

Package: shinypal (via r-universe)

October 8, 2026

Title Tools to Make Modular and Interactive R Workflow Builders

Version 0.0.0.9000

Depends R (>= 4.1.0)

Imports shiny, shinymeta, DT, bslib (>= 0.11.0), rlang, sortable, htmltools, clipr, purrr, khroma, grDevices, shinycssloaders, fs, rmarkdown, callr, xfun, zip

Suggests knitr, testthat (>= 3.0.0)

Description Provides the building blocks for modular, interactive shiny applications that turn point-and-click pipelines into reproducible R and R Markdown code. Using shinymeta, shinypal assembles code modules into a draggable workflow, shows the equivalent code as the user builds it, and lets them export the whole pipeline as a runnable script or rendered report.

License GPL (>= 3)

URL <https://github.com/willgearty/shinypal>,
<http://williamgearty.com/shinypal/>

BugReports <https://github.com/willgearty/shinypal/issues>

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Config/Needs/website pkgdown

VignetteBuilder knitr

Config/pak/sysreqs cmake make libuv1-dev libx11-dev zlib1g-dev

Repository <https://willgearty.r-universe.dev>

Date/Publication 2026-07-07 19:52:15 UTC

RemoteUrl <https://github.com/willgearty/shinypal>

RemoteRef HEAD

RemoteSha b1cdc3f101a340c94b9c16219d56d74c679dc04c

Contents

accordion_panel_remove_button	2
add_shinypal_data_step	3
add_shinypal_plot_step	5
add_shinypal_step	6
clip_observe	8
column_select_observe	8
df_modal_button	9
df_modal_observe	10
df_select_observe	11
file_observe	11
get_chunk	12
get_colors	13
get_int_data	14
get_int_dfs	14
is_shinylive	15
next_step_index	16
select_column_input	16
select_dataset_input	17
set_int_data	18
shinypal_setup	18
shinypal_ui	20
step_varname	21
var_name_observe	21
varname_input	22
verbatimTextOutput_copy	23
workflow_has_errors	24
Index	25

accordion_panel_remove_button
<i>Accordion panel that includes a remove button</i>

Description

A `bslib::accordion_panel()` pre-wired with a "Remove this step" button and the `data-rank-id` attribute shinypal’s sortable workflow needs. Use it as the panel returned by a step’s `fun_workflow`.

Usage

```
accordion_panel_remove_button(ind, ...)
```

Arguments

- `ind` The index of the step.
- `...` Additional arguments passed to `bslib::accordion_panel()`.

Value

A `bslib::accordion_panel()` tag with value = "step_<ind>" and a matching data-rank-id attribute.

See Also

Other step UI: `df_modal_button()`, `select_column_input()`, `select_dataset_input()`, `varname_input()`, `verbatimTextOutput_copy()`

Examples

```
accordion_panel_remove_button(1, "My step")
```

```
add_shinypal_data_step
```

Register a complete data-producing step

Description

Convenience wrapper around the boilerplate shared by every step that produces an intermediate dataset: it stores the data reactive, wires the data-preview modal, registers the step via `add_shinypal_step()`, and (optionally) keeps the dataset dropdown and column selectors in sync. A module only supplies the data reactive and its UI/report functions.

Usage

```
add_shinypal_data_step(
  ind,
  data,
  fun_workflow,
  fun_report,
  libs = character(0),
  code_guard = NULL,
  ec_subs = NULL,
  select_dataset = FALSE,
  column_ids = character(0),
  rename = TRUE
)
```

Arguments

<code>ind</code>	The index of the step.
<code>data</code>	A <code>shinymeta::metaReactive2()</code> object produced by the step. It should use <code>varname = step_varname(ind)</code> so its generated variable name matches the stored name.
<code>fun_workflow</code>	A function that generates the UI elements for the workflow.

