

# Package: shidashi (via r-universe)

June 16, 2024

**Type** Package

**Title** A Shiny Dashboard Template System

**Version** 0.1.6

**Language** en-US

**URL** <https://dipterix.org/shidashi/>,  
<https://github.com/dipterix/shidashi>

**BugReports** <https://github.com/dipterix/shidashi/issues>

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.1

**Description** A template system based on 'AdminLTE3' (<https://adminlte.io/themes/v3/>) theme. Comes with default theme that can be easily customized. Developers can upload modified templates on 'Github', and users can easily download templates with 'RStudio' project wizard. The key features of the default template include light and dark theme switcher, resizing graphs, synchronizing inputs across sessions, new notification system, fancy progress bars, and card-like flip panels with back sides, as well as various of 'HTML' tool widgets.

**Imports** utils, digest (>= 0.6.27), fastmap (>= 1.1.0), formatR (>= 1.11), httr (>= 1.4.2), shiny (>= 1.7.0), yaml (>= 2.2.1), jsonlite (>= 1.7.2)

**Suggests** htmltools (>= 0.5.2), logger (>= 0.2.1), rstudioapi (>= 0.13), ggplot2, ggExtra

**Repository** <https://dipterix.r-universe.dev>

**RemoteUrl** <https://github.com/dipterix/shidashi>

**RemoteRef** HEAD

**RemoteSha** 6153b1ab8491a477b5c84610528c5e1a99cb24bd

## Contents

accordion	2
accordion_item	4
add-remove-html-class	5
adminlte	6
as_badge	7
as_icon	7
back_top_button	8
card	9
card_tabset	12
card_tabset_operate	14
card_tool	15
clipboardOutput	16
flex_container	17
flip_box	19
format_text_r	20
get_construct_string	21
guess_body_class	22
include_view	22
info_box	23
javascript-tunnel	24
module_info	26
notification	27
progressOutput	29
register_global_reactiveValues	31
render	31
reset_output	32
shiny_progress	33
show_ui_code	34
template_settings	35
use_template	36
<b>Index</b>	<b>38</b>

---

accordion	<i>Generates an 'accordion' tab-set</i>
-----------	---

---

### Description

Generates an 'accordion' tab-set that only one tab is expanded at a time. This feature is experimental and has bugs in some situations. Please use it at your own risk.

**Usage**

```

accordion(
  ...,
  id = rand_string(prefix = "accordion-"),
  class = NULL,
  style_header = NULL,
  style_body = NULL,
  env = parent.frame(),
  extras = list(),
  root_path = template_root()
)

accordion_operate(
  id,
  itemId,
  item_title,
  method = c("expand", "collapse", "toggle"),
  session = shiny::getDefaultReactiveDomain()
)

```

**Arguments**

...	'accordion' items, generated by <a href="#">accordion_item</a>
id	the element id, must be unique
class	the additional 'HTML' class
style_header	additional 'CSS' styles for header
style_body	additional 'CSS' styles for content body
env	environment to evaluate ...
extras	key-value pairs that overrides the parameters in <a href="#">accordion_item</a>
root_path	see <a href="#">template_root</a>
itemId	<a href="#">accordion_item</a> id
item_title	<a href="#">accordion_item</a> title, if item id is specified, this title will be ignored
method	operation, choices are 'expand' (default), 'collapse', or 'toggle'
session	shiny session

**Value**

'shiny.tag.list' 'HTML' tags

**See Also**

[accordion\\_item](#)

## Examples

```
if(interactive()) {  
  
  library(shiny)  
  library(shidashi)  
  
  accordion(  
    id = "input-set",  
    accordion_item(  
      title = "Input Group A",  
      textInput("input_1", "Input 1"),  
      collapsed = FALSE,  
      footer = "Anim pariatur cliche reprehenderit dolor brunch."  
    ),  
    accordion_item(  
      title = "Input Group B",  
      textInput("input_2", "Input 2"),  
      footer = actionButton("btn1", "OK"),  
      collapsed = FALSE  
    )  
  )  
}
```

---

accordion_item	<i>'Accordion' items</i>
----------------	--------------------------

---

## Description

*'Accordion' items*

## Usage

```
accordion_item(  
  title,  
  ...,  
  footer = NULL,  
  class = "",  
  collapsed = TRUE,  
  parentId = rand_string(prefix = "accordion-"),  
  itemId = rand_string(prefix = "accordion-item-"),  
  style_header = NULL,  
  style_body = NULL,  
  root_path = template_root()  
)
```

**Arguments**

title	character title to show in the header
...	body content
footer	footer element, hidden if NULL
class	the class of the item
collapsed	whether collapsed at the beginning
parentId	parent <a href="#">accordion</a> id
itemId	the item id
style_header, style_body	'CSS' style of item header and body
root_path	see <code>template_root</code>

**Value**

'shiny.tag.list' 'HTML' tags

**See Also**

[accordion](#)

---

add-remove-html-class *Add or remove 'HTML' class from 'RAVE' application*

---

**Description**

Only works in template framework provided by 'shidashi' package, see [use\\_template](#)

**Usage**

```
add_class(selector, class, session = shiny::getDefaultReactiveDomain())
```

```
remove_class(selector, class, session = shiny::getDefaultReactiveDomain())
```

**Arguments**

selector	'CSS' selector
class	class to add or to remove from selected elements
session	shiny session

**Value**

No value is returned

## Examples

```
server <- function(input, output, session){  
  
  # Add class `hidden` to element with ID `elemid`  
  add_class("#elemid", "hidden")  
  
  # Remove class `hidden` from element with class `shiny-input-optional`  
  remove_class(".shiny-input-optional", "hidden")  
}
```

---

adminlte

*Generates 'AdminLTE' theme-related 'HTML' tags*

---

## Description

These functions should be called in 'HTML' templates. Please see vignettes for details.

