

# Package ‘shidashi’

May 9, 2026

**Type** Package

**Title** A Shiny Dashboard Template Modular System with Chat Bot Support

**Version** 0.2.0

**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** 7.3.3

**Description** A template dashboard system with AI agent integrated.

Comes with default themes that can be 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, tools, bslib, digest (>= 0.6.27), ellmer (>= 0.4.0), fastmap (>= 1.1.0), formatR (>= 1.11), httr2, S7, shiny (>= 1.7.0), shinychat, yaml, jsonlite, htmlwidgets

**Suggests** cli, coro, logger (>= 0.2.1), processx, promises, later, rstudioapi (>= 0.13), htmltools, httpuv, ggplot2, ggExtra, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** Zhengjia Wang [cph, aut, cre] (ORCID:

[<https://orcid.org/0000-0001-5629-1116>](https://orcid.org/0000-0001-5629-1116)),

ColorlibHQ [cph] (AdminLTE - Bootstrap 4 Admin Dashboard),

Bootstrap contributors [ctb] (Bootstrap library),

Twitter, Inc [cph] (Bootstrap library),

Ivan Sagalaev [ctb, cph] (highlight.js library),  
 Rene Haas [ctb, cph] (OverlayScrollbars library),  
 Zeno Rocha [ctb, cph] (Clipboard.js library)

**Maintainer** Zhengjia Wang <dipterix.wang@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-04-10 17:00:02 UTC

## Contents

accordion	3
accordion_item	5
add-remove-html-class	6
adminlte	6
as_badge	7
as_icon	8
back_top_button	9
bslib_dependency	10
card	10
card_badge	13
card_tabset	15
card_tabset_operate	17
card_tool	18
clipboardOutput	19
drawer	20
fire_event	21
flex_container	22
flip_box	24
format_text_r	25
get_construct_string	26
guess_body_class	27
html_asis	27
html_class	28
include_view	29
info_box	30
init_app	31
mcp_wrapper	31
module_drawer	32
module_info	33
notification	36
open_url	37
progressOutput	38
read_stream_vis	39
register_io	40
register_session	42
render	45
renderStreamViz	46
reset_output	47

shiny\_progress . . . . . 47  
 show\_ui\_code . . . . . 49  
 skill\_wrapper . . . . . 50  
 streamVizOutput . . . . . 51  
 stream\_file\_id . . . . . 51  
 stream\_init . . . . . 52  
 stream\_path . . . . . 52  
 stream\_to\_js . . . . . 53  
 stream\_viz . . . . . 54  
 template\_settings . . . . . 55  
 updateStreamViz . . . . . 56  
 use\_template . . . . . 57

**Index 59**

---

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 <a href="#">template_root</a>

## 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 *Convert characters, shiny icons into 'fontawesome' 4*

---

## 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	<i>'HTML' code to generate small back-to-top button</i>
-----------------	---

---

## Description

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

When `open_drawer = TRUE`, an additional button is rendered that fires a `"button.click"` shidashi-event with `type = "open_drawer"`. Module server code (or `chatbot_server`) can observe this event via `get_event` and call `drawer_open / shiny::renderUI` to fill the drawer with content.

## Usage

```
back_top_button(  
  icon = "chevron-up",  
  title = "Jump to",  
  open_drawer = TRUE,  
  drawer_icon = "ellipsis"  
)
```

## Arguments

<code>icon</code>	the icon for back-to-top button
<code>title</code>	the expanded menu title
<code>open_drawer</code>	logical; whether to include a drawer-toggle button. Defaults to TRUE if a <code>.shidashi-drawer</code> element will be present in the page (e.g. from <code>module_drawer()</code> ).
<code>drawer_icon</code>	the icon for the drawer-toggle button; defaults to "ellipsis" (three dots). Use e.g. "robot" for AI-agent modules.

## Value

'HTML' tags

## Examples

```
back_top_button()  
back_top_button("rocket")  
back_top_button("rocket", drawer_icon = "robot")
```

---

bslib_dependency	<i>Get Bootstrap 5 dependencies via <b>bslib</b></i>
------------------	--

---

**Description**

Returns Bootstrap 5 HTML dependencies provided by **bslib**. Intended to be called from HTML templates so that `headContent()` renders Bootstrap 5 CSS and JavaScript.

**Usage**

```
bslib_dependency(...)
```

**Arguments**

... additional arguments passed to `bslib::bs_theme`

**Value**

An [tagList](#) containing Bootstrap 5 dependencies

---

card	<i>Card-like 'HTML' element</i>
------	---------------------------------

---

**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\_badge

*Create a badge widget located at card header*


---

### Description

Create a badge widget located at card header

**Usage**

```

card_badge(text = NULL, class = NULL, ...)

card_recalculate_badge(
  text = "Recalculate needed",
  class = NULL,
  event_name = "run_analysis",
  ...
)

enable_recalculate_badge(text = "Recalculate needed", ...)

disable_recalculate_badge(text = "Up-to-date", ...)

set_card_badge(
  id = NULL,
  class = NULL,
  text = NULL,
  add_class = NULL,
  remove_class = NULL,
  session = shiny::getDefaultReactiveDomain()
)