fun_report	A function that generates the UI elements for the report.
libs	A character vector of R packages required for this step.
code_guard	An optional zero-argument function evaluated inside the code output before <code>get_chunk()</code> . Use it to surface step-specific <code>validate()</code> messages; plain <code>req()</code> s inside data already propagate through <code>get_chunk()</code> , so simple steps can leave this NULL.
ec_subs	An optional list of length 2, where the first element is a <code>metaReactive</code> object and the second element is a callback function. This is used to substitute expansion contexts in the code chain.
select_dataset	Whether the step consumes an upstream dataset (and thus needs a <code>df_select_observe()</code> to keep its dataset dropdown current).
column_ids	A character vector of column-selector input ids to keep in sync via <code>column_select_observe()</code> (one per <code>select_column_input()</code>).
rename	Whether to render a <code>varname_input()</code> text field in the step's panel (and install <code>var_name_observe()</code>) so the user can give this step's dataset a custom name, used as its label in later selectors and its variable name in the generated script. Defaults to TRUE.

Value

Called for its side effects and returns NULL invisibly. Stores the step's data reactive under its `step_varname()` id (e.g. `data_1`), wires its code output and data-preview modal, registers the step via `add_shinypal_step()`, and (when requested) installs the dataset/column selector observers and, when `rename = TRUE`, the dataset-rename field and its observer.

See Also

Other workflow steps: `add_shinypal_plot_step()`, `add_shinypal_step()`, `next_step_index()`, `step_varname()`

Examples

```
## Not run:
# inside a module's server.R
ind <- next_step_index()
occs <- shinymeta::metaReactive2(varname = step_varname(ind), {
  shinymeta::metaExpr(head(mtcars, input[[paste0("n_", ind)]])
})
add_shinypal_data_step(
  ind, data = occs,
  fun_workflow = function(ind) accordion_panel_remove_button(ind, "Subset"),
  fun_report = function(ind) verbatimTextOutput_copy(ind)
)

## End(Not run)
```

add_shinypal_plot_step

Register a complete plot-producing step

Description

Convenience wrapper around the boilerplate shared by every step that renders a plot: it assigns the rendered plot to its output slot, renders the step's generated code, registers the step via [add_shinypal_step\(\)](#), and (optionally) keeps the dataset dropdown and column selectors in sync. A module only supplies the plot render and its UI/report functions.

Unlike [add_shinypal_data_step\(\)](#), a plot step does not store an intermediate dataset or show a data-preview modal; its result is a figure, not a selectable data.frame.

Usage

```
add_shinypal_plot_step(
  ind,
  plot,
  fun_workflow,
  fun_report,
  libs = character(),
  code_guard = NULL,
  output_prefix = "plot_",
  select_dataset = TRUE,
  column_ids = character()
)
```

Arguments

ind	The index of the step.
plot	A shinymeta::metaRender2() object that renders the plot.
fun_workflow	A function that generates the UI elements for the workflow.
fun_report	A function that generates the UI elements for the report.
libs	A character vector of R packages required for this step.
code_guard	An optional zero-argument function evaluated inside the code output before get_chunk() . Use it to surface step-specific validate() messages.
output_prefix	The prefix for the plot's output slot, combined with ind to form the id (default "plot_", giving plot_<ind>). Must match the shiny::plotOutput() id used in fun_report (e.g. "map_").
select_dataset	Whether the step consumes an upstream dataset (and thus needs a df_select_observe() to keep its dataset dropdown current).
column_ids	A character vector of column-selector input ids to keep in sync via column_select_observe() (one per select_column_input()).

Value

Called for its side effects and returns NULL invisibly. Assigns the rendered plot to `output[[<output_prefix><ind>]]`, renders the step's generated code, registers the step via `add_shinypal_step()`, and (when requested) installs the dataset/column selector observers.