## Usage

```
adminlte_ui(root_path = template_root())  
  
adminlte_sidebar(  
  root_path = template_root(),  
  settings_file = "modules.yaml",  
  shared_id = rand_string(26)  
)
```

## Arguments

`root_path` the root path of the website project; see [template\\_settings](#)  
`settings_file` the settings file containing the module information  
`shared_id` a shared identification by session to synchronize the inputs; assigned internally.

## Value

'HTML' tags

---

as_badge	<i>Generates badge icons</i>
----------	------------------------------

---

**Description**

Usually used along with [card](#), [card2](#), and [card\\_tabset](#). See `tools` parameters in these functions accordingly.

**Usage**

```
as_badge(badge = NULL)
```

**Arguments**

`badge` characters, "shiny.tag" object or NULL

**Details**

When `badge` is NULL or empty, then `as_badge` returns empty strings. When `badge` is a "shiny.tag" object, then 'HTML' class 'right' and 'badge' will be appended. When `badge` is a string, it should follow the syntax of "message|class". The text before "|" will be the badge message, and the text after the "|" becomes the class string.

**Value**

'HTML' tags

**Examples**

```
# Basic usage
as_badge("New")

# Add class `bg-red` and `no-padding`
as_badge("New|bg-red no-padding")
```

---

as_icon	<i>Convert characters, shiny icons into 'fontawesome' 4</i>
---------	---

---

**Description**

Convert characters, shiny icons into 'fontawesome' 4

**Usage**

```
as_icon(icon = NULL, class = "fas")
```

**Arguments**

icon	character or <a href="#">icon</a>
class	icon class; change this when you are using 'fontawesome' professional version. The choices are 'fa' (compatible), 'fas' (strong), 'far' (regular), 'fal' (light), and 'fad' (duo-tone).

**Value**

'HTML' tag

**Examples**

```
if(interactive()){
  as_icon("bookmark", class = "far")
  as_icon("bookmark", class = "fas")

# no icon
as_icon(NULL)
}
```

---

back\_top\_button

*'HTML' code to generate small back-to-top button*


---

**Description**

This function is a template function that should be called in 'HTML' templates before closing the "</body>" tag.

**Usage**

```
back_top_button(icon = "chevron-up", title = "Jump to")
```

**Arguments**

icon	the icon for back-to-top button
title	the expanded menu title

**Value**

'HTML' tags

**Examples**

```
back_top_button()
back_top_button("rocket")
```



---

card

*Card-like 'HTML' element*

---

### Description

Card-like 'HTML' element

### Usage

```
card(  
  title,  
  ...,  
  footer = NULL,  
  tools = NULL,  
  inputId = NULL,  
  class = "",  
  class_header = "",  
  class_body = "",  
  class_foot = "",  
  style_header = NULL,  
  style_body = NULL,  
  start_collapsed = FALSE,  
  resizable = FALSE,  
  root_path = template_root()  
)
```

```
card2(  
  title,  
  body_main,  
  body_side = NULL,  
  footer = NULL,  
  tools = NULL,  
  inputId = NULL,  
  class = "",  
  class_header = "",  
  class_body = "min-height-400",  
  class_foot = "",  
  style_header = NULL,  
  style_body = NULL,  
  start_collapsed = FALSE,  
  root_path = template_root()  
)
```

```
card2_open(inputId, session = shiny::getDefaultReactiveDomain())
```

```
card2_close(inputId, session = shiny::getDefaultReactiveDomain())
```

```
card2_toggle(inputId, session = shiny::getDefaultReactiveDomain())

card_operate(
  inputId,
  title,
  method,
  session = shiny::getDefaultReactiveDomain()
)
```

### Arguments

title	the title of the card
...	the body content of the card
footer	the footer of the card; will be hidden if footer=NULL
tools	a list of tools or badges to be displayed at top-right corner, generated by <a href="#">as_badge</a> or <a href="#">card_tool</a>
inputId	the id of the card
class	the 'HTML' class of the entire card
class_header	the the 'HTML' class of the card header
class_body	the the 'HTML' class of the card body
class_foot	the the 'HTML' class of the card footer
style_header	'CSS' style of the header
style_body	'CSS' style of the body
start_collapsed	whether the card starts as collapsed
resizable	whether the card body can be resized vertically; notice that if true, then the default padding for body will be zero
root_path	see <a href="#">template_root</a>
body_main, body_side	used by card2, the body content of the front and back sides of the card
session	shiny session domain
method	action to expand, minimize, or remove the cards; choices are "collapse", "expand", "remove", "toggle", "maximize", "minimize", and "toggleMaximize"