```

**Arguments**

<code>text</code>	inner text content of the badge
<code>class</code>	additional 'HTML' class of the badge; for <code>set_card_badge</code> , this is the class selector of the badge that is to be changed
<code>...</code>	additional 'HTML' tag attributes
<code>event_name</code>	name of the event to trigger
<code>id</code>	the badge 'HTML' ID to be changed, will be enclosed with session namespace <code>session\$ns(id)</code> automatically.
<code>add_class, remove_class</code>	add or remove class
<code>session</code>	shiny session

**Examples**

```

library(shidashi)

# UI: a Bootstrap badge with green background
card_badge("Ready", class = "bg-green shidashi-output-status")

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

  shiny::observe({

```

```

# ... check if the inputs have changed

set_card_badge(
  class = "shidashi-output-status",
  text = "Refresh needed",
  add_class = "bg-warning",
  remove_class = "bg-success disabled"
)

})

}

```

---

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	<i>Generates outputs that can be written to clipboards with one click</i>
-----------------	---

---

**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**

<code>outputId</code>	the output id
<code>message</code>	tool tip to show when mouse hovers on the element
<code>clip_text</code>	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 <code>clip_text</code>
env	environment to evaluate <code>expr</code>
quoted	whether <code>expr</code> is quoted
outputArgs	used to replace default arguments of <code>clipboardOutput</code>

### Value

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

### Examples

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

---

drawer

*Open, close, or toggle the drawer panel*

---

### Description

Send messages to the client to open, close, or toggle the off-canvas drawer panel on the right side of the dashboard.

### Usage

```
drawer_open(session = shiny::getDefaultReactiveDomain())  
drawer_close(session = shiny::getDefaultReactiveDomain())  
drawer_toggle(session = shiny::getDefaultReactiveDomain())
```

### Arguments

session      shiny session

### Value

No value is returned (called for side effect).

**Examples**

```
server <- function(input, output, session) {
  # Open the drawer
  drawer_open()

  # Close the drawer
  drawer_close()

  # Toggle the drawer
  drawer_toggle()
}
```

---

fire_event	<i>Fire or read a session event</i>
------------	-------------------------------------

---

**Description**

fire\_event sets a reactive event value on the current session. When global = TRUE, the event is propagated to all sessions that share the same shared\_id (i.e. other browser tabs for the same user).

get\_event reads the current value of an event key from the session registry.

get\_theme is a convenience wrapper around get\_event("theme.changed") that returns the current dashboard theme.

All three functions must be called inside a Shiny server context.

**Usage**

```
fire_event(
  key,
  value,
  session = shiny::getDefaultReactiveDomain(),
  global = FALSE
)
```

```
get_event(key, session = shiny::getDefaultReactiveDomain(), default = NULL)
```

```
get_theme(session = shiny::getDefaultReactiveDomain())
```

**Arguments**

key	a single character string identifying the event type
value	the event payload (any R object)
session	a Shiny session (defaults to the current reactive domain)
global	logical; if TRUE, the event is broadcast to all sessions sharing the same shared_id
default	value to return when the event has not been fired yet (used by get_event only)

**Details**

get\_theme and get\_event auto-register the session if needed and must be called inside a reactive context ([observe](#), [observeEvent](#), [reactive](#), render functions).

**Value**

fire\_event returns NULL invisibly. get\_event returns the last value fired for key, or default if none. get\_theme returns a named list with three character elements:

**theme** Either "light" or "dark".

**foreground** Hex color string for text / foreground elements.

**background** Hex color string for the page background.

Before the browser fires its first theme event, the light-theme fallback `list(theme = "light", background = "#FFFFFF", foreground = "#000000")` is returned.

**Examples**

```
library(shiny)
server <- function(input, output, session) {
  # fire an event
  shidashi::fire_event("my_event", list(a = 1), session = session)

  # read it back
  observe({
    evt <- shidashi::get_event("my_event", session = session)
    print(evt)
  })

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

**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	<i>Get re-formatted R expressions in characters</i>
---------------	---

---

### 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

---

html_asis	<i>Escape HTML strings</i>
-----------	----------------------------

---

**Description**

Escape HTML strings so that they will be displayed 'as-is' in websites.

**Usage**

```
html_asis(s, space = TRUE)
```

**Arguments**

s                    characters  
space                whether to also escape white space, default is true.

**Value**

An R string

## Examples

```
html_asis("<a><----> <b>")
```

---

html_class	<i>Combine, add, or remove 'HTML' classes</i>
------------	---

---

## Description

Combine 'HTML' classes to produce nice, clean 'HTML' class string via `combine_html_class`, or to remove a class via `remove_html_class`

## Usage

```
combine_html_class(...)  
remove_html_class(target, class)
```

## Arguments

<code>...</code>	one or more characters, classes to combine; duplicated classes will be removed
<code>target</code>	characters, class list
<code>class</code>	one or more characters, classes to be removed from <code>target</code>

## Value

A character string of new 'HTML' class

## Examples

```
# Combine classes "a b c d e"  
combine_html_class("a", "b a", c("c", " d", "b"), list("e ", "a"))  
  
# Remove class  
remove_html_class("a b c e", c("b", "c "))
```

---

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)
```

---

init_app	<i>Initialize a shidashi application</i>
----------	--

---

**Description**

Creates a fresh environment holding all per-application global stores (session registry, per-module input registries, etc.). This function is designed to be called from a 'global.R' file generated by `render()`, not for user's call.

