Package 'shidashi'

November 10, 2026

Type Package

Title A Shiny Dashboard Template Modular System with Chat Bot Support

Version 0.3.1

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

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 (>= 0.5.0),
fastmap (>= 1.1.0), formatR (>= 1.11), httr2, S7, shiny (>=
1.7.0), shinychat, yaml, jsonlite, htmltools, htmlwidgets,
later, promises

Suggests cli, coro, logger (>= 0.2.1), processx, rstudioapi (>= 0.13),
httpuv, ggplot2, ggExtra, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Zhengjia Wang [cph, aut, cre] (ORCID:
<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 2026-11-10 17:10:02 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
colormapSelectInput . . . 20
drawer . . . 21
fire_event . . . 22
flex_container . . . 24
flip_box . . . 25
format_text_r . . . 26
get_construct_string . . . 27
guess_body_class . . . 28
html_asis . . . 29
html_class . . . 29
include_view . . . 30
info_box . . . 31
init_app . . . 32
mcp_call_active . . . 32
mcp_click . . . 33
mcp_wrapper . . . 34
module_drawer . . . 35
module_info . . . 36
notification . . . 39
objectListInput . . . 40
open_url . . . 42
progressOutput . . . 43
read_stream_vis . . . 44
register_input_state . . . 45

register_io 46
register_session 48
render 51
renderPlot2 53
renderStreamViz 53
render_hidden_output 54
reset_output 55
save_launcher 55
shiny_progress 57
show_ui_code 58
skill_wrapper 60
streamVizOutput 61
stream_file_id 61
stream_init 62
stream_path 62
stream_to_js 63
stream_viz 64
template_settings 65
updateStreamViz 66
use_template 67

Index **69**

`accordion` *Generates an 'accordion' tab-set*

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 accordion_item
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 accordion_item
root_path	see template_root
itemId	accordion_item id
item_title	accordion_item 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` *'Accordion' items*

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 `accordion` id
`itemId`	the item id
`style_header, style_body`	'CSS' style of item header and body
`root_path`	see `template_root`

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 *Generates badge icons*

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 `icon`
`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.

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

icon	the icon for back-to-top button
title	the expanded menu title
open_drawer	logical; whether to include a drawer-toggle button. Defaults to TRUE if a .shidashi-drawer element will be present in the page (e.g.\ from module_drawer()).
drawer_icon	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 *Get Bootstrap 5 dependencies via* **bslib**

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 *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 `as_badge` or `card_tool`
`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 template_root
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 corder. ",
      "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

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

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 `card_tabset`
`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` *Generates small icon widgets*

Description

The icons cane 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 card_tool
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")
```

---

colormapSelectInput *A shiny color picker*

Description

Displays color right by the color names for better user experience; implemented using vanilla shiny selector

Usage

```
colormapSelectInput(
  inputId,
  label,
  colormaps,
  selected = NULL,
  continuous = FALSE
)
```

Arguments

inputId, label, selected	passed to shiny selectInput; a selected that is not one of the names of the color map list (including NULL, character(0) and "") selects the first color map
colormaps	A named list with name being the color-map names and values being a character vector of key colors
continuous	whether the color map is continuous; default is false

Details

The selector always holds one of the color maps while there are any: the selected item cannot be removed with the keyboard, and an update that empties the value (for example `updateSelectizeInput` with `selected = character(0)`) puts the previous value back, or the first color map. Only an empty `colormaps` list yields the value `""`, which a server should treat as "no choice".

Value

A shiny selector

Examples

```
colormaps <- list(
  Dark2 = c("#1b9e77", "#d95f02", "#7570b3", "#e7298a",
            "#66a61e", "#e6ab02", "#a6761d", "#666666"),
  Paired = c("#a6cee3", "#1f78b4", "#b2df8a", "#33a02c",
             "#fb9a99", "#e31a1c", "#fdbf6f", "#ff7f00"),
  Pastel1 = c("#fbb4ae", "#b3cde3", "#ccebc5", "#decbe4",
              "#fed9a6", "#ffffcc", "#e5d8bd", "#fddaec"),
  Set1 = c("#e41a1c", "#377eb8", "#4daf4a", "#984ea3",
           "#ff7f00", "#ffff33", "#a65628", "#f781bf")
)

color_picker <- colormapSelectInput(
  inputId = "colormap", label = "Pick a colormap",
  colormaps = colormaps, continuous = FALSE
)

if (interactive()) {
  shiny::shinyApp(
    ui = color_picker,
    server = function(input, output, server) {
      shiny::observe({
        print(colormaps[[input$colormap]])
      })
    }
  )
}
```

`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 *Fire or read a session event*

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)
  })
}
```