### Value

'HTML' tags

### Examples

```
library(shiny)
library(shidashi)

# Used for example only
ns <- I
```

```

session <- MockShinySession$new()

# ----- Basic usage -----
card(
  title = "Badges", div(
    class = "padding-20",
    p(
      "Add badges to the top-right corner. ",
      "Use `|` to indicate the badge classes; ",
      "for example: `badge-info`, `badge-warning`..."
    ),
    hr(), p(
      "Use `resizable = TRUE` to make card resizable."
    )
  ),
  tools = list(
    as_badge("New|badge-info"),
    as_badge("3|badge-warning")
  ),
  class_body = "height-300",
  resizable = TRUE
)

# ----- With tools -----
card(
  title = "Default Tools",
  plotOutput(
    ns("card_defaulttool_plot"),
    height = "100%"
  ),
  tools = list(
    card_tool(
      widget = "link",
      href = "https://github.com/dipterix"
    ),
    card_tool(widget = "collapse"),
    card_tool(widget = "maximize")
  ),
  class_body = "height-300",
  resizable = TRUE
)

# ----- Card2 example -----
card2(
  title = "Card2 Example", body_main =
  plotOutput(
    outputId = ns("card2_plot"),
    height = "100%"
  ),
  ),
  body_side = fluidRow(
    column(
      6L, textInput(
        ns("card2_plot_title"),

```

```

      "Plot title"
    )
  ),
  column(
    6L, sliderInput(
      ns("card2_plot_npts"),
      "# of points", min = 1, max = 100,
      value = 10, step = 1, round = TRUE
    )
  )
),
tools = list(
  card_tool(widget = "link",
            href = "https://github.com/dipterix"),
  card_tool(widget = "collapse"),
  card_tool(widget = "maximize")
),
class_body = "height-300"
)

```

---

card\_tabset

*Generates a set of card panels*


---

## Description

To insert, remove, or active card panels, see [card\\_tabset\\_operate](#).

## Usage

```

card_tabset(
  ...,
  inputId = rand_string(prefix = "tabset-"),
  title = NULL,
  names = NULL,
  active = NULL,
  tools = NULL,
  footer = NULL,
  class = "",
  class_header = "",
  class_body = "",
  class_foot = ""
)

```

## Arguments

... 'HTML' tags; each tag will be placed into a card  
inputId the id of the card-set, must start with letters

title	the title of the card-set
names	title of the tabs
active	the title that will be active on load
tools	a list of tools or badges generated by <a href="#">card_tool</a> or <a href="#">as_badge</a>
footer	the footer element of the card-set
class	the 'HTML' class the of card-set
class_header, class_body, class_foot	additional 'HTML' class the of card header, body, and footer accordingly

**Value**

'HTML' tags

**See Also**

[card\\_tabset\\_operate](#)

**Examples**

```
library(shiny)
library(shidashi)

# Fake session to operate on card_tabset without shiny
session <- MockShinySession$new()

card_tabset(
  inputId = "card_set",
  title = "Cardset with Tools",
  `Tab 1` = p("Tab content 1"),
  class_body = "height-500",
  tools = list(
    as_badge(
      "New|badge-success"
    ),
    card_tool(
      widget = "collapse"
    ),
    card_tool(
      widget = "maximize"
    )
  )
)

card_tabset_insert(
  inputId = "card_set",
  title = "Tab 2",
  p("New content"),
  session = session
)
```

```
card_tabset_activate(  
  inputId = "card_set",  
  title = "Tab 1",  
  session = session  
)  
  
card_tabset_remove(  
  inputId = "card_set",  
  title = "Tab 2",  
  session = session  
)
```

---

card\_tabset\_operate    *Add, active, or remove a card within [card\\_tabset](#)*

---

### Description

Add, active, or remove a card within [card\\_tabset](#)

### Usage

```
card_tabset_insert(  
  inputId,  
  title,  
  ...,  
  active = TRUE,  
  notify_on_failure = TRUE,  
  session = shiny::getDefaultReactiveDomain()  
)  
  
card_tabset_remove(  
  inputId,  
  title,  
  notify_on_failure = TRUE,  
  session = shiny::getDefaultReactiveDomain()  
)  
  
card_tabset_activate(  
  inputId,  
  title,  
  notify_on_failure = TRUE,  
  session = shiny::getDefaultReactiveDomain()  
)
```

**Arguments**

inputId	the element id of <a href="#">card_tabset</a>
title	the title of the card to insert, activate, or to remove
...	the content of the card
active	whether to set the card to be active once added
notify_on_failure	whether to show notifications on failure
session	shiny session domain

**Value**

These functions execute `session$sendCustomMessage` and return whatever value generated by that function; usually nothing.

**See Also**

[card\\_tabset](#)

---

card_tool	<i>Generates small icon widgets</i>
-----------	-------------------------------------

---

**Description**

The icons can be displayed at header line within [accordion](#), [card](#), [card2](#), [card\\_tabset](#). See their examples.