**Usage**

```
init_app(env = parent.frame())
```

**Arguments**

env environment where the application is initialized into.

**Value**

Nothing.

**Examples**

```
init_app(env = new.env())
```

---

mcp_wrapper	<i>Wrap an MCP Tool Generator Function</i>
-------------	--

---

**Description**

Creates a wrapper around a generator function to ensure it returns valid Model Context Protocol (MCP) tool definitions. The wrapper validates input and filters output to contain only 'e11mer::ToolDef' objects.

**Usage**

```
mcp_wrapper(generator)
```

**Arguments**

generator A function that accepts a 'session' parameter and returns either a single tool object or a list/vector of such objects; see [tool](#).

**Details**

The wrapper performs the following validations: - Ensures 'generator' is a function - Checks that 'generator' accepts a 'session' parameter

The returned function automatically handles both single tool definitions and lists of tools, providing a consistent interface for MCP tool registration.

**Value**

A wrapped function with class 'shidashi\_mcp\_wrapper' that: - Accepts a 'session' parameter - Calls the generator function with the session - Normalizes the output to a list - Filters to keep only valid tool objects - Returns a list of tool objects (possibly empty)

**Examples**

```
# Define a generator function that returns tool definitions
my_tool_generator <- function(session) {
  # Define MCP tools using ellmer package

  tool_rnorm <- tool(
    function(n, mean = 0, sd = 1) {
      shiny::updateNumericInput(session, "rnorm", value = rnorm)
    },
    description = "Draw numbers from a random normal distribution",
    arguments = list(
      n = type_integer("The number of observations. Must be positive"),
      mean = type_number("The mean value of the distribution."),
      sd = type_number("The standard deviation of the distribution.")
    )
  )

  # or `list(tool_rnorm)`
  tool_rnorm
}

# Wrap the generator
wrapped_generator <- mcp_wrapper(my_tool_generator)
```

---

 module\_drawer

*Drawer shell for module templates*


---

**Description**

Emits a minimal .shidashi-drawer container with a `uiOutput` placeholder inside, plus the drawer overlay. The drawer starts empty; module server code fills it dynamically via `shiny::renderUI`.

Typical usage in a 'module-ui.html' template:

```
{{ shidashi::module_drawer() }}
```

Then in the module server:

```
output$shidashi_drawer <- shiny::renderUI({
  shiny::tagList(
    shiny::h5("My settings"),
    shiny::p("Custom drawer content here.")
  )
})
```

The `ns()` function from the module's template evaluation environment is used automatically so that the output ID is properly scoped to the module namespace.

### Usage

```
module_drawer(output_id = "shidashi_drawer")
```

### Arguments

`output_id` character; the output ID for the `uiOutput` placeholder inside the drawer. Defaults to "shidashi\_drawer".

### Value

A `shiny::tagList` containing the drawer div and its overlay.

---

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

---

### Description

`current_module` returns the information of the currently running module. It looks up the `.module_id` variable in the calling environment (set automatically when a module is loaded), then retrieves the corresponding row from the module table.

`active_module` returns a *reactive* value with information about the module that is currently visible in the `iframe` tab (or the standalone module if no `iframe` manager is present). Unlike `current_module` which is static and always returns the module whose server code is running, `active_module` dynamically tracks which module the user is looking at from any context.

`switch_module` programmatically switches the active module in the dashboard UI. It sends a `shidashi.switch_module` message to the JavaScript front-end. When called from a module running inside an `iframe`, the handler automatically forwards the request to the parent window via `postMessage` so that the sidebar highlight, tab bar, and `iframe` all update correctly.

**Usage**

```

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

current_module(
  session = shiny::getDefaultReactiveDomain(),
  root_path = template_root()
)

active_module(
  session = shiny::getDefaultReactiveDomain(),
  root_path = template_root()
)

switch_module(module_id, session = shiny::getDefaultReactiveDomain())

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

```

**Arguments**

<code>root_path</code>	the root path of the website project
<code>settings_file</code>	the settings file containing the module information
<code>session</code>	shiny reactive domain; used to extract the module id from the URL query string when <code>.module_id</code> is not found.
<code>module_id</code>	character string; the target module identifier (must match an entry in 'modules.yaml').
<code>request</code>	'HTTP' request string
<code>env</code>	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"`.

`active_module` works by reading the '@shidashi\_active\_module@' Shiny input that is set by the JavaScript front-end whenever a module tab is activated. Because it accesses `session$rootScope().input`, the return value is reactive: when called inside an observe or reactive context it will re-fire whenever the user switches modules.

If the input has not been set yet (e.g. before any module is opened), the function falls back to `current_module()`.

## 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.

`current_module`: a named list with `id`, `group`, `label`, `icon`, `badge`, and `url` of the current module, or NULL if no module is active.

`active_module`: a named list with `id`, `group`, `label`, `icon`, `badge`, and `url` of the currently active (visible) module, or NULL if no module is active.

