

Package ‘blockr.dock’

May 7, 2026

Title A Docking Layout Manager for 'blockr'

Version 0.1.1

Description Building on the docking layout manager provided by 'dockViewR', this provides a flexible front-end to 'blockr.core'. It provides an extension mechanism which allows for providing means to manipulate a board object via panel-based user interface components.

URL <https://bristolmyerssquibb.github.io/blockr.dock/>

BugReports <https://github.com/BristolMyersSquibb/blockr.dock/issues>

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

Imports blockr.core (>= 0.1.2), bsicons, shiny, shinyjs, bslib, glue, dockViewR (>= 0.2.1), htmltools, cli, shinyWidgets, jsonlite

Suggests testthat (>= 3.0.0), DT, colorspace, methods, withr, shinytest2, knitr, rmarkdown

Config/testthat/edition 3

Collate 'action-class.R' 'action-block.R' 'action-link.R'
'action-modal.R' 'action-stack.R' 'action-ui.R'
'action-utils.R' 'block-meta.R' 'block-ui.R' 'board-plugins.R'
'board-server.R' 'board-ui.R' 'dock-board.R' 'dock-stack.R'
'ext-class.R' 'ext-edit.R' 'ext-ui.R' 'layout-class.R'
'plugin-block.R' 'plugin-serdes.R' 'utils-dock.R' 'utils-id.R'
'utils-misc.R' 'utils-pkg.R' 'utils-serdes.R' 'utils-serve.R'
'utils-ui.R' 'layouts-class.R' 'view-ui.R'

VignetteBuilder knitr

NeedsCompilation no

Author Nicolas Bennett [aut, cre],
David Granjon [aut]

Maintainer Nicolas Bennett <nicolas@cynkra.com>

Repository CRAN

Date/Publication 2026-04-29 13:10:17 UTC

Contents

blks_metadata	2
dock_id	3
new_action	4
new_dock_board	6
new_dock_extension	8
new_dock_layout	10
new_dock_layouts	11
new_dock_stack	13
new_edit_board_extension	14
show_panel	14

Index	16
--------------	-----------

blks_metadata	<i>Get block metadata</i>
---------------	---------------------------

Description

Returns various metadata for blocks or block categories, as well as styling for block icons.

Usage

```
blks_metadata(blocks)
```

```
blk_color(category)
```

```
blk_icon_data_uri(icon_svg, color, size = 48, mode = c("uri", "inline"))
```

Arguments

blocks	Blocks passed as blocks or block object
category	Block category
icon_svg	Character string containing the SVG icon markup
color	Hex color code for the background
size	Numeric size in pixels (default: 48)
mode	Switch between URI and inline HTML mode

Details

- `blks_metadata()`: Retrieves metadata given a block or blocks object from the block registry. Can also handle blocks which are not registered and provides default values in that case.
- `blk_color()`: Produces colors using the Okabe-Ito colorblind-friendly palette for a character vector of block categories.
- `blk_icon_data_uri()`: Processes block icons to add color and turn them into square-shaped icons.

Value

Metadata is returned from `blks_metadata()` as a `data.frame` with each row corresponding to a block. Both `blk_color()` and `blk_icon_data_uri()` return character vectors.

Examples

```
blk <- blockr.core::new_dataset_block()
meta <- blks_metadata(blk)

col <- blk_color(meta$category)
blk_icon_data_uri(meta$icon, col)
```

`dock_id`*ID utilities*

Description

Objects, such as blocks and dock extensions carry their own IDs. These can be converted into other ID types, such as panel IDs or "handle" IDs. Panel IDs are used to refer to dock panels, while handle IDs provide "handles" for DOM manipulations. All such IDs inherit from `dock_id` and panel IDs additionally inherit from `dock_panel_id`, while handle IDs inherit from `dock_handle_id`. For panel IDs, depending on whether the panel is showing a block or an extension, the inheritance structure additionally contains `block_panel_id` or `ext_panel_id`, respectively. Similarly, for handle IDs, we have `block_handle_id` and `ext_handle_id`. All `dock_id` objects can be converted back to native IDs, by calling `as_obj_id()`. The utility function `dock_id()` returns a (possibly namespaced) ID of the dock instance that is used to manage all visible panels.

Usage

```
dock_id(ns = NULL)

as_dock_panel_id(x)

as_obj_id(x)

as_block_panel_id(x)

as_ext_panel_id(x)

as_dock_handle_id(x)

as_block_handle_id(x)

as_ext_handle_id(x)
```

Arguments

ns	Namespace prefix
x	Object

Value

Coercion functions `as_block_panel_id()`, `as_ext_panel_id()`, `as_block_handle_id()` and `as_ext_handle_id()` return objects that inherit from `block_panel_id`, `ext_panel_id`, `block_handle_id` and `ext_handle_id` as classed character vectors. The less specific coercion functions `as_dock_panel_id()` and `as_dock_handle_id()` return objects that inherit from `dock_panel_id` and `dock_handle_id`, in addition to a sub-class such as `block_panel_id` or `ext_panel_id` (in the case of `as_dock_panel_id()`). If a mix of sub-classes is returned, this will be represented by a list of classed character vectors. Finally, `as_obj_id()` returns a character vector, as does `dock_id()`.