See Also

Other workflow steps: `add_shinypal_data_step()`, `add_shinypal_step()`, `next_step_index()`, `step_varname()`

Examples

```
## Not run:
# inside a module's server.R
ind <- next_step_index()
p <- shinymeta::metaRender2(shiny::renderPlot, {
  df <- get_int_data(input[[paste0("dataset_", ind)]])()
  shinymeta::metaExpr(plot(df))
})
add_shinypal_plot_step(
  ind, plot = p,
  fun_workflow = function(ind) accordion_panel_remove_button(ind, "Plot"),
  fun_report = function(ind) shiny::plotOutput(paste0("plot_", ind))
)

## End(Not run)
```

add_shinypal_step

Add a step to the report, workflow, and the code chain

Description

Registers a single step with shinypal's reactive state. It inserts the step's panel into the workflow accordion, appends the step's block to the report, adds the step's quoted code to the code chain used to assemble the reproducible script, and records any packages the step needs. When `ec_subs` is supplied, it also registers an expansion-context substitution so the downloadable script can swap in alternate code (for example, fully-qualified calls).

Most data-producing steps should instead use `add_shinypal_data_step()`, which wraps this together with the data storage, code output, and selector boilerplate. Call `add_shinypal_step()` directly for steps that don't fit that pattern.

Usage

```
add_shinypal_step(
  ind,
  fun_workflow,
  fun_report,
  code_chain_list,
```

```

    libs = character(0),
    ec_subs = NULL
  )

```

Arguments

<code>ind</code>	The index of the step.
<code>fun_workflow</code>	A function that generates the UI elements for the workflow.
<code>fun_report</code>	A function that generates the UI elements for the report.
<code>code_chain_list</code>	A list of quoted code bits that will be added to <code>code_chain()</code> .
<code>libs</code>	A character vector of R packages required for this step.
<code>ec_subs</code>	An optional list of length 2, where the first element is a <code>metaReactive</code> object and the second element is a callback function. This is used to substitute expansion contexts in the code chain.

Value

Called for its side effects and returns `NULL` invisibly. The step is registered with `shinypal`'s reactive state (its panel is inserted into the workflow accordion, its block appended to the report, its quoted code added to the code chain, and its packages recorded), and a remove-button observer is installed for it.

See Also

Other workflow steps: [add_shinypal_data_step\(\)](#), [add_shinypal_plot_step\(\)](#), [next_step_index\(\)](#), [step_varname\(\)](#)

Examples

```

## Not run:
# inside a module's server.R, which shinypal_setup() sources with local = TRUE
ind <- next_step_index()
add_shinypal_step(
  ind,
  fun_workflow = function(ind) accordion_panel_remove_button(ind, "Head"),
  fun_report = function(ind) verbatimTextOutput_copy(ind),
  code_chain_list = list(quote(head(mtcars))),
  libs = "utils"
)

## End(Not run)

```

clip_observe	<i>Add an observer to a copy button</i>
--------------	---

Description

Generates an event observer that watches for when a copy button is clicked. Upon clicking, the `code_expr` is evaluated and copied to the clipboard using `clipr::write_clip()`.

Usage

```
clip_observe(ind, code_expr)
```

Arguments

<code>ind</code>	The index of the step.
<code>code_expr</code>	An expression (generated by <code>rlang::expr()</code> that, when evaluated, returns content to be copied (such as the output from <code>shinymeta::expandChain()</code>).

Value

Called for its side effects; invisibly returns the observer.

See Also

`verbatimTextOutput_copy()`, whose copy button it wires.

Other step observers: `column_select_observe()`, `df_modal_observe()`, `df_select_observe()`, `file_observe()`, `var_name_observe()`

Examples

```
## Not run:
clip_observe(ind, rlang::expr(get_chunk(ind)))

## End(Not run)
```

column_select_observe	<i>Keep a varSelectInput of data.frame column names up-to-date</i>
-----------------------	--

Description

Installs an observer that keeps a `shiny::varSelectInput()` of column names in sync with the dataset currently chosen in the step's `dataset_<ind>` dropdown. Pair with `select_dataset_input()` and `select_column_input()`.

Usage

```
column_select_observe(ind, inputId)
```

Arguments

`ind` The index of the step.

`inputId` The id of the `shiny::varSelectInput()` object.

Value

Called for its side effects; invisibly returns the observer.

See Also

`select_column_input()`, the selector this observer updates.