## 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	<i>The 'Bootstrap' notification</i>
--------------	-------------------------------------

---

## 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 <https://adminlte.io/docs/3.1//javascript/toasts.html#options>

### 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)
```

---

open_url	<i>Open a URL in a new browser tab</i>
----------	--

---

### Description

Sends a message to the client to open the specified URL in a new browser window/tab.

### Usage

```
open_url(url, target = "_blank", session = shiny::getDefaultReactiveDomain())
```

**Arguments**

url	character string, the URL to open
target	the window.open target; default is "_blank" (new tab)
session	shiny session

**Value**

No value is returned (called for side effect).

---

progressOutput	<i>Progress bar in shiny dashboard</i>
----------------	--

---

**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-end", "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!"
    ))
  })
}
```

---

read_stream_vis	<i>Read a shidashi stream binary file</i>
-----------------	---

---

**Description**

Reads the binary envelope written by [stream\\_to\\_js](#) and returns the header and decoded body as a list.

**Usage**

```
read_stream_vis(path)
```

**Arguments**

path                    Character scalar. Absolute path to a .bin file produced by [stream\\_to\\_js](#).

**Value**

A list with components:

header   Named list parsed from the JSON header (contains data\_type, signature, timestamp, and any extra fields).

data   Decoded body: a raw vector for "raw", an R object for "json", or a numeric/integer vector for "int32", "float32", "float64".

**See Also**

[stream\\_to\\_js](#), [stream\\_path](#)

---

 register\_io

*Register Shiny Inputs and Outputs for MCP Access*


---

**Description**

Register shiny inputs and outputs for MCP (Model Context Protocol) agent access.

`register_input()` wraps a shiny input constructor to register metadata. It evaluates `expr` and returns the UI element.

`register_output()` is a server-side function that registers a render function call (e.g. `renderPlot({...})`), assigns it to `session$output`, registers the MCP output spec, and sets up download-widget handlers. The UI overlay icons are injected entirely by JS.

**Usage**

```
register_input(
  expr,
  inputId,
  update,
  description = "",
  writable = TRUE,
  quoted = FALSE,
  env = parent.frame()
)
```

```
register_output(
  expr,
  outputId,
  description = "",
  quoted = FALSE,
  env = parent.frame(),
  ...,
  output_opts = list(),
  download_function = NULL,
  download_type = c("image", "htmlwidget", "threeBrain", "no-download", "data",
    "stream_viz"),
  extension = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

**Arguments**

`expr` For `register_input`: a call expression that creates a shiny input widget. For `register_output`: a render function call such as `renderPlot({...})`.

inputId	character string. The shiny input ID (without the module namespace prefix).
update	character string. The fully qualified update function, e.g. "shiny::updateTextInput". Field mappings such as "shiny::updateSelectInput(value=selected)" override the default argument names passed to the update function.
description	character string. A human-readable description of the input or output purpose, exposed to LLM agents via MCP tools.
writable	logical (default TRUE). Whether the MCP update tool is allowed to change this input.
quoted	logical (default FALSE). If TRUE, expr is treated as already quoted; otherwise it is captured with <code>substitute()</code> .
env	the environment in which to evaluate expr.
outputId	character string. The shiny output ID (without the module namespace prefix).
...	reserved for future use.
output_opts	a named list of extra options for the output (e.g. width, height defaults).
download_function	a custom download handler function. When <code>download_type = "data"</code> , this function receives the file path and writes the download content.
download_type	character string. One of "image", "threeBrain", "data", or "no-download".
extension	character vector of allowed file extension for download, or NULL.
session	the shiny session object. For <code>register_output</code> , defaults to <code>shiny::getDefaultReactiveDomain()</code> .

### Value

`register_input` returns the evaluated UI element. `register_output` is called for its side effects (assigning the render function and registering widgets) and returns NULL invisibly.

### See Also

[init\\_app](#), [mcp\\_wrapper](#)

### Examples

```
## Not run:
# inside a shidashi module UI function:
ns <- shiny::NS("demo")

register_input(
  expr = shiny::sliderInput(
    inputId = ns("threshold"),
    label = "Threshold",
    min = 0, max = 1, value = 0.5
  ),
  inputId = "threshold",
  update = "shiny::updateSliderInput",
  description = "Filter threshold for the plot"
)
```

```
# inside a shidashi module server function:
register_output(
  expr = renderPlot({ plot(iris) }),
  outputId = "my_plot",
  description = "Scatter plot of iris data",
  download_type = "image"
)

## End(Not run)
```

---

 register\_session

*Shiny session registration and cross-tab synchronization*


---

## Description

Shiny session registration and cross-tab synchronization

## Usage