`flex_container` *Generate 'HTML' tags with 'flex' layout*

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

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 *Escape HTML strings*

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 *Combine, add, or remove 'HTML' classes*

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

...	one or more characters, classes to combine; duplicated classes will be removed
target	characters, class list
class	one or more characters, classes to be removed from `target`

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` *Template function to include 'snippets' in the view folder*

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 *Generates 'HTML' info box*

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 template_root

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` *Initialize a shidashi application*

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_call_active` *Whether an* `MCP` *tool call is running*

Description

Returns `TRUE` while a tool runs for an `MCP` request, and `FALSE` otherwise, for example when the in-dashboard chat calls the same tool. While it is `TRUE`, the environment variable `SHIDASHI_USING_MCP` is set to `"TRUE"`, so code that cannot call this function, such as skill scripts running in their own process, can check the variable instead.

Usage

```
mcp_call_active(v)
```

Arguments

v	optional logical; when given, turns the state on (TRUE) or off (anything else), and sets or removes SHIDASHI_USING_MCP to match. App code normally only reads the state.

Details

The dashboard's MCP handler turns the state on before each tool call and off when the call returns, also when the tool fails. A tool that returns a promise returns before the promise settles, so code that runs once it settles sees FALSE.

Value

TRUE or FALSE: the state after any change.

Examples

```
# Outside of an MCP tool call
mcp_call_active()
Sys.getenv("SHIDASHI_USING_MCP")

# In a tool: print less when an agent calls it over MCP
summarize_values <- function(x) {
  if (!mcp_call_active()) {
    message("Summarizing ", length(x), " values")
  }
  summary(x)
}
summarize_values(1:10)
```

---

mcp_click *Click a page element for an agent*

Description

Asks the browser to click the first element that matches selector, for a tool call made by an agent (for example over MCP). The browser does not click an element inside [mcp-agent-disabled="true"]: such controls are for people only (for example a button that overwrites files), and agents should ask the user to click them. People's own clicks are not affected. Use this function in tools instead of sending the "shidashi.click" message.

Usage

```
mcp_click(
  selector,
  session = shiny::getDefaultReactiveDomain(),
  timeout = getOption("shidashi.click_timeout", 10)
)
```

Arguments

selector	CSS selector of the element to click
session	shiny session; for modules, the module session
timeout	seconds to wait for the browser's answer

Value

A promise resolved with a list: matched (an element matched the selector), clicked, refused (the element is disabled for agents), and note (an invalid selector, for example). The promise is rejected when the browser does not answer in time.

Examples

```
# In a tool function (shiny session required):
# promises::then(mcp_click("#my_button", session), function(res) {
#   if (isTRUE(res$refused)) stop("The button is for people only")
#   "Clicked"
# })
```

---

mcp_wrapper *Wrap an* MCP *Tool Generator Function*

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 'ellmer::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.

A tool can call mcp_call_active to tell whether an agent called it over MCP or the in-dashboard chat did, for example to print less for agents.

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 *Obtain the module information*

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 shows a module in the dashboard, as clicking it in the sidebar does: the module's open tab comes to the front, or the module opens in a new tab. It sends a shidashi.switch_module message to the JavaScript front-end. When called from a module running inside an iframe, the page asks the dashboard in its parent frame to switch, 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(),
  auto_new = TRUE
)

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
session	shiny reactive domain; used to extract the module id from the URL query string when .module_id is not found.
module_id	character string; the target module identifier (must match an entry in 'modules.yaml').
auto_new	logical; whether to open the module in a new tab when it is not open; FALSE only switches to an open tab.
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".