Other step observers: `clip_observe()`, `df_modal_observe()`, `df_select_observe()`, `file_observe()`, `var_name_observe()`

Examples

```
## Not run:
column_select_observe(ind, paste0("column_", ind))

## End(Not run)
```

df_modal_button	<i>Button to show dataframe modal</i>
-----------------	---------------------------------------

Description

Generates a `shiny::actionButton()` that, when clicked, generates a modal for the resulting dataset for the specified step. Make sure to set up a corresponding observer using `df_modal_observe()`.

Usage

```
df_modal_button(ind, text = "View data")
```

Arguments

`ind` The index of the step.

`text` The text to display on the button.

Value

A `shiny::actionButton()` with id `df_modal_<ind>`.

See Also

[df_modal_observe\(\)](#), which opens the modal this button triggers.

Other step UI: [accordion_panel_remove_button\(\)](#), [select_column_input\(\)](#), [select_dataset_input\(\)](#), [varname_input\(\)](#), [verbatimTextOutput_copy\(\)](#)

Examples

```
df_modal_button(1)
```

df_modal_observe	<i>Show a modal with a reactable data.frame</i>
------------------	---

Description

Installs an observer that opens a modal showing a [DT::datatable\(\)](#) of the named intermediate dataset when the step's "view data" button is clicked. Pair with [df_modal_button\(\)](#).

Usage

```
df_modal_observe(ind, df_name)
```

Arguments

ind	The index of the step.
df_name	The name of the data.frame to be displayed.

Value

Called for its side effects; invisibly returns the observer.

See Also

[df_modal_button\(\)](#), the button that opens this modal.

Other step observers: [clip_observe\(\)](#), [column_select_observe\(\)](#), [df_select_observe\(\)](#), [file_observe\(\)](#), [var_name_observe\(\)](#)

Examples

```
## Not run:
df_modal_observe(ind, step_varname(ind))

## End(Not run)
```

df_select_observe	<i>Keep a selectInput of intermediate data.frames up-to-date</i>
-------------------	--

Description

Installs an observer that keeps the step's dataset dropdown (dataset_<ind>) populated with the available upstream datasets, preserving the current selection where possible and otherwise defaulting to the most recent one.

Usage

```
df_select_observe(ind)
```

Arguments

ind	The index of the step.
-----	------------------------

Value

Called for its side effects; invisibly returns the observer.

See Also

[select_dataset_input\(\)](#), the dropdown this observer updates.

Other step observers: [clip_observe\(\)](#), [column_select_observe\(\)](#), [df_modal_observe\(\)](#), [file_observe\(\)](#), [var_name_observe\(\)](#)

Examples

```
## Not run:
df_select_observe(ind)

## End(Not run)
```

file_observe	<i>Observe a file input to be included in the download bundle</i>
--------------	---

Description

Installs an observer that records the file uploaded through inputId so it is bundled into the downloadable report archive, keyed by the upload's original file name.

Usage

```
file_observe(inputId)
```

Arguments

`inputId` The id of the `shiny::fileInput()` object whose uploaded file should be included in the downloadable bundle.

Value

Called for its side effects; invisibly returns the observer.

See Also

Other step observers: `clip_observe()`, `column_select_observe()`, `df_modal_observe()`, `df_select_observe()`, `var_name_observe()`

Examples

```
## Not run:
file_observe("user_file")

## End(Not run)
```

`get_chunk`*Get the expanded code chunk for a registered step*

Description

Returns the code chunk for `ind`, expanded with a shared context across all currently-registered steps. Use this inside a reactive consumer (e.g., `renderPrint()`, `observeEvent()`) to display or copy a step's code. The chunk is rebuilt whenever any dependency changes.

Usage

```
get_chunk(ind)
```

Arguments

`ind` The index of the step.

Value

A code object suitable for printing or passing to `shinymeta::displayCodeModal()`, or NULL if the step is not registered.

See Also

Other generated code: `workflow_has_errors()`

Examples

```
## Not run:  
# inside renderPrint() or observeEvent() in a module  
get_chunk(ind)  
  
## End(Not run)
```

get_colors	<i>Generate step colors from a step index</i>
------------	---

Description

Maps a step index to a reproducible accent color (with a readable foreground color) drawn from the khroma "smooth rainbow" palette, so each workflow step gets a distinct, stable color for its accordion header and report border.