**Usage**

```
card_tool(
  inputId = NULL,
  title = NULL,
  widget = c("maximize", "collapse", "remove", "flip", "refresh", "link", "custom"),
  icon,
  class = "",
  href = "#",
  target = "_blank",
  start_collapsed = FALSE,
  ...
)
```

**Arguments**

inputId	the button id, only necessary when widget is "custom"
title	the tip message to show when the mouse cursor hovers on the icon
widget	the icon widget type; choices are "maximize", "collapse", "remove", "flip", "refresh", "link", and "custom"; see 'Details'
icon	icon to use if you are unsatisfied with the default ones
class	additional class for the tool icons
href, target	used when widget is "link", will open an external website; default is open a new tab
start_collapsed	used when widget is "collapse", whether the card should start collapsed
...	passed to the tag as attributes

**Details**

There are 7 widget types:

"maximize" allow the elements to maximize themselves to full-screen

"collapse" allow the elements to collapse

"remove" remove a [card](#) or [card2](#)

"flip" used together with [flip\\_box](#), to allow card body to flip over

"refresh" refresh all shiny outputs

"link" open a hyper-link pointing to external websites

"custom" turn the icon into a `actionButton`. in this case, `inputId` must be specified.

**Value**

'HTML' tags to be included in `tools` parameter in [accordion](#), [card](#), [card2](#), [card\\_tabset](#)

---

clipboardOutput

*Generates outputs that can be written to clipboards with one click*

---

**Description**

Generates outputs that can be written to clipboards with one click



**Usage**

```
clipboardOutput(
  outputId = rand_string(prefix = "clipboard"),
  message = "Copy to clipboard",
  clip_text = "",
  class = NULL,
  as_card_tool = FALSE
)

renderClipboard(
  expr,
  env = parent.frame(),
  quoted = FALSE,
  outputArgs = list()
)
```

**Arguments**

outputId	the output id
message	tool tip to show when mouse hovers on the element
clip_text	the initial text to copy to clipboards
class	'HTML' class of the element
as_card_tool	whether to make the output as <code>card_tool</code>
expr	expression to evaluate; the results will replace clip_text
env	environment to evaluate expr
quoted	whether expr is quoted
outputArgs	used to replace default arguments of clipboardOutput

**Value**

'HTML' elements that can write to clip-board once users click on them.

**Examples**

```
clipboardOutput(clip_text = "Hey there")
```

---

flex_container	<i>Generate 'HTML' tags with 'flex' layout</i>
----------------	--

---

**Description**

Generate 'HTML' tags with 'flex' layout

**Usage**

```

flex_container(
  ...,
  style = NULL,
  direction = c("row", "column"),
  wrap = c("wrap", "nowrap", "wrap-reverse"),
  justify = c("flex-start", "center", "flex-end", "space-around", "space-between"),
  align_box = c("stretch", "flex-start", "center", "flex-end", "baseline"),
  align_content = c("stretch", "flex-start", "flex-end", "space-between", "space-around",
    "center")
)

flex_item(
  ...,
  size = 1,
  style = NULL,
  order = NULL,
  flex = as.character(size),
  align = c("flex-start", "flex-end", "center"),
  class = NULL,
  .class = "fill-width padding-5"
)

flex_break(..., class = NULL)

```

**Arguments**

...	for flex_container, it's elements of flex_item; for flex_item, ... are shiny 'HTML' tags
style	the additional 'CSS' style for containers or inner items
direction, wrap, justify, align_box, align_content	'CSS' styles for 'flex' containers
size	numerical relative size of the item; will be ignored if flex is provided
order, align, flex	CSS' styles for 'flex' items
class, .class	class to add to the elements

**Value**

'HTML' tags

**Examples**

```

x <- flex_container(
  style = "position:absolute;height:100vh;top:0;left:0;width:100%",
  flex_item(style = 'background-color:black;'),
  flex_item(style = 'background-color:red;')
)

```

```
# You can view it via `htmltools::html_print(x)`
```

---

flip_box	An 'HTML' container that can flip
----------	-----------------------------------

---

## Description

An 'HTML' container that can flip

## Usage

```
flip_box(
  front,
  back,
  active_on = c("click", "click-front", "manual"),
  inputId = NULL,
  class = NULL
)

flip(inputId, session = shiny::getDefaultReactiveDomain())
```

## Arguments

front	'HTML' elements to show in the front
back	'HTML' elements to show when the box is flipped
active_on	the condition when a box should be flipped; choices are 'click': flip when double-click on both sides; 'click-front': only flip when the front face is double-clicked; 'manual': manually flip in R code (see {flip(inputId)} function)
inputId	element 'HTML' id; must be specified if active_on is not 'click'
class	'HTML' class
session	shiny session; default is current active domain

## Value

flip\_box returns 'HTML' tags; flip should be called from shiny session, and returns nothing

## Examples

```
# More examples are available in demo

library(shiny)
library(shidashi)

session <- MockShinySession$new()
```

```
flip_box(front = info_box("Side A"),
         back = info_box("Side B"),
         inputId = 'flip_box1')

flip('flip_box1', session = session)
```