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 *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 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)
```

---

objectListInput *A ordered list of objects as a shiny input*

Description

Displays the selected objects as a vertical list; entries can be dragged to re-order and removed with the X button on the right. The value is a character vector of the entry keys, in the order shown on screen. Implemented using vanilla shiny selectizeInput, hence no additional 'JavaScript' library is needed.

Usage

```
objectListInput(
  inputId,
  label = NULL,
  choices = NULL,
  selected = NULL,
  width = NULL,
  placeholder = "(No object selected)",
  sortable = TRUE,
  removable = TRUE,
  allow_readd = TRUE
)

updateObjectListInput(
  session = shiny::getDefaultReactiveDomain(),
  inputId,
  label = NULL,
  choices = NULL,
  selected = choices
)
```

Arguments

inputId, label, width	passed to selectizeInput
choices	a named list or character vector of the entries; the names are the text displayed in each row and the values are the entry keys returned by the input; the order of choices is the order of the rows
selected	keys of the entries to show; default is all of choices
placeholder	text to display when the list is empty
sortable	whether the entries can be dragged to re-order; requires **'jQuery'** shipped with **shiny**, and is silently disabled when unavailable
removable	whether each entry has a button to remove itself
allow_readd	whether the entries can be selected again from a drop-down menu; when FALSE the list only shows what the server puts in it, and removing an entry is final until it is added again from the server; when TRUE a removed entry stays available in the menu and re-selecting it appends it to the end of the list; default is TRUE
session	shiny session

Value

objectListInput returns a shiny input; the input value is a character vector of the entry keys in display order. updateObjectListInput is called for its side effect.

Examples

```
ui <- shiny::basicPage(
  objectListInput(
    inputId = "objects", label = "Selected objects",
    choices = character(),
    selected = character()
  ),
  shiny::actionButton("btn", "Add")
)

if (interactive()) {
  shiny::shinyApp(
    ui = shiny::fluidPage(ui, shiny::verbatimTextOutput("selected")),
    server = function(input, output, session) {
      output$selected <- shiny::renderPrint({ input$objects })

      shiny::bindEvent(
        shiny::observe({
          shidashi::updateObjectListInput(
            inputId = "objects",
            choices = c(
              "Electrode 1 [LA1]" = "elec_1",
              "Streamlines [n=12]" = "streamlines_1",
              "Overlay aparc [discrete]" = "volume_1"
            )
          )
        }),
        input$btn
      )
    }
  )
}
```

`open_url` *Open a URL in a new browser tab*

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 *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-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 *Read a shidashi stream binary file*

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_input_state *Report session state to agents*

Description

Registers a value that the MCP tool shiny_input_info reports next to the inputs, under the key "@state". Use it for state that changes what an agent should ask, for example whether the data loader is open (a closed loader usually means the data are loaded and the user wants to change analysis inputs only).

Usage

```
register_input_state(
  name,
  getter,
  description = "",
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

name	character string, the name of the state
getter	a function without arguments that returns the current value; it is called in shiny::isolate() each time shiny_input_info runs, and an error is reported as the state's error instead of failing the tool call
description	character string: what the value means, for agents
session	the shiny session (the module's session)

Value

NULL, invisibly

Examples

```
## Not run:
# in a module server function
register_input_state(
  "loader_opened",
  function() isTRUE(loader_is_open()),
  description = "TRUE while the data loader is open"
)

## End(Not run)
```

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, with tooltip as its hover tip (the HTML title attribute). as_tooltip() gives the default hover tip: the first sentence of description.

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 = "",
  tooltip = as_tooltip(description),
  writable = TRUE,
  hint = "no_hint",
  quoted = FALSE,
  env = parent.frame()
)

input_hint_classes

as_tooltip(description)