```
register_session(session)

unregister_session(session)

enable_input_broadcast(
  session = shiny::getDefaultReactiveDomain(),
  once = FALSE
)

disable_input_broadcast(session = shiny::getDefaultReactiveDomain())

enable_input_sync(session = shiny::getDefaultReactiveDomain(), once = FALSE)

disable_input_sync(session = shiny::getDefaultReactiveDomain())

get_handler(name, session = shiny::getDefaultReactiveDomain())

set_handler(name, handler, session = shiny::getDefaultReactiveDomain())
```

## Arguments

session	A Shiny session object or session proxy (created by <code>moduleServer</code> ). Most functions in this family default to <code>shiny::getDefaultReactiveDomain()</code> so callers inside module server functions do not need to pass it explicitly. <code>register_session</code> and <code>unregister_session</code> require an explicit value.
once	Logical; if TRUE, resume the suspended observer for a single run via <code>\$run()</code> instead of permanently re-activating it with <code>\$resume()</code> . Useful for one-shot snapshots without installing a persistent observer.

name	A single character string identifying a named slot in the session's handler list (used by <code>get_handler</code> and <code>set_handler</code> ). Three names are reserved for internal use and will trigger an error if passed to <code>set_handler</code> : "event_handler", "broadcast_handler", and "input_sync_handler".
handler	A Shiny Observer object created by <code>shiny::observe()</code> , or NULL to clear the named slot (used by <code>set_handler</code> ). Passing any other object type raises an error.

## Details

### Session registration:

`register_session()` — Call once at the top of every Shiny module server function. It is idempotent: calling it a second time on the same session safely refreshes the session object and URL in the registry entry without re-creating observers.

Internally it creates an entry in the application-global session registry (initialized by `init_app`), resolves a `shared_id` token shared across browser tabs from the `?shared_id=...` URL query string (or generates a random 26-character string when absent), sets up the per-session reactive event bus, and — for named module sessions — sends a `shidashi.register_module_token` custom message to bind the module namespace to its session token on the JavaScript side.

`unregister_session()` — Removes the session entry from the registry and destroys all attached observers. This is called automatically when the session ends via the `onSessionEnded` hook installed by `register_session()`. Direct calls are only needed for explicit early cleanup (e.g. in tests).

### Session-scoped handlers:

Each registered session maintains a named slot list for Shiny Observer objects called *handlers*. This provides a lightweight system for attaching module-level life-cycle hooks that are tied to the session's lifetime.

**User-defined handlers** — `get_handler()` / `set_handler()`

`set_handler(name, handler)` installs `handler` under `name`, first suspending and destroying any Observer already stored there. Pass `handler = NULL` to clear the slot. Returns `FALSE` invisibly if the session is already closed.

`get_handler(name)` retrieves the stored Observer (or NULL). It auto-registers the session if not yet registered and returns NULL gracefully if the session is closed.

Three handler names are reserved for internal shidashi infrastructure and will raise an error if passed to `set_handler`: "event\_handler", "broadcast\_handler", and "input\_sync\_handler".

### Built-in cross-tab sync handlers

shidashi installs two opt-in Observer slots in every registered session (both start *suspended*):

**Input broadcast** (`enable_input_broadcast()` / `disable_input_broadcast()`) Continuously monitors the session's input values and, whenever they change, pushes a snapshot to the client via `shidashi.cache_session_input`. Other browser tabs sharing the same `shared_id` can read this cached snapshot to restore or compare input state.

**Input sync** (`enable_input_sync()` / `disable_input_sync()`) Listens for serialized input maps broadcast by *other* sessions sharing the same `shared_id` via the root-session `@shidashi@input`. Values differing from the local input are written into the session's private `inputs` reactiveValues. Messages from the same session are ignored to prevent feedback loops.

Both observers run at priority `-100000` (after all ordinary reactive computations have settled). Use `once = TRUE` to trigger a single cycle without permanently resuming the observer. The `disable_*` variants suspend the observer and are silent no-ops when the session has already ended.

### Session life-cycle:

```

init_app()                # global.R, once per app start
  |
  v
register_session(session) # top of each module server()
  |
  v
... reactive code ...
  get_handler() / set_handler() # attach user-defined session observers
  enable_input_broadcast()      # optional: push inputs to browser cache
  enable_input_sync()          # optional: receive peer-tab inputs
  |
  v
session ends -> unregister_session() # runs automatically

```

### Value

`register_session` returns the session token (`session$token`) invisibly; it is idempotent and safe to call multiple times for the same session.

`unregister_session` returns `NULL` invisibly; it is idempotent.

`enable_input_broadcast`, `disable_input_broadcast`, `enable_input_sync`, and `disable_input_sync` all return invisibly. They are silent no-ops when the session is already closed.

`get_handler` returns the named `Observer` object stored under `name`, or `NULL` if the slot is empty or the session is closed.

`set_handler` returns `TRUE` invisibly when it successfully installs the handler, or `FALSE` invisibly when the session is already closed.

### Examples

```

library(shiny)

# --- Basic usage in a module server ---
server <- function(input, output, session) {
  shidashi::register_session(session)

  # opt-in to cross-tab input broadcast (suspended by default)
  shidashi::enable_input_broadcast(session)

  # opt-in to receive inputs from peer tabs
  shidashi::enable_input_sync(session)

  # get_theme must be called within a reactive context
  output$plot <- renderPlot({
    theme <- shidashi::get_theme()
    par(bg = theme$background, fg = theme$foreground)
  })
}

```

```

    plot(1:10)
  })
}

# --- Named handler: attach a reusable session-scoped observer ---
server2 <- function(input, output, session) {
  shidashi::register_session(session)

  cleanup <- shiny::observe({
    # ... module-level teardown logic ...
    shidashi::set_handler("my_cleanup", NULL, session)
  }, suspended = TRUE, domain = session)

  shidashi::set_handler("my_cleanup", cleanup, session)

  # retrieve and resume the observer elsewhere in the same session
  h <- shidashi::get_handler("my_cleanup", session)
  if (!is.null(h)) h$resume()
}

```

---

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

<code>root_path</code>	the project path, default is the demo folder from <code>template_root()</code>
<code>...</code>	additional parameters passed to <code>runApp</code> , such as <code>host</code> , <code>port</code>
<code>prelaunch</code>	expression to execute before launching the session; the expression will execute in a brand new session
<code>prelaunch_quoted</code>	whether the expression is quoted; default is <code>false</code>

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()
}
```

---

renderStreamViz	<i>Render a streaming widget</i>
-----------------	----------------------------------

---

**Description**

Server-side render function for [streamVizOutput](#). The expression should evaluate to a [stream\\_viz](#) object.