---

format\_text\_r

*Get re-formatted R expressions in characters*

---

### Description

Get re-formatted R expressions in characters

### Usage

```
format_text_r(
  expr,
  quoted = FALSE,
  reformat = TRUE,
  width.cutoff = 80L,
  indent = 2,
  wrap = TRUE,
  args.newline = TRUE,
  blank = FALSE,
  ...
)

html_highlight_code(
  expr,
  class = NULL,
  quoted = FALSE,
  reformat = TRUE,
  copy_on_click = TRUE,
  width.cutoff = 80L,
  indent = 2,
  wrap = TRUE,
  args.newline = TRUE,
  blank = FALSE,
  ...,
  hover = c("overflow-visible-on-hover", "overflow-auto")
)
```

### Arguments

expr	R expressions
quoted	whether expr is quoted

reformat            whether to reformat  
 width.cutoff, indent, wrap, args.newline, blank, ...  
                     passed to [tidy\\_source](#)  
 class               class of <pre> tag  
 copy\_on\_click      whether to copy to clipboard if user clicks on the code; default is true  
 hover               mouse hover behavior

**Value**

format\_text\_r returns characters, html\_highlight\_code returns the 'HTML' tags wrapping expressions in <pre> tag

**See Also**

[get\\_construct\\_string](#)

**Examples**

```

s <- format_text_r(print(local({a<-1;a+1})))
cat(s)

x <- info_box("Message", icon = "cogs")
s <- format_text_r(get_construct_string(x),
                  width.cutoff = 15L, quoted = TRUE)
cat(s)

```

---

get\_construct\_string   *Get R expression used to generate the 'HTML' tags*

---

**Description**

This function only works on the elements generated by this package

**Usage**

```
get_construct_string(x)
```

**Arguments**

x                    'HTML' tags

**Value**

Quoted R expressions that can generate the 'HTML' tags

**See Also**[format\\_text\\_r](#)**Examples**

```
x <- info_box("Message")
get_construct_string(x)
```

---

guess_body_class	<i>Guess the 'AdminLTE' body class for modules, used internally</i>
------------------	---

---

**Description**

Guess the 'AdminLTE' body class for modules, used internally

**Usage**

```
guess_body_class(cls)
```

**Arguments**

cls	the class string of the <body> tag in 'index.html'
-----	--

**Value**

The proposed class for <body> tag

---

include_view	<i>Template function to include 'snippets' in the view folder</i>
--------------	---

---

**Description**

Store the reusing 'HTML' segments in the views folder. This function should be used in the 'index.html' template

**Usage**

```
include_view(file, ..., .env = parent.frame(), .root_path = template_root())
```

**Arguments**

file	files in the template views folder
...	ignored
.env, .root_path	internally used

**Value**

rendered 'HTML' segments

**Examples**

```
## Not run:
# in your 'index.html' file
<html>
<header>
{{ shidashi::include_view("header.html") }}
</header>
<body>

</body>
<!-- Before closing html tag -->
{{ shidashi::include_view("footer.html") }}
</html>

## End(Not run)
```

---

info_box	<i>Generates 'HTML' info box</i>
----------	----------------------------------

---

**Description**

Generates 'HTML' info box

**Usage**

```
info_box(
  ...,
  icon = "envelope",
  class = "",
  class_icon = "bg-info",
  class_content = "",
  root_path = template_root()
)
```

**Arguments**

...	box content
icon	the box icon; default is "envelope", can be hidden by specifying NULL
class	class of the box container
class_icon	class of the icon
class_content	class of the box body
root_path	see <a href="#">template_root</a>

**Value**

'HTML' tags

**Examples**

```
library(shiny)
library(shidashi)

info_box("Message", icon = "cogs")

info_box(
  icon = "thumbs-up",
  span(class = "info-box-text", "Likes"),
  span(class = "info-box-number", "12,320"),
  class_icon = "bg-red"
)

info_box("No icons", icon = NULL)
```

---

javascript-tunnel

*The 'JavaScript' tunnel*

---

**Description**

The 'JavaScript' tunnel

**Usage**

```
register_session_id(
  session = shiny::getDefaultReactiveDomain(),
  shared_id = NULL,
  shared_inputs = NA
)

register_session_events(session = shiny::getDefaultReactiveDomain())

get_theme(event_data, session = shiny::getDefaultReactiveDomain())

get_jsevent(
  event_data,
  type,
  default = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```



**Arguments**

session	shiny reactive domain
shared_id	the shared id of the session, usually automatically set
shared_inputs	the input names to share to/from other sessions
event_data	a reactive value list returned by register_session_events
type	event type; see 'Details'
default	default value if type is missing

**Details**

The register\_session\_id should be used in the module server function. It registers a shared\_id and a private\_id to the session. The sessions with the same shared\_id can synchronize their inputs, specified by shared\_inputs even on different browser tabs.

register\_session\_events will read the session events from 'JavaScript' and passively update these information. Any the event fired by shidashi.broadcastEvent in 'JavaScript' will be available as reactive value. get\_jsevent provides a convenient way to read these events provided the right event types. get\_theme is a special get\_jsevent that with event type "theme.changed".

Function register\_session\_id and register\_session\_events should be called at the beginning of server functions. They can be called multiple times safely. Function get\_jsevent and get\_theme should be called in reactive contexts (such as [observe](#), [observeEvent](#)).

**Value**

register\_session\_id returns a list of function to control "sharing" inputs with other shiny sessions with the same shared\_id. register\_session\_events returns a reactive value list that reflects the session state. get\_jsevent returns events fired by shidashi.broadcastEvent in 'JavaScript'. get\_theme returns a list of theme, foreground, and background color.

**Examples**