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.
tooltip	character string shown when the mouse hovers over the input; defaults to the first sentence of description (see as_tooltip()). Set to NULL for no hover tip. The hover tip is added only when the outer tag of the UI element is an input (a button, a, input, select, or textarea tag, or a tag with class shiny-input-container) without a title; other elements, such as cards, are returned unchanged.
writable	logical (default TRUE). Whether the MCP update tool is allowed to change this input.
hint	character string, one of input_hint_classes: what an agent does with the input before acting, reported by the shiny_input_info tool. The loader_* hints apply before loading data and the analysis_* hints before running the module's main action: *_mandatory inputs are asked from the user, *_optional inputs may keep their current or default value, and *_forbidden inputs are left alone. The default "no_hint" gives no guidance.
quoted	logical (default FALSE). If TRUE, expr is treated as already quoted; otherwise it is captured with substitute().
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 download_type = "data", this function receives the file path and writes the download content.
download_type	character string. One of "image", "threeBrain", "data", or "no-download". A "threeBrain" download saves the **threeBrain** viewer as it is shown: its current control panel values and camera replace the ones the render expression sets.
extension	character vector of allowed file extension for download, or NULL.
session	the shiny session object. For register_output, defaults to shiny::getDefaultReactiveDomain().

Value

register_input returns the evaluated UI element, with tooltip as the title of an input (see tooltip). as_tooltip returns the first sentence of description as a character string ("" when it is blank). 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

```
# The default hover tip: the first sentence of the description
as_tooltip("Plot threshold, e.g. 0.5. Agents set it before running.")

## 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 moduleServer). Most functions in this family default to shiny::getDefaultReactiveDomain() so callers inside module server functions do not need to pass it explicitly. register_session and unregister_session require an explicit value.
once	Logical; if TRUE, resume the suspended observer for a single run via $run() instead of permanently re-activating it with $resume(). 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 get_handler and set_handler). Three names are reserved for internal use and will trigger an error if passed to set_handler: "event_handler", "broadcast_handler", and "input_sync_handler".
handler	A Shiny Observer object created by shiny::observe(), or NULL to clear the named slot (used by set_handler). 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),
  server_name = "shidashi"
)
```

Arguments

root_path	the project path, default is the demo folder from `template_root()`
...	additional parameters passed to runApp, 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
server_name	server name to show in the `MCP` route

Value

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

Examples

```
template_root()

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

renderPlot2 *Shiny render plot function with automated theme switcher*

Description

A wrapper around renderPlot, but with themes automatically set via par; only supports base plots. For **ggplot2**, please manually call get_theme to get the theme.

Usage

```
renderPlot2(expr, ..., env = parent.frame(), quoted = FALSE)
```

Arguments

expr, env, quoted, ...
: passed to renderPlot

Value

See renderPlot

Examples

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

  output$plot <- renderPlot2({
    plot(rnorm(100))
  })

}
```

renderStreamViz *Render a streaming widget*

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 stream_viz 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

---

render_hidden_output *Render an Output Even When It Is Hidden*

Description

By default shiny does not render outputs that are hidden, for example inside an inactive tab or a collapsed card (suspendWhenHidden = TRUE). render_hidden_output() turns this off for one output, so the output renders at the next flush even when hidden. With once = TRUE, the option is restored after that flush.

Usage

```
render_hidden_output(
  outputId,
  session = shiny::getDefaultReactiveDomain(),
  once = TRUE
)
```

Arguments

outputId	character string. The shiny output ID (without the module namespace prefix).
session	the shiny session object.
once	logical (default TRUE). Whether to restore the option after the next flush; when FALSE, the output keeps rendering while hidden until the returned function is called.

Value

A function (invisible) that restores the option; calling it more than once has no further effect. Nothing is changed when the output already renders while hidden.

Examples

```
## Not run:
# inside a shidashi module server function:
register_output(renderPlot({ plot(1:10) }), outputId = "my_plot")

# render `my_plot` once, even while its tab is not shown
render_hidden_output("my_plot")

## End(Not run)
```

`reset_output` *Reset shiny outputs with messages*

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

`save_launcher` *Save and start 'shidashi' apps for AI agents*

Description

`save_launcher` saves how to start a 'shidashi' app, so that an AI agent connected through the `MCP` proxy can list it and, after asking the user, start it and open one of its modules in the browser. Nothing is started automatically: the agent only launches a saved app when the user picks it.

`run_launcher` starts a saved app. The `MCP` proxy calls it when the agent launches an app; it can also be called from R.