Usage

```
get_colors(ind)
```

Arguments

ind	The index of the step.
-----	------------------------

Value

A list with two elements: color, the foreground color ("black" or "white", chosen for contrast against the background), and background, the accent color as a hex string.

See Also

Other utilities: [is_shinylive\(\)](#)

Examples

```
get_colors(1)  
get_colors(2)$background
```

get_int_data	<i>Get an intermediate data object</i>
--------------	--

Description

Retrieves a stored intermediate data reactive by name (the counterpart to `set_int_data()`). Call the returned reactive to obtain the data.

Usage

```
get_int_data(name)
```

Arguments

name	The name of the data object to retrieve.
------	--

Value

The stored reactive; call it (e.g. `get_int_data(name)()`) to get the data. Propagates a `req()` failure if name is not registered.

See Also

Other intermediate data: `get_int_dfs()`, `set_int_data()`

Examples

```
## Not run:
df <- get_int_data(input[[paste0("dataset_", ind)]])()

## End(Not run)
```

get_int_dfs	<i>Get the names of all intermediate data.frames for a given step</i>
-------------	---

Description

Returns the names of intermediate datasets produced by steps *earlier* than `ind` in the current workflow order, so a step can only ever consume upstream output. Used to populate the dataset selectors.

Usage

```
get_int_dfs(ind)
```

Arguments

ind	The index of the step.
-----	------------------------

Value

A named character vector of intermediate datasets in workflow order: values are the stable internal ids ([step_varname\(\)](#), e.g. `data_1`) and names are the display labels (a custom name if set, otherwise the id). Empty if none are available upstream.

See Also

Other intermediate data: [get_int_data\(\)](#), [set_int_data\(\)](#)

Examples

```
## Not run:  
# inside a reactive or observer in a module, after shinypal_setup()  
get_int_dfs(ind)  
  
## End(Not run)
```

`is_shinylive`*Detect a shinylive (webR) session*

Description

Returns TRUE when the app is running in a shinylive/webR session (i.e. compiled to WebAssembly via Emscripten) and FALSE in a normal R session. Useful for gating behavior that can't run in the browser, such as loading packages with no WebAssembly build or bundling a downloadable zip.

Usage

```
is_shinylive()
```

Value

A length-1 logical: TRUE under shinylive/webR, otherwise FALSE.

See Also

Other utilities: [get_colors\(\)](#)

Examples

```
is_shinylive()
```

next_step_index	<i>Get the next workflow step index</i>
-----------------	---

Description

Returns a unique, monotonically increasing index for a new workflow step. Each call increments a server-side counter. The counter is initialized by [shinyPAL_setup\(\)](#) and is deliberately never reset while the session is running.

Usage

```
next_step_index()
```

Value

A single positive integer to use as the new step's index.

See Also

Other workflow steps: [add_shinyPAL_data_step\(\)](#), [add_shinyPAL_plot_step\(\)](#), [add_shinyPAL_step\(\)](#), [step_varname\(\)](#)

Examples

```
## Not run:
ind <- next_step_index()

## End(Not run)
```

select_column_input	<i>Select input to choose a column from a specified dataset</i>
---------------------	---

Description

This should be paired with [select_dataset_input\(\)](#).

Usage

```
select_column_input(ind, label = "Choose a column:", default = NULL, ...)
```

Arguments

ind	The index of the step.
label	The label for the select input.
default	The default value for the select input.
...	Additional arguments passed to shiny::varSelectInput() .

Value

A `shiny::varSelectInput()` tag with id `column_<ind>`.

See Also

`column_select_observe()`, which keeps this selector in sync.

Other step UI: `accordion_panel_remove_button()`, `df_modal_button()`, `select_dataset_input()`, `varname_input()`, `verbatimTextOutput_copy()`

Examples

```
## Not run:
select_column_input(ind)

## End(Not run)
```

select_dataset_input	Select input to choose a shinypal intermediate dataset
----------------------	--

Description

A `shiny::selectInput()` listing the intermediate datasets available to step `ind` (those produced by earlier steps), defaulting to the most recent. Keep its choices current with `df_select_observe()`.