**Usage**

```
renderStreamViz(expr, env = parent.frame(), quoted = FALSE)
```

**Arguments**

expr	An R expression that returns a <a href="#">stream_viz</a> widget.
env	Environment in which to evaluate expr.
quoted	Logical. Whether expr is already quoted.

**Value**

A server-side render function for use with Shiny.

**See Also**

[streamVizOutput](#), [updateStreamViz](#)

---

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

<code>x</code>	'HTML' tags generated by this package
<code>class</code>	additional 'HTML' class
<code>code_only</code>	whether to show code only
<code>as_card</code>	whether to wrap results in <code>card</code>
<code>card_title</code> , <code>class_body</code>	used by <code>card</code> if <code>as_card=TRUE</code>
<code>width.cutoff</code> , <code>indent</code> , <code>wrap</code> , <code>args.newline</code> , <code>blank</code> , <code>copy_on_click</code> , ...	passed to <code>html_highlight_code</code>

### Value

'HTML' tags

### See Also

`html_highlight_code`

---

 skill\_wrapper

*Wrap a Skill Directory as an MCP Tool Generator*


---

### Description

Creates a closure that produces an `ellmer::tool` dispatching on an action enumerator:

`readme` Returns the full `SKILL.md` instructions. Must be called first to unlock other actions.

`reference` Returns content from a reference file in the skill directory (gated behind `readme`).

`script` Executes a script in the `scripts/` subdirectory via `processx::run()` (gated behind `readme`).

### Usage

```
skill_wrapper(skill_path)
```

### Arguments

`skill_path` Path to the skill directory containing `SKILL.md`. Can be absolute or relative to the project root.

### Details

The returned tool enforces a soft gate: calling `reference` or `script` before `readme` is allowed, but if the call errors the message is augmented with a condensed summary (~200 tokens) instructing the AI to read the full instructions first. This minimizes token waste (the summary is only sent on failure).

The gate state is per-instance: each call to the wrapper produces a closure with an independent `readme_unlocked` flag.

### Value

A function with class `c("shidashi_skill_wrapper", "function")` that returns an `ellmer::ToolDef` object.

### Examples

```
skill_dir <- system.file(
  "builtin-templates/bslib-bare/agents/skills/greet",
  package = "shidashi"
)
wrapper <- skill_wrapper(skill_dir)
tool_def <- wrapper()
cat(tool_def(action = "readme"))
```

---

streamVizOutput	<i>Output placeholder for a streaming visualization widget</i>
-----------------	--

---

**Description**

Use in a Shiny UI to reserve a slot for a multi-channel signal viewer that is driven by binary stream data.

**Usage**

```
streamVizOutput(outputId, width = "100%", height = "400px")
```

**Arguments**

outputId	Character scalar. Output ID that matches the corresponding <a href="#">renderStreamViz</a> call.
width, height	CSS width and height of the widget container.

**Value**

An HTML output element suitable for inclusion in a Shiny UI.

**See Also**

[renderStreamViz](#), [updateStreamViz](#)

---

stream_file_id	<i>Build the token-qualified stream file identifier</i>
----------------	---

---

**Description**

Combines `session$token` with `id` to produce the string used both by [stream\\_path](#) (as the filename stem) and by the browser as the URL path component under `stream/`.

**Usage**

```
stream_file_id(id, session = shiny::getDefaultReactiveDomain(), token = NULL)
```

**Arguments**

id	Character scalar. The base stream identifier (no path separators).
session	Shiny session object. Defaults to the active reactive domain.
token	Character scalar or NULL. When NULL (default) the session's own token is used. Override with a parent session token so that a standalone viewer (child session) can address the parent's binary stream file.

**Value**

Character scalar: "{token}\_{id}".

**See Also**

[stream\\_path](#), [stream\\_viz](#), [updateStreamViz](#)

---

stream\_init

*Initialize the shidashi stream directory for a Shiny session*

---

**Description**

Call once in the server function to set up the directory that [stream\\_path](#) and [stream\\_to\\_js](#) will write to. When running inside a shidashi template the directory is automatically resolved from the template root; when running in plain Shiny a temporary directory is created and registered with [addResourcePath](#) so that the browser can fetch `stream/{token}_{id}.bin`. An optional cleanup hook is registered to remove this session's files when the session ends.

**Usage**

```
stream_init(session = shiny::getDefaultReactiveDomain())
```

**Arguments**

`session` Shiny session object. Defaults to the currently active reactive domain.

**Value**

Invisibly returns the absolute path to the stream directory.