Examples

```
blks <- c(
  a = blockr.core::new_dataset_block(),
  b = blockr.core::new_head_block()
)

ext <- new_edit_board_extension()

as_dock_panel_id(blks)
as_dock_panel_id(ext)

identical(names(blks), as_obj_id(as_block_panel_id(blks)))

as_dock_handle_id(blks)
as_dock_handle_id(ext)

identical(names(blks), as_obj_id(as_block_handle_id(blks)))
```

new_action

Board actions

Description

Logic including a modal-based UI for board actions such as "append block" or "edit stack" can be specified using action objects, which essentially are classed shiny server functions.

Usage

```
new_action(func, id)

is_action(x)
```

```

is_action_generator(x)

action_id(x)

board_actions(x, ...)

action_triggers(x)

block_input_select(
  block = NULL,
  block_id = NULL,
  links = NULL,
  mode = c("create", "update", "inputs"),
  ...
)

block_registry_selectize(id, blocks = list_blocks())

board_select(id, blocks, selected = NULL, ...)

```

Arguments

func	A function which will be used to create a <code>shiny::moduleServer()</code> .
id	Input ID
x	Object
...	Forwarded to other methods
block	Block object
block_id	Block ID
links	Links object
mode	Switch for determining the return object
blocks	Character vector of block registry IDs
selected	Character vector of pre-selected block (registry) IDs

Details

An action is a function that can be called with arguments `input`, `output` and `session`, behaving as one would expect from a shiny server module function. Actions are typically created by action generator functions, they each have a unique ID and a `shiny::reactiveVal()`-based trigger object (inheriting from `action_trigger`). Action trigger objects implement their own counter-based invalidation mechanism (on top of how reactive values behave).

Value

The constructor `new_action` returns a classed function that inherits from `action`. Inheritance can be checked with functions `is_action()`, `is_action_generator()` checks whether an objects is a function that returns an action object. String-value action IDs can be retrieved with `action_id()`

and the set of actions associated with a board can be enumerated via `board_actions()`. Finally, `action_triggers()` returns a named list of objects suitable for use as action triggers.

For utilities `block_input_select()`, `block_registry_selectize()` and `board_select`, see the respective sections.

`block_input_select()`

Determine input options for a block by removing inputs that are already used and also takes into account some edge-cases, such as variadic blocks. If mode is set as "inputs", this will return a character vector, for "create", the return value of a `shiny::selectizeInput()` call and for "update", the return value of a `shiny::updateSelectizeInput()` call.

`block_registry_selectize()`

This creates UI for a block registry selector via `shiny::selectizeInput()` and returns an object that inherits from `shiny.tag`.

`board_select()`

Block selection UI, enumerating all blocks in a board is available as `board_select()`. An object that inherits from `shiny.tag` is returned, which contains the result from a `shiny::selectizeInput()` call.

new_dock_board

Dock board

Description

Using the docking layout manager provided by `dockViewR`, a `dock_board` extends `blockr.core::new_board()`. In addition to the attributes contained in a core board, this also includes dock extensions (as `extensions`) and the panel arrangement (as `layout`). The layout is always stored internally as a `dock_layouts()` object (multi-view); single-page boards are a degenerate case with one auto-named "Page" view.

Usage

```
new_dock_board(
  blocks = list(),
  links = list(),
  stacks = list(),
  ...,
  extensions = new_dock_extensions(),
  layout = dock_layouts(Page = default_view_grid(blocks, extensions)),
  options = dock_board_options(),
  ctor = NULL,
  pkg = NULL,
  class = character()
)
```

```

is_dock_board(x)
as_dock_board(x, ...)
dock_layout(x)
dock_layout(x) <- value
dock_extensions(x)
dock_extensions(x) <- value
dock_ext_ids(x)
dock_board_options()

```

Arguments

blocks	Set of blocks
links	Set of links
stacks	Set of stacks
...	Further (metadata) attributes
extensions	Dock extensions
layout	A <code>dock_layouts()</code> object, a <code>dock_layout</code> , or a raw grid specification (list). All forms are normalised to <code>dock_layouts</code> .
options	Board-level user settings
ctor, pkg	Constructor information (used for serialization)
class	Board sub-class
x	Board object
value	Replacement value

Details

Dispatch is type-driven: a `dock_layouts` is used as-is, a `dock_layout` is wrapped via `as_dock_layouts()`, and a plain list (raw grid spec) is resolved via `create_dock_layout()` and then wrapped.