Usage

```
select_dataset_input(ind, label = "Choose a dataset:")
```

Arguments

<code>ind</code>	The index of the step.
<code>label</code>	The label for the select input.

Value

A `shiny::selectInput()` tag with id `dataset_<ind>`.

See Also

`df_select_observe()`, which keeps this dropdown populated.

Other step UI: `accordion_panel_remove_button()`, `df_modal_button()`, `select_column_input()`, `varname_input()`, `verbatimTextOutput_copy()`

Examples

```
## Not run:
select_dataset_input(ind)

## End(Not run)
```

set_int_data	<i>Set an intermediate data object</i>
--------------	--

Description

Stores a reactive data object in shinypal's intermediate-data registry under name, making it available to later steps and to the assembled script. Usually called for you by [add_shinypal_data_step\(\)](#).

Usage

```
set_int_data(obj, name)
```

Arguments

obj	A reactive data object to store.
name	A name to store data object as.

Value

Called for its side effects; invisibly returns NULL.

See Also

Other intermediate data: [get_int_data\(\)](#), [get_int_dfs\(\)](#)

Examples

```
## Not run:  
set_int_data(occs, step_varname(ind))  
  
## End(Not run)
```

shinypal_setup	<i>Setup shinypal</i>
----------------	-----------------------

Description

A function to set up the shinypal environment. This function should be called at the top of the server function of your shiny app.

Usage

```
shinypal_setup(
  input,
  output,
  session,
  modules,
  download_filename = "shinypal_script.zip",
  download_template = "../modules/test_report.qmd",
  prefix = "data_"
)
```

Arguments

input	The shiny input object.
output	The shiny output object.
session	The shiny session object.
modules	A character vector of paths to independent modules.
download_filename	The name of the file to download when the user clicks the download button.
download_template	The path to the Quarto markdown template file that will be used to generate the report.
prefix	Prefix for each data step's internal id and the variable name it gets in the generated script (default "data_"). Use step_varname() to retrieve the full value for a given step index.

Details

Each module should have a `ui-aux.R` file that defines the UI elements for that module in the "Workflow" accordion and a `server.R` file that defines the server-side logic for the module.

Value

Called for its side effects and returns NULL invisibly. It initializes shinypal's shared reactive state, renders the live report and wires the download handler and workflow observers, then sources each module's `ui-aux.R` and `server.R` so their steps become available.

See Also

Other app setup: [shinypal_ui\(\)](#)

Examples

```
## Not run:
server <- function(input, output, session) {
  modules <- list.dirs("../modules", recursive = FALSE)
  shinypal_setup(input, output, session, modules)
}
```

```
## End(Not run)
```

shinypal_ui

UI for ShinyPal

Description

A function to create the UI for the ShinyPal app. This function should be called in the UI function of your shiny app.

Usage

```
shinypal_ui(modules)
```

Arguments

modules A character vector of paths to independent modules.

Details

Each module should have a ui-main.R file that defines the UI elements for that module in the "Steps" accordion.

Value

A `shiny::tagList()` holding the shinypal interface: a sidebar layout with the "Possible Workflow Steps" and "Workflow" cards on the left and the live report/download sidebar on the right. Drop it into your app's UI, for example inside a `bslib::page_navbar()` panel.

See Also

Other app setup: [shinypal_setup\(\)](#)

Examples

```
## Not run:
ui <- function() {
  modules <- list.dirs("./modules", recursive = FALSE)
  bslib::page_navbar(title = "My app", bslib::nav_panel("Build", shinypal_ui(modules)))
}

## End(Not run)
```

step_varname	<i>Build a data step's internal id and generated variable name</i>
--------------	--

Description

Returns the name shinypal uses both as a data step's stored-dataset id and as the variable it is assigned in the generated script (set prefix in [shinypal_setup\(\)](#)). A module should pass this as the varname of its [shinymeta::metaReactive2\(\)](#) so the emitted variable name matches the id shinypal stores and renames.