**See Also**

[stream\\_path](#), [stream\\_to\\_js](#)

---

stream\_path

*Get the absolute path for a shidashi stream file*

---

**Description**

Returns the full path to the binary file that will be served at `stream/{token}_{id}.bin`. The filename embeds the session token so that simultaneous Shiny sessions never overwrite each other's data. Call [stream\\_init](#) at least once per session before relying on this function.

**Usage**

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

**Arguments**

id	Character scalar. Stream identifier. Must be a valid file-name component (no path separators).
session	Shiny session object. Defaults to the currently active reactive domain.

**Value**

Invisible character scalar: absolute path to the .bin file.

**See Also**

[stream\\_init](#), [stream\\_to\\_js](#)

---

stream\_to\_js

*Write data to a **shidashi** stream binary file*

---

**Description**

Serializes data together with a JSON header into the binary envelope expected by the JavaScript window.shidashi.fetchStreamData(id) method.

**Usage**

```
stream_to_js(
  abspath,
  data,
  type = c("raw", "json", "int32", "float32", "float64"),
  ...
)
```

**Arguments**

abspath	Character scalar. Absolute path to the target .bin file. Use <a href="#">stream_path(id)</a> to obtain a path under the app's www/stream/ directory.
data	The data to serialize. See Details for how each type interprets this argument.
type	Character scalar. One of "raw", "json", "int32", "float32", or "float64".
...	Additional named scalar values to embed in the JSON header and expose to the JavaScript caller via result.header.

**Details****Wire format**

[endianFlag: 1 byte] [headerLen: uint32 LE] [header: UTF-8 JSON] [body]

endianFlag is always 0x01 (little-endian). header is a JSON object containing at minimum data\_type plus any extra fields passed via . . . .

### Body encoding by type

"raw" Passed through verbatim; data must be a raw vector or will be coerced via as.raw().

"json" data is serialized with jsonlite::toJSON(auto\_unbox = TRUE).

"int32" data is coerced to integer and written as 4-byte little-endian signed integers.

"float32" data is coerced to double and written as 4-byte little-endian IEEE 754 single-precision floats.

"float64" data is coerced to double and written as 8-byte little-endian IEEE 754 double-precision floats.

### Value

Invisibly returns abspath.

### See Also

[stream\\_path](#)

---

stream_viz	<i>Create a streaming widget</i>
------------	----------------------------------

---

### Description

Creates a multi-channel signal viewer widget that fetches its data from the binary stream file produced by [stream\\_to\\_js](#).

### Usage

```
stream_viz(
  outputId,
  session = shiny::getDefaultReactiveDomain(),
  stream_id = NULL,
  width = NULL,
  height = NULL,
  refresh_rate = 33,
  show_controls = TRUE,
  streaming = FALSE,
  token = NULL
)
```

### Arguments

outputId	Character scalar. The output ID matching the corresponding <a href="#">streamVizOutput</a> call. Used to compute stream_id automatically when stream_id is NULL.
session	Shiny session object. Defaults to the active reactive domain. Together with outputId determines the token-qualified stream file identifier.
stream_id	Character scalar or NULL. When NULL (default) the stream identifier is computed as <code>stream_file_id(outputId, session, token = token)</code> . Provide an explicit value to override for non-Shiny or advanced use.
width, height	Widget dimensions (passed to <a href="#">createWidget</a> ).
refresh_rate	Numeric scalar (milliseconds). Polling interval used when the viewer is in <i>active streaming</i> mode (play button pressed). Lower values give smoother animation but increase CPU/network load. Default 33 (approximately 30 Hz).
show_controls	Logical. Whether to display the hover toolbar (play/pause, zoom, reset, export). Default TRUE.
streaming	Logical. Whether to start active streaming immediately when the widget is rendered. Default FALSE; set to TRUE to begin polling for new data automatically. Can also be toggled later via <a href="#">updateStreamViz</a> .
token	Character scalar or NULL. When NULL (default) the session's own token is used. Override with a parent session token when running inside a standalone viewer (pop-out window) so that the child session fetches the parent's binary stream file.

### Details

The outputId together with session is used to auto-compute the token-qualified stream file identifier via [stream\\_file\\_id](#). You can override this by passing an explicit stream\_id.

Use [updateStreamViz](#) for in-place updates without recreating the widget.

### Value

An `htmlwidgets` object.

### See Also

[streamVizOutput](#), [renderStreamViz](#), [updateStreamViz](#), [stream\\_file\\_id](#)

---

template\_settings

*Configure template options that are shared across the sessions*

---

### 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()
```

---

`updateStreamViz`*Trigger an in-place update of a streaming widget*

---

**Description**

Sends a custom Shiny message that causes the browser to re-fetch the binary stream file and re-render the signal viewer without tearing down and recreating the widget. Call this after writing new data with `stream_to_js`. The `outputId` is used both to locate the widget as the stream file identifier — it must match the `id` passed to `stream_path`.

**Usage**

```
updateStreamViz(
  outputId,
  streaming = NULL,
  token = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

**Arguments**

outputId	Character scalar. The output ID passed to <a href="#">streamVizOutput</a> (and to <a href="#">stream_path</a> ).
streaming	Logical or NULL. TRUE starts active streaming (the widget polls for new data via <code>requestAnimationFrame</code> ); FALSE pauses it; NULL (default) leaves the current streaming state unchanged.
token	Character scalar or NULL. When NULL (default) the session's own token is used. Override with a parent session token when running inside a standalone viewer (pop-out window) so that the child session fetches the parent's binary stream file.
session	Shiny session object. Defaults to the active reactive domain.