Usage

```
save_launcher(
  id,
  root_path,
  host = "127.0.0.1",
  port = NA,
  modules = NULL,
  description = "",
  prelaunch = NULL,
  copy_app = FALSE,
  ...
)

run_launcher(id, ...)
```

Arguments

id	a short name for the app, made of letters, digits, "-", and "_". The agent and the user refer to the app by this name.
root_path	the app directory; it must contain 'modules.yaml'
host	the host to listen on
port	the port to listen on; NA picks a free port at launch
modules	the module identifiers the agent may open; default is the modules that have AI agents enabled (an 'agents.yaml' file)
description	a short description shown to the agent
prelaunch	optional R code, as a character string, to run before the app starts
copy_app	whether to start a copy of the app, kept in the cache folder, instead of the app directory itself
...	for save_launcher, named metadata about the app (character, numeric, or logical values), shown to the agent but not used to start the app; for run_launcher, extra arguments passed to render, which override the defaults (launch_browser = FALSE, as_job = FALSE)

Details

All launchers are kept in one file, 'launchers.json', in the 'shidashi' cache folder (tools::R_user_dir("shidashi", "cache")). Saving again with the same id replaces that launcher. Saving a launcher also installs or updates the MCP proxy script.

With copy_app = TRUE, the app folder is copied to 'saved_apps/<id>/' in the cache folder (without '.git' and 'node_modules'), and the launcher starts the copy. The folder is cleared before each copy. Without copy_app, a copy left from an earlier save of the same id is removed.

Value

save_launcher returns the saved launcher, invisibly. run_launcher runs the app; see render.

Examples

```
# This example saves into a temporary folder; by default launchers are
# saved in the user's cache folder
old_opt <- options(shidashi.cache_dir = tempfile())

root <- system.file("builtin-templates", "bslib-bare", package = "shidashi")
save_launcher(
  "demo-app", root,
  description = "Demo dashboard shipped with shidashi",
  lab = "example"
)

if (interactive()) {
  run_launcher("demo-app", launch_browser = TRUE)
}

options(old_opt)
```

`shiny_progress` *Wrapper of shiny progress that can run without shiny*

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 `Progress`
`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 progressOutput, 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

skill_wrapper *Wrap a Skill Directory as* MCP *Tool Generators*

Description

Creates a closure that produces two `ellmer::tool` objects for a skill:

`skill_load__<name>` Reads the skill. Its `action` argument is `readme` (the full `SKILL.md` instructions, the default) or `reference` (content from a reference file in the skill directory). It never changes anything.

`skill_run__<name>` Executes a script in the `scripts/` subdirectory via `processx::run()`. Only created when the skill has scripts.

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 two tools share a soft gate: reading a reference or running a script before the `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 pair of tools with an independent `readme_unlocked` flag.

Scripts inherit the app's environment variables, plus the ones the caller passes in `envs`. When an agent runs a script over `MCP`, `SHIDASHI_USING_MCP` is `"TRUE"`; see `mcp_call_active`.

Value

A function with class `c("shidashi_skill_wrapper", "function")` that returns a list with elements `load` (an `ellmer::ToolDef`) and `run` (an `ellmer::ToolDef`, or `NULL` when the skill has no scripts).

Examples

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

streamVizOutput *Output placeholder for a streaming visualization widget*

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 renderStreamViz 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 *Build the token-qualified stream file identifier*

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 stream_path(id) 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 *Create a streaming widget*

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 streamVizOutput 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 stream_file_id(outputId, session, token = token). Provide an explicit value to override for non-Shiny or advanced use.
width, height	Widget dimensions (passed to createWidget).
refresh_rate	Numeric scalar (milliseconds). Polling interval used when the viewer is in *active streaming* 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 updateStreamViz.
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