Usage

```
step_varname(ind)
```

Arguments

ind	The index of the step.
-----	------------------------

Value

A length-1 character string, the prefix followed by ind (e.g., "data_1").

See Also

Other workflow steps: [add_shinypal_data_step\(\)](#), [add_shinypal_plot_step\(\)](#), [add_shinypal_step\(\)](#), [next_step_index\(\)](#)

Examples

```
## Not run:  
# inside a module's add-step observer, after shinypal_setup()  
step_varname(ind)  
  
## End(Not run)
```

var_name_observe	<i>Validate and store a custom name for a step's dataset</i>
------------------	--

Description

Installs an observer on the step's varname_<ind> text input. A valid, unique R variable name is recorded in shinypal's var_names registry, becoming the dataset's label in later selectors and its symbol in the generated script. Pair with [varname_input\(\)](#).

Usage

```
var_name_observe(ind)
```

Arguments

`ind` The index of the step.

Value

Called for its side effects; invisibly returns the observer.

See Also

`varname_input()`, the input this observer watches.

Other step observers: `clip_observe()`, `column_select_observe()`, `df_modal_observe()`, `df_select_observe()`, `file_observe()`

Examples

```
## Not run:
var_name_observe(ind)

## End(Not run)
```

varname_input	<i>Text input to give a step's dataset a custom name</i>
---------------	--

Description

A `shiny::textInput()` (id `varname_<ind>`) for naming the dataset a step produces. The name becomes the dataset's label in later selectors and its variable name in the generated script. Pair with `var_name_observe()`, which validates the entry; `add_shinydal_data_step()` inserts both automatically when `rename = TRUE`.

Usage

```
varname_input(ind, label = "Name this dataset (optional):")
```

Arguments

`ind` The index of the step.
`label` The label for the text input.

Value

A `shiny::textInput()` tag with id `varname_<ind>`.

See Also

`var_name_observe()`, which validates and stores the entry.

Other step UI: `accordion_panel_remove_button()`, `df_modal_button()`, `select_column_input()`, `select_dataset_input()`, `verbatimTextOutput_copy()`

Examples

```
## Not run:  
varname_input(ind)  
  
## End(Not run)
```

```
verbatimTextOutput_copy
```

Text element with a button to copy the code to the clipboard

Description

Render a reactive output variable as text within an application page. Uses `shiny::verbatimTextOutput()` which is usually paired with `shiny::renderPrint()` and provides fixed-width text in a `<pre>`. Make sure to set up a corresponding observer using `clip_observe()`.

Usage

```
verbatimTextOutput_copy(ind)
```

Arguments

<code>ind</code>	The index of the step.
------------------	------------------------

Value

A `htmltools::div()` wrapping a `shiny::verbatimTextOutput()` (id `code_<ind>`) and a copy `shiny::actionButton()` (id `copy_<ind>`).

See Also

`clip_observe()`, which copies the displayed code to the clipboard.

Other step UI: `accordion_panel_remove_button()`, `df_modal_button()`, `select_column_input()`, `select_dataset_input()`, `varname_input()`

Examples

```
verbatimTextOutput_copy(1)
```

workflow_has_errors	<i>Check whether the workflow has incomplete or errored steps</i>
---------------------	---

Description

Returns TRUE if any step in the current workflow cannot be expanded into the downloadable script due to failing req()/validate() checks.

Usage

```
workflow_has_errors()
```

Value

A length-one logical.

See Also

Other generated code: [get_chunk\(\)](#)

Examples

```
## Not run:  
if (workflow_has_errors()) {  
  shiny::showNotification("Some steps are incomplete.")  
}  
  
## End(Not run)
```

