Package 'shiny.semantic'

May 11, 2018

Type Package

Title Semantic UI Support for Shiny

Version 0.2.1

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

Suggests dplyr, gapminder, testthat, lintr

RoxygenNote 6.0.1

NeedsCompilation no

Author Filip Stachura [aut, cre]

Maintainer Filip Stachura <filip@appsilondatascience.com>

Repository CRAN

Date/Publication 2018-05-11 10:38:06 UTC

R topics documented:

`CDN_PATH` *Cloudfront path*

Description

Cloudfront path

Usage

```
CDN_PATH
```

Format

An object of class `character` of length 1.

`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

`check_semantic_theme` *Semantic theme path validator*

Description

Semantic theme path validator

Usage

```
check_semantic_theme(theme_css)
```

Arguments

`theme_css`	it can be either NULL, character with css path, or theme name

Value

path to theme

Examples

```
check_semantic_theme(NULL)
check_semantic_theme("darkly")
```

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)
```

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.

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_default_semantic_js` *Get default semantic js*

Description

Get default semantic js

Usage

```
get_default_semantic_js()
```

Value

path to default js semantic file

get_default_semantic_theme *Get default semantic css*

Description

Get default semantic css

Usage

```
get_default_semantic_theme()
```

Value

path to default css semantic file

get_dependencies *Add dashboard dependencies to html*

Description

Internal function that adds dashboard dependencies to html.

Usage

```
get_dependencies()
```

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, icon, row)
```

Arguments

data	data to list
is_description	description flag
icon	Icon character
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.

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")
```

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 wirh 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.

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

`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.

---

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"

---

slider_input *Add Semantic UI slider component*

Description

It implements slider extension provided by: https://github.com/tyleryasaka/semantic-ui-range

Usage

```
slider_input(name, min, max, value, step = 0.01, n_ticks = 5, color = "")
```

Arguments

name	Input name. Reactive value is available under input[[name]].
min	Minimum possible value to set.
max	Miaksimum possible value to set.
value	Initial value.
step	Possible step for slider inputs.
n_ticks	Specifies how many ticks the slider should display.
color	Possible slider color, see: semantic_palette.

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, 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 two elements - first element defines menu item, second element defines tab content.
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")

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.

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.)

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, icon, is_divided = FALSE, is_description = FALSE)
```

Arguments

data	A dataframe with columns 'header' and/or 'description' containing the list items headers and descriptions. 'description' column is optional and should be provided if 'is_description' parameter TRUE.
icon	A string with icon name. Empty string will render list without 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),
  stringsAsFactors = FALSE
)

# Create a 5 element divided list with alert icons and description
uilist(list_content, "alert", 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

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.

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

Description

::: hack solution

Usage

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

Arguments

pkg	package name
name	function name

Value

function

Index