semantic UI. The input is available under `input[[name]]`.

Usage

```
uinumericinput(name, value, min = NA, max = NA, step = NA)
```

Arguments

name	Input name. Reactive value is available under input[[name]].
value	Initial value of the numeric input.
min	Minimum allowed value.
max	Maximum allowed value.
step	Interval to use when stepping between min and max.

Details

The inputs are updateable by using updateNumericInput.

Examples

```
library(shiny)
library(shiny.semantic)

# Text input
uiinput(
  tags$label("Numeric Input"),
  uinumericinput("ex", 10)
)
```

---

uirender *Render semanticui htmlwidget*

Description

htmlwidget that adds semanticui dependencies and renders in viewer or rmarkdown.

Usage

```
uirender(ui, width = NULL, height = NULL, element_id = NULL)
```

Arguments

ui	UI, which will be wraped in an htmlwidget.
width	Fixed width for widget (in css units). The default is NULL, which results in intelligent automatic sizing.
height	Fixed height for widget (in css units). The default is NULL, which results in intelligent automatic sizing.
element_id	Use an explicit element ID for the widget (rather than an automatically generated one).

uisegment *Create Semantic UI segment*

Description

This creates a segment using Semantic UI styles.

Usage

```
uisegment(..., class = "")
```

Arguments

...	Other arguments to be added as attributes of the tag (e.g. style, class or childrens etc.)
class	Additional classes to add to html tag.

uislider *Create Semantic UI Slider*

Description

This creates a slider input using Semantic UI. Slider is already initialized and available under `input[[name]]`.

Usage

```
uislider(name, value, min, max, step = 1, type = NULL)

uirange(name, value, value2, min, max, step = 1, type = NULL)
```

Arguments

name	Input name. Reactive value is available under `input[[name]]`.
value	The initial value to be selected for the sldier (lower value if using range).
min	The minimum value allowed to be selected for the slider.
max	The maximum value allowed to be selected for the slider.
step	The interval between each selectable value of the slider.
type	UI class of the slider. Can include `"Labeled"` and `"ticked"`.
value2	The initial upper value of the slider.

Details

Use `update_slider` to update the slider/range within the shiny session.

See Also

update_slider for input updates, https://fomantic-ui.com/modules/slider.html for preset classes.

Examples

```
if (interactive()) {

  library(shiny)
  library(shiny.semantic)

  # Slider example
  ui <- shinyUI(
    semanticPage(
      title = "Slider example",
      tags$br(),
      uislider("slider", 10, 0, 20),
      p("Selected value:"),
      textOutput("slider")
    )
  )

   server <- shinyServer(function(input, output, session) {
     output$slider <- renderText(input$slider)
   })

   shinyApp(ui = ui, server = server)

   # Range example
   ui <- shinyUI(
     semanticPage(
       title = "Range example",
       tags$br(),
       uirange("range", 10, 15, 0, 20),
       p("Selected values:"),
       textOutput("range")
     )
   )

   server <- shinyServer(function(input, output, session) {
     output$range <- renderText(paste(input$range, collapse = " - "))
   })

   shinyApp(ui = ui, server = server)

 }
```

---

uitextinput *Create Semantic UI Text Input*

Description

This creates a default text input using Semantic UI. The input is available under `input[[name]]`.

Usage

```
uitextinput(
  name,
  value = "",
  type = "text",
  placeholder = NULL,
  attribs = list()
)
```

Arguments

`name`	Input name. Reactive value is available under `input[[name]]`.
`value`	Pass value if you want to have default text.
`type`	Change depending what type of input is wanted. See details for options.
`placeholder`	Text visible in the input when nothing is inputted.
`attribs`	A named list of attributes to assign to the input.

Details

The following types are allowed:

- textThe standard input
- textareaAn extended space for text
- passwordA censored version of the text input
- emailA special version of the text input specific for email addresses
- urlA special version of the text input specific for URLs
- telA special version of the text input specific for telephone numbers

The inputs are updateable by using `updateTextInput` or `updateTextAreaInput` if `type = "textarea"`.

Examples

```
library(shiny)
library(shiny.semantic)

# Text input
uiinput(
  tags$label("Text input"),
  uitextinput("ex", type = "text", placeholder = "Enter Text")
)
```

update_dropdown *Update dropdown Semantic UI component*

Description

Change the value of a dropdown input on the client.

Usage

```
update_dropdown(
  session,
  name,
  choices = NULL,
  choices_value = choices,
  value = NULL
)
```

Arguments

session	The session object passed to function given to shinyServer.
name	The id of the input object
choices	All available options one can select from. If no need to update then leave as NULL
choices_value	What reactive value should be used for corresponding choice.
value	The initially selected value.

update_slider *Update slider Semantic UI component*

Description

Change the value of a uislider input on the client.

Usage

```
update_slider(session, name, value)

update_range(session, name, value, value2)
```

Arguments

session	The session object passed to function given to shinyServer.
name	The id of the input object
value	The value to be selected for the sldier (lower value if using range).
value2	The upper value of the slider.

See Also

uislider

`%:::%` *::: hack solution*

Description

::: hack solution

Usage

```
pkg %:::% name
```

Arguments

`pkg`	package name
`name`	function name

Value

function

Index

∗Topic **datasets**
checkbox_positions, 4
semantic_palette, 22
SUPPORTED_THEMES, 26
.onLoad, 3
%:::%, 43

attach_rule, 3

check_proper_color, 4
check_semantic_theme, 5
checkbox_positions, 4
create_modal, 5

date_input, 6
define_selection_type, 7
dropdown, 7, *42*

get_cdn_path, 9
get_default_semantic_theme, 9
get_dependencies, 10

hide_modal *(*show_modal*)*, 25

label, 10
list_element, 11

menu_divider, 11
menu_header, 12
menu_item, 12
modal, 13
multiple_checkbox, 15
multiple_radio *(*multiple_checkbox*)*, 15

parse_val, 17

register_search, 18
remove_modal *(*show_modal*)*, 25

search_field, 18
search_selection_api, 19
search_selection_choices, 20
semantic_palette, 22
semanticPage, 22
set_tab_id, 23
shiny.semantic, 23
shiny_input, 24
shiny_text_input, 24
show_modal, 25
simple_checkbox, 25
SUPPORTED_THEMES, 26

tabset, 26

uibutton, 27
uicalendar, 28
uicard, 29
uicards, 30
uicheckbox, 30
uidropdown, 31
uifield, 31
uifields, 32
uiform, 32
uiheader, 33
uiicon, *27*, 33
uiinput, 34
uilabel, 34
uilist, 35
uimenu, 36
uimessage, 37
uinumericinput, 37
uirange *(*uislider*)*, 39
uirender, 38
uisegment, 39
uislider, 39, *42*
uitextinput, 40
update_calendar *(*uicalendar*)*, 28
update_dropdown, 42
update_range *(*update_slider*)*, 42
update_slider, *39*, 42
updateCheckboxInput, *25*

updateNumericInput, *38*
updateTextAreaInput, *41*
updateTextInput, *41*