Index

- * **app setup**
 - shinyapp_setup, 18
 - shinyapp_ui, 20
 - * **generated code**
 - get_chunk, 12
 - workflow_has_errors, 24
 - * **intermediate data**
 - get_int_data, 14
 - get_int_dfs, 14
 - set_int_data, 18
 - * **step UI**
 - accordion_panel_remove_button, 2
 - df_modal_button, 9
 - select_column_input, 16
 - select_dataset_input, 17
 - varname_input, 22
 - verbatimTextOutput_copy, 23
 - * **step observers**
 - clip_observe, 8
 - column_select_observe, 8
 - df_modal_observe, 10
 - df_select_observe, 11
 - file_observe, 11
 - var_name_observe, 21
 - * **utilities**
 - get_colors, 13
 - is_shinylive, 15
 - * **workflow steps**
 - add_shinyapp_data_step, 3
 - add_shinyapp_plot_step, 5
 - add_shinyapp_step, 6
 - next_step_index, 16
 - step_varname, 21
- accordion_panel_remove_button, 2
- accordion_panel_remove_button(), 10, 17, 22, 23
- add_shinyapp_data_step, 3
- add_shinyapp_data_step(), 5–7, 16, 18, 21, 22
- add_shinyapp_plot_step, 5
- add_shinyapp_plot_step(), 4, 7, 16, 21
- add_shinyapp_step, 6
- add_shinyapp_step(), 3–6, 16, 21
- bslib::accordion_panel(), 2, 3
- bslib::page_navbar(), 20
- clip_observe, 8
- clip_observe(), 9–12, 22, 23
- clipr::write_clip(), 8
- column_select_observe, 8
- column_select_observe(), 4, 5, 8, 10–12, 17, 22
- df_modal_button, 9
- df_modal_button(), 3, 10, 17, 22, 23
- df_modal_observe, 10
- df_modal_observe(), 8–12, 22
- df_select_observe, 11
- df_select_observe(), 4, 5, 8–10, 12, 17, 22
- DT::datatable(), 10
- file_observe, 11
- file_observe(), 8–11, 22
- get_chunk, 12
- get_chunk(), 4, 5, 24
- get_colors, 13
- get_colors(), 15
- get_int_data, 14
- get_int_data(), 15, 18
- get_int_dfs, 14
- get_int_dfs(), 14, 18
- htmltools::div(), 23
- is_shinylive, 15
- is_shinylive(), 13
- next_step_index, 16

`next_step_index()`, [4](#), [6](#), [7](#), [21](#)

`rlang::expr()`, [8](#)

`select_column_input`, [16](#)

`select_column_input()`, [3–5](#), [8–10](#), [17](#), [22](#),
[23](#)

`select_dataset_input`, [17](#)

`select_dataset_input()`, [3](#), [8](#), [10](#), [11](#), [16](#),
[17](#), [22](#), [23](#)

`set_int_data`, [18](#)

`set_int_data()`, [14](#), [15](#)

`shiny::actionButton()`, [9](#), [23](#)

`shiny::fileInput()`, [12](#)

`shiny::plotOutput()`, [5](#)

`shiny::renderPrint()`, [23](#)

`shiny::selectInput()`, [17](#)

`shiny::tagList()`, [20](#)

`shiny::textInput()`, [22](#)

`shiny::varSelectInput()`, [8](#), [9](#), [16](#), [17](#)

`shiny::verbatimTextOutput()`, [23](#)

`shiny::displayCodeModal()`, [12](#)

`shiny::expandChain()`, [8](#)

`shiny::metaReactive2()`, [3](#), [21](#)

`shiny::metaRender2()`, [5](#)

`shiny::pal_setup`, [18](#)

`shiny::pal_setup()`, [16](#), [20](#), [21](#)

`shiny::pal_ui`, [20](#)

`shiny::pal_ui()`, [19](#)

`step_varname`, [21](#)

`step_varname()`, [4](#), [6](#), [7](#), [15](#), [16](#), [19](#)

`var_name_observe`, [21](#)

`var_name_observe()`, [4](#), [8–12](#), [22](#)

`varname_input`, [22](#)

`varname_input()`, [3](#), [4](#), [10](#), [17](#), [21–23](#)

`verbatimTextOutput_copy`, [23](#)

`verbatimTextOutput_copy()`, [3](#), [8](#), [10](#), [17](#),
[22](#)

`workflow_has_errors`, [24](#)

`workflow_has_errors()`, [12](#)