**Value**

Invisibly NULL.

**See Also**

[streamVizOutput](#), [stream\\_to\\_js](#), [stream\\_file\\_id](#)

---

use\_template

*Download 'shidashi' templates from 'Github'*

---

**Description**

Download 'shidashi' templates from 'Github'

**Usage**

```
use_template(
  path,
  user = "dipterix",
  theme = "bslib",
  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, 55
  
- accordion, 3, 5, 18, 19
- accordion\_item, 4, 5
- accordion\_operate (accordion), 3
- active\_module (module\_info), 33
- add-remove-html-class, 6
- add\_class (add-remove-html-class), 6
- addResourcePath, 52
- adminlte, 6
- adminlte\_sidebar (adminlte), 6
- adminlte\_ui (adminlte), 6
- as\_badge, 7, 11, 15, 35
- as\_icon, 8, 35
  
- back\_top\_button, 9
- bslib\_dependency, 10
  
- card, 7, 10, 18, 19, 49
- card2, 7, 18, 19
- card2 (card), 10
- card2\_close (card), 10
- card2\_open (card), 10
- card2\_toggle (card), 10
- card\_badge, 13
- card\_operate (card), 10
- card\_recalculate\_badge (card\_badge), 13
- card\_tabset, 7, 15, 17–19
- card\_tabset\_activate  
    (card\_tabset\_operate), 17
- card\_tabset\_insert  
    (card\_tabset\_operate), 17
- card\_tabset\_operate, 15, 16, 17
- card\_tabset\_remove  
    (card\_tabset\_operate), 17
- card\_tool, 11, 15, 18, 20
- chatbot\_server, 9
- clear\_notifications (notification), 36
- clipboardOutput, 19
  
- combine\_html\_class (html\_class), 28
- createWidget, 55
- current\_module (module\_info), 33
  
- disable\_input\_broadcast  
    (register\_session), 42
- disable\_input\_sync (register\_session), 42
- disable\_recalculate\_badge (card\_badge), 13
- drawer, 20
- drawer\_close (drawer), 20
- drawer\_open, 9
- drawer\_open (drawer), 20
- drawer\_toggle (drawer), 20
  
- enable\_input\_broadcast  
    (register\_session), 42
- enable\_input\_sync (register\_session), 42
- enable\_recalculate\_badge (card\_badge), 13
  
- fire\_event, 21
- flex\_break (flex\_container), 22
- flex\_container, 22
- flex\_item (flex\_container), 22
- flip (flip\_box), 24
- flip\_box, 19, 24
- format\_text\_r, 25, 27
  
- get\_construct\_string, 26, 26
- get\_event, 9
- get\_event (fire\_event), 21
- get\_handler (register\_session), 42
- get\_theme (fire\_event), 21
- guess\_body\_class, 27
  
- html\_asis, 27
- html\_class, 28
- html\_highlight\_code, 49
- html\_highlight\_code (format\_text\_r), 25

icon, 8  
include\_view, 29  
info\_box, 30  
init\_app, 31, 41, 43  
  
load\_module (module\_info), 33  
  
mcp\_wrapper, 31, 41  
module\_drawer, 9, 32  
module\_info, 33  
moduleServer, 42  
  
notification, 36  
  
observe, 22  
observeEvent, 22  
open\_url, 37  
  
Progress, 48  
progressOutput, 38, 48  
  
reactive, 22  
read\_stream\_vis, 39  
register\_input (register\_io), 40  
register\_io, 40  
register\_output (register\_io), 40  
register\_session, 42  
remove\_class (add-remove-html-class), 6  
remove\_html\_class (html\_class), 28  
render, 31, 45  
renderClipboard (clipboardOutput), 19  
renderProgress (progressOutput), 38  
renderStreamViz, 46, 51, 55  
reset\_output, 47  
runApp, 45  
  
set\_card\_badge (card\_badge), 13  
set\_handler (register\_session), 42  
shiny\_progress, 39, 47  
show\_notification (notification), 36  
show\_ui\_code, 49  
skill\_wrapper, 50  
stream\_file\_id, 51, 55, 57  
stream\_init, 52, 52, 53  
stream\_path, 40, 51, 52, 52, 53, 54, 56, 57  
stream\_to\_js, 39, 40, 52, 53, 53, 54, 56, 57  
stream\_viz, 46, 52, 54  
streamVizOutput, 46, 51, 55, 57  
switch\_module (module\_info), 33  
  
tagList, 10  
template\_root, 4, 12, 30  
template\_root (template\_settings), 55  
template\_settings, 7, 55  
template\_settings\_get  
    (template\_settings), 55  
template\_settings\_set  
    (template\_settings), 55  
tidy\_source, 26  
tool, 31  
  
uiOutput, 32  
unregister\_session (register\_session),  
    42  
updateStreamViz, 46, 51, 52, 55, 56  
use\_template, 6, 57