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 streamVizOutput (and to stream_path).
streaming	Logical or NULL. TRUE starts active streaming (the widget polls for new data via requestAnimationFrame); 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

accordion, 3, *5*, *18, 19*
accordion_item, *4*, 5
accordion_operate *(*accordion*)*, 3
active_module *(*module_info*)*, 36
add-remove-html-class, 6
add_class *(*add-remove-html-class*)*, 6
addResourcePath, *62*
adminlte, 6
adminlte_sidebar *(*adminlte*)*, 6
adminlte_ui *(*adminlte*)*, 6
as_badge, 7, *11*, *15*, *38*
as_icon, 8, *38*
as_tooltip *(*register_io*)*, 46

back_top_button, 9
bslib_dependency, 10

card, *7*, 10, *18, 19*, *59*
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*)*, 39
clipboardOutput, 19
colormapSelectInput, 20
combine_html_class *(*html_class*)*, 29
createWidget, *65*
current_module *(*module_info*)*, 36

disable_input_broadcast
        *(*register_session*)*, 48
disable_input_sync *(*register_session*)*,
        48
disable_recalculate_badge *(*card_badge*)*,
        13
drawer, 21
drawer_close *(*drawer*)*, 21
drawer_open, *9*
drawer_open *(*drawer*)*, 21
drawer_toggle *(*drawer*)*, 21

enable_input_broadcast
        *(*register_session*)*, 48
enable_input_sync *(*register_session*)*, 48
enable_recalculate_badge *(*card_badge*)*,
        13

fire_event, 22
flex_break *(*flex_container*)*, 24
flex_container, 24
flex_item *(*flex_container*)*, 24
flip *(*flip_box*)*, 25
flip_box, *19*, 25
format_text_r, 26, *28*

get_construct_string, *27*, 27
get_event, *9*
get_event *(*fire_event*)*, 22
get_handler *(*register_session*)*, 48
get_theme, *53*
get_theme *(*fire_event*)*, 22
guess_body_class, 28

html_asis, 29
html_class, 29
html_highlight_code, *59*
html_highlight_code *(*format_text_r*)*, 26

icon, *8*
include_view, 30
info_box, 31
init_app, 32, *48, 49*
input_hint_classes *(*register_io*)*, 46

load_module *(*module_info*)*, 36

mcp_call_active, 32, *34*, *60*
mcp_click, 33
mcp_wrapper, 34, *48*
module_drawer, *9*, 35
module_info, 36
moduleServer, *49*

notification, 39

objectListInput, 40
observe, *23*
observeEvent, *23*
open_url, 42

par, *53*
Progress, *57*
progressOutput, 43, *58*

reactive, *23*
read_stream_vis, 44
register_input *(*register_io*)*, 46
register_input_state, 45
register_io, 46
register_output *(*register_io*)*, 46
register_session, 48
remove_class *(*add-remove-html-class*)*, 6
remove_html_class *(*html_class*)*, 29
render, *32*, 51, *56*
render_hidden_output, 54
renderClipboard *(*clipboardOutput*)*, 19
renderPlot, *53*
renderPlot2, 53
renderProgress *(*progressOutput*)*, 43
renderStreamViz, 53, *61*, *65*
reset_output, 55
run_launcher *(*save_launcher*)*, 55
runApp, *52*

save_launcher, 55
selectInput, *20*
selectizeInput, *40, 41*
set_card_badge *(*card_badge*)*, 13
set_handler *(*register_session*)*, 48
shiny_progress, *44*, 57
show_notification *(*notification*)*, 39
show_ui_code, 58
skill_wrapper, 60
stream_file_id, 61, *65*, *67*
stream_init, *62*, 62, *63*
stream_path, *45*, *61, 62*, 62, *63, 64*, *66, 67*
stream_to_js, *44, 45*, *62, 63*, 63, *64*, *66, 67*
stream_viz, *53, 54*, *62*, 64
streamVizOutput, *53, 54*, 61, *65*, *67*
switch_module *(*module_info*)*, 36

tagList, *10*
template_root, *4*, *12*, *31*
template_root *(*template_settings*)*, 65
template_settings, *7*, 65
template_settings_get
*(*template_settings*)*, 65
template_settings_set
*(*template_settings*)*, 65
tidy_source, *27*
tool, *34*

uiOutput, *35*
unregister_session *(*register_session*)*, 48
updateObjectListInput
*(*objectListInput*)*, 40
updateSelectizeInput, *21*
updateStreamViz, *54*, *61, 62*, *65*, 66
use_template, *6*, 67