```
# shiny server function

library(shiny)
server <- function(input, output, session){
  sync_tools <- register_session_id(session = session)
  event_data <- register_session_events(session = session)

  # if you want to enable syncing. They are suspended by default
  sync_tools$enable_broadcast()
  sync_tools$enable_sync()

  # get_theme should be called within reactive context
  output$plot <- renderPlot({
    theme <- get_theme(event_data)
    mar(bg = theme$background, fg = theme$foreground)
    plot(1:10)
  })
}
```

```
}

```

---

module_info	<i>Obtain the module information</i>
-------------	--------------------------------------

---

## Description

Obtain the module information

## Usage

```
module_info(root_path = template_root(), settings_file = "modules.yaml")

load_module(
  root_path = template_root(),
  request = list(QUERY_STRING = "/"),
  env = parent.frame()
)
```

## Arguments

root_path	the root path of the website project
settings_file	the settings file containing the module information
request	'HTTP' request string
env	environment to load module variables into

## Details

The module files are stored in `modules/` folder in your project. The folder names are the module id. Within each folder, there should be one `"server.R"`, `R/`, and a `"module-ui.html"`.

The `R/` folder stores R code files that generate variables, which will be available to the other two files. These variables, along with some built-ins, will be used to render `"module-ui.html"`. The built-in functions are

**ns** shiny name-space function; should be used to generate the id for inputs and outputs. This strategy avoids conflict id effectively.

**.module\_id** a variable of the module id

**module\_title** a function that returns the module label

The `"server.R"` has access to all the code in `R/` as well. Therefore it is highly recommended that you write each 'UI' component side-by-side with their corresponding server functions and call these server functions in `"server.R"`.

**Value**

A data frame with the following columns that contain the module information:

`id` module id, folder name

`order` display order in side-bar

`group` group menu name if applicable, otherwise NA

`label` the readable label to be displayed on the side-bar

`icon` icon that will be displayed ahead of label, will be passed to `as_icon`

`badge` badge text that will be displayed following the module label, will be passed to `as_badge`

`url` the relative 'URL' address of the module.

**Examples**

```
library(shiny)
module_info()

# load master module
load_module()

# load specific module
module_data <- load_module(
  request = list(QUERY_STRING = "?module=module_id"))
env <- module_data$environment

if(interactive()){

# get module title
env$module_title()

# generate module-specific shiny id
env$ns("input1")

# generate part of the UI
env$ui()

}
```

---

notification

*The 'Bootstrap' notification*

---

**Description**

The 'Bootstrap' notification

**Usage**

```

show_notification(
  message,
  title = "Notification!",
  subtitle = "",
  type = c("default", "info", "warning", "success", "danger", "white", "dark"),
  close = TRUE,
  position = c("topRight", "topLeft", "bottomRight", "bottomLeft"),
  autohide = TRUE,
  fixed = TRUE,
  delay = 5000,
  icon = NULL,
  collapse = "",
  session = shiny::getDefaultReactiveDomain(),
  class = NULL,
  ...
)

clear_notifications(class = NULL, session = shiny::getDefaultReactiveDomain())

```

**Arguments**

message	notification body content, can be 'HTML' tags
title, subtitle	title and subtitle of the notification
type	type of the notification; can be "default", "info", "warning", "success", "danger", "white", "dark"
close	whether to allow users to close the notification
position	where the notification should be; choices are "topRight", "topLeft", "bottomRight", "bottomLeft"
autohide	whether to automatically hide the notification
fixed	whether the position should be fixed
delay	integer in millisecond to hide the notification if autohide=TRUE
icon	the icon of the title
collapse	if message is a character vector, the collapse string
session	shiny session domain
class	the extra class of the notification, can be used for style purposes, or by clear_notifications to close specific notification types.
...	other options; see <a href="https://adminlte.io/docs/3.1//javascript/toasts.html#options">https://adminlte.io/docs/3.1//javascript/toasts.html#options</a>

**Value**

Both functions should be used in shiny reactive contexts. The messages will be sent to shiny 'JavaScript' interface and nothing will be returned.

## Examples

```
## Not run:

# the examples must run in shiny reactive context

show_notification(
  message = "This validation process has finished. You are welcome to proceed.",
  autohide = FALSE,
  title = "Success!",
  subtitle = "type='success'",
  type = "success"
)

show_notification(
  message = "This notification has title and subtitle",
  autohide = FALSE,
  title = "Hi there!",
  subtitle = "Welcome!",
  icon = "kiwi-bird",
  class = "notification-auto"
)

# only clear notifications with class "notification-auto"
clear_notifications("notification-auto")

## End(Not run)
```

---

progressOutput

*Progress bar in shiny dashboard*

---

## Description

For detailed usage, see demo application by running `render()`.

## Usage

```
progressOutput(
  outputId,
  ...,
  description = "Initializing",
  width = "100%",
  class = "bg-primary",
  value = 0,
  size = c("md", "sm", "xs")
)

renderProgress(expr, env = parent.frame(), quoted = FALSE, outputArgs = list())
```

**Arguments**

outputId	the element id of the progress
...	extra elements on the top of the progress bar
description	descriptive message below the progress bar
width	width of the progress
class	progress class, default is "bg-primary"
value	initial value, ranging from 0 to 100; default is 0
size	size of the progress bar; choices are "md", "sm", "xs"
expr	R expression that should return a named list of value and description
env	where to evaluate expr
quoted	whether expr is quoted
outputArgs	a list of other parameters in progressOutput

**Value**

progressOutput returns 'HTML' tags containing progress bars that can be rendered later via [shiny\\_progress](#) or renderProgress. renderProgress returns shiny render functions internally.

**Examples**

```
library(shiny)
library(shidashi)
progressOutput("sales_report_prog1",
               description = "6 days left!",
               "Add Products to Cart",
               span(class="float-right", "123/150"),
               value = 123/150 * 100)

# server function
server <- function(input, output, session, ...){
  output$sales_report_prog1 <- renderProgress({
    return(list(
      value = 140 / 150 * 100,
      description = "5 days left!"
    ))
  })
}
```

---

register\_global\_reactiveValues  
*Register global reactive list*

---

**Description**

Creates or get reactive value list that is shared within the same shiny session

**Usage**

```
register_global_reactiveValues(  
  name,  
  session = shiny::getDefaultReactiveDomain()  
)
```

**Arguments**

name	character, the key of the list
session	shiny session

**Value**

A shiny [reactiveValues](#) object

---

render *Render a 'shidashi' project*

---

**Description**

Render a 'shidashi' project

**Usage**

```
render(  
  root_path = template_root(),  
  ...,  
  prelaunch = NULL,  
  prelaunch_quoted = FALSE,  
  launch_browser = TRUE,  
  as_job = TRUE,  
  test_mode = getOption("shiny.testmode", FALSE)  
)
```

**Arguments**

root_path	the project path, default is the demo folder from <code>template_root()</code>
...	additional parameters passed to <code>runApp</code> , such as host, port
prelaunch	expression to execute before launching the session; the expression will execute in a brand new session
prelaunch_quoted	whether the expression is quoted; default is false
launch_browser	whether to launch browser; default is TRUE
as_job	whether to run as 'RStudio' jobs; this options is only available when 'RStudio' is available
test_mode	whether to test the project; this options is helpful when you want to debug the project without relaunching shiny applications

**Value**

This functions runs a 'shiny' application, and returns the job id if 'RStudio' is available.

**Examples**

```
template_root()

if(interactive()){
  render()
}
```

---

reset_output	<i>Reset shiny outputs with messages</i>
--------------	--

---

**Description**

Forces outdated output to reset and show a silent message.

**Usage**

```
reset_output(
  outputId,
  message = "This output has been reset",
  session = shiny::getDefaultReactiveDomain()
)
```

**Arguments**

outputId	output ID
message	output message
session	shiny reactive domain



**Value**

No value

---

shiny_progress	<i>Wrapper of shiny progress that can run without shiny</i>
----------------	---

---

**Description**

Wrapper of shiny progress that can run without shiny

**Usage**

```
shiny_progress(  
  title,  
  max = 1,  
  ...,  
  quiet = FALSE,  
  session = shiny::getDefaultReactiveDomain(),  
  shiny_auto_close = FALSE,  
  log = NULL,  
  outputId = NULL  
)
```

**Arguments**

title	the title of the progress
max	max steps of the procedure
...	passed to initialization method of <a href="#">Progress</a>
quiet	whether the progress needs to be quiet
session	shiny session domain
shiny_auto_close	whether to close the progress once function exits
log	alternative log function
outputId	the element id of <a href="#">progressOutput</a> , or NULL to use the default shiny progress

**Value**

a list of functions that controls the progress

## Examples

```
{
  progress <- shiny_progress("Procedure A", max = 10)
  for(i in 1:10){
    progress$inc(sprintf("Step %s", i))
    Sys.sleep(0.1)
  }
  progress$close()
}

if(interactive()){
  library(shiny)

  ui <- fluidPage(
    fluidRow(
      column(12, actionButton("click", "Click me"))
    )
  )

  server <- function(input, output, session) {
    observeEvent(input$click, {
      progress <- shiny_progress("Procedure B", max = 10,
                                shiny_auto_close = TRUE)

      for(i in 1:10){
        progress$inc(sprintf("Step %s", i))
        Sys.sleep(0.1)
      }
    })
  }

  shinyApp(ui, server)
}
```

---

show\_ui\_code

*Used by demo project to show the generating code*

---

## Description

Please write your own version. This function is designed for demo-use only.

## Usage

```
show_ui_code(
  x,
  class = NULL,
  code_only = FALSE,
  as_card = FALSE,
```

```

    card_title = "",
    class_body = "bg-gray-70",
    width.cutoff = 80L,
    indent = 2,
    wrap = TRUE,
    args.newline = TRUE,
    blank = FALSE,
    copy_on_click = TRUE,
    ...
)

```

### Arguments

**x**                    'HTML' tags generated by this package  
**class**                additional 'HTML' class  
**code\_only**            whether to show code only  
**as\_card**              whether to wrap results in `card`  
**card\_title, class\_body**  
                           used by `card` if `as_card=TRUE`  
**width.cutoff, indent, wrap, args.newline, blank, copy\_on\_click, ...**  
                           passed to `html_highlight_code`

### Value

'HTML' tags

### See Also

`html_highlight_code`

---

<code>template_settings</code>	<i>Configure template options that are shared across the sessions</i>
--------------------------------	---

---

### Description

Configure template options that are shared across the sessions

### Usage

```

template_settings

template_settings_set(...)

template_settings_get(name, default = NULL)

template_root()

```

**Arguments**

...	key-value pair to set options
name	character, key of the value
default	default value if the key is missing

**Format**

An object of class `list` of length 3.

**Details**

The settings is designed to store static key-value pairs that are shared across the sessions. The most important key is `"root_path"`, which should be a path pointing to the template folder.

**Value**

`template_settings_get` returns the values represented by the corresponding keys, or the default value if key is missing.

**Examples**

```
# Get current website root path

template_root()
```

---

use\_template

*Download 'shidashi' templates from 'Github'*

---

**Description**

Download 'shidashi' templates from 'Github'

**Usage**

```
use_template(  
  path,  
  user = "dipterix",  
  theme = "AdminLTE3",  
  repo = "shidashi-templates",  
  branch = "main",  
  ...  
)
```

**Arguments**

path	the path to create 'shidashi' project
user	'Github' user name
theme	the theme to download
repo	repository if the name is other than 'shidashi-templates'
branch	branch name if other than 'main' or 'master'
...	ignored

**Details**

To publish a 'shidashi' template, create a 'Github' repository called 'shidashi-templates', or fork the [built-in templates](#). The theme is the sub-folder of the template repository.

An easy way to use a template in your project is through the 'RStudio' project widget. In the 'RStudio' navigation bar, go to "File" menu, click on the "New Project..." button, select the "Create a new project" option, and find the item that creates 'shidashi' templates. Use the widget to set up template directory.

**Value**

the target project path

# Index

## \* datasets

- template\_settings, 35
- accordion, 2, 5, 15, 16
- accordion\_item, 3, 4
- accordion\_operate (accordion), 2
- add-remove-html-class, 5
- add\_class (add-remove-html-class), 5
- adminlte, 6
- adminlte\_sidebar (adminlte), 6
- adminlte\_ui (adminlte), 6
- as\_badge, 7, 10, 13, 27
- as\_icon, 7, 27
- back\_top\_button, 8
- card, 7, 9, 15, 16, 35
- card2, 7, 15, 16
- card2 (card), 9
- card2\_close (card), 9
- card2\_open (card), 9
- card2\_toggle (card), 9
- card\_operate (card), 9
- card\_tabset, 7, 12, 14–16
- card\_tabset\_activate
  - (card\_tabset\_operate), 14
- card\_tabset\_insert
  - (card\_tabset\_operate), 14
- card\_tabset\_operate, 12, 13, 14
- card\_tabset\_remove
  - (card\_tabset\_operate), 14
- card\_tool, 10, 13, 15, 17
- clear\_notifications (notification), 27
- clipboardOutput, 16
- flex\_break (flex\_container), 17
- flex\_container, 17
- flex\_item (flex\_container), 17
- flip (flip\_box), 19
- flip\_box, 16, 19
- format\_text\_r, 20, 22
- get\_construct\_string, 21, 21
- get\_jsevent (javascript-tunnel), 24
- get\_theme (javascript-tunnel), 24
- guess\_body\_class, 22
- html\_highlight\_code, 35
- html\_highlight\_code (format\_text\_r), 20
- icon, 8
- include\_view, 22
- info\_box, 23
- javascript-tunnel, 24
- load\_module (module\_info), 26
- module\_info, 26
- notification, 27
- observe, 25
- observeEvent, 25
- Progress, 33
- progressOutput, 29, 33
- reactiveValues, 31
- register\_global\_reactiveValues, 31
- register\_session\_events
  - (javascript-tunnel), 24
- register\_session\_id
  - (javascript-tunnel), 24
- remove\_class (add-remove-html-class), 5
- render, 31
- renderClipboard (clipboardOutput), 16
- renderProgress (progressOutput), 29
- reset\_output, 32
- runApp, 32
- shiny\_progress, 30, 33

show\_notification(notification), 27  
show\_ui\_code, 34

template\_root, 3, 10, 23  
template\_root(template\_settings), 35  
template\_settings, 6, 35  
template\_settings\_get  
    (template\_settings), 35  
template\_settings\_set  
    (template\_settings), 35  
tidy\_source, 21

use\_template, 5, 36