Value

The constructor `new_dock_board()` returns a board object, as does the coercion function `as_dock_board()`. Inheritance can be checked using `is_dock_board()`, which returns a boolean. `board_views()` returns the board's `dock_layouts`; `dock_layout()` returns the active view's resolved `dock_layout` and `dock_layout<=()` writes into the active view. The `dock_extensions()` and `dock_extensions<=()` accessors return / set the board's `dock_extension` objects. A character vector of IDs is returned by `dock_ext_ids()` and `dock_board_options()` returns a `board_options` object.

Examples

```
brd <- new_dock_board(c(a = blockr.core::new_dataset_block()))
str(dock_layout(brd), max.level = 2)
```

new_dock_extension *Dock extensions*

Description

Functionality of a dock_board can be extended by supplying one or more dock_extension objects, which essentially provide UI shown in a dock panel that allows for manipulating the board state. A set of dock extensions can be combined into a dock_extensions object.

Usage

```
new_dock_extension(
  server,
  ui,
  name,
  class,
  ctor = sys.parent(),
  pkg = NULL,
  options = new_board_options(),
  ...
)

is_dock_extension(x)

validate_extension(x, ...)

extension_ui(x, id, ...)

extension_server(x, ...)

extension_id(x)

extension_name(x)

extension_ctor(x)

new_dock_extensions(x = list())

is_dock_extensions(x)

validate_extensions(x)
```

```

as_dock_extensions(x, ...)

## S3 method for class 'dock_extensions'
as_dock_extensions(x, ...)

## S3 method for class 'dock_extension'
as_dock_extensions(x, ...)

## S3 method for class 'list'
as_dock_extensions(x, ...)

extension_block_callback(x, ...)

```

Arguments

server	A function returning <code>shiny::moduleServer()</code>
ui	A function with a single argument (ns) returning a <code>shiny.tag</code>
name	Name for extension
class	Extension subclass
ctor	Constructor function name
pkg	Package to look up ctor
options	Board options supplied by an extension
...	Further attributes
x	Extension object
id	Namespace ID

Value

The constructors `new_dock_extension()` and `new_dock_extensions()`, as do the coercion functions `as_dock_extension()` and `as_dock_extensions()`, return objects that inherit from `dock_extension` and `dock_extensions` respectively. This inheritance structure can be checked using `is_dock_extension()` and `is_dock_extensions()`, which both return a boolean. A `dock_extension` can be validated using `validate_extension()` and a `dock_extensions` object using `validate_extensions()`, which return the input object invisibly and throw errors as side-effects. Several getter functions return extension attributes, including `extension_ui()` (a function), `extension_server()` (a function), `extension_id()` (a string), `extension_name()` (a string) and `extension_ctor()` (an object that inherits from `blockr_ctor`).

Examples

```

ext <- new_edit_board_extension()
is_dock_extension(ext)

```

new_dock_layout	<i>Dock layout</i>
-----------------	--------------------

Description

The arrangement of panels in a dock can be specified using a `dock_layout` object. A default layout is available via `default_grid()` which results in two panel groups, the one on the left containing all extension panels and the one on the right all block panels. Complementing the low-level constructor `new_dock_layout()`, a high-level entry point `create_dock_layout()` will create panels for extensions and blocks, which can then be arranged via a nested list of character vectors passed as `grid` argument.

Usage

```
new_dock_layout(grid = NULL, panels = NULL, active_group = NULL)
```

```
default_grid(blocks, extensions)
```

```
create_dock_layout(
  blocks = list(),
  extensions = list(),
  grid = default_grid(blocks, extensions)
)
```

```
is_dock_layout(x)
```

```
validate_dock_layout(x, blocks = character())
```

```
as_dock_layout(x, ...)
```

Arguments

<code>grid, panels, active_group</code>	Layout components
<code>blocks, extensions</code>	Dock board components
<code>x</code>	Object
<code>...</code>	Generic consistency

Value

The constructor `new_dock_layout()`, as does the high-level utility `create_dock_layout()`, as well as the coercion function `as_dock_layout()`, all return a `dock_layout` object. A helper function for specifying a default grid is available as `default_grid()`, which returns a list of character vectors. The validator `validate_dock_layout()` returns its input and throws errors as side-effect and inheritance can be checked using `is_dock_layout` which returns a boolean.

Examples

```
blks <- c(
  a = blockr.core::new_dataset_block(),
  b = blockr.core::new_head_block()
)

exts <- list(
  edit = new_edit_board_extension()
)

grid <- list("edit", list("a", "b"))

layout <- create_dock_layout(blks, exts, grid)
is_dock_layout(layout)
```

new_dock_layouts *Dock views (layouts)*

Description

A dock_board always holds a dock_layouts object (multi-view tabs). Single-page boards are a degenerate case: one auto-named "Page" view. Blocks and extensions are shared across views via the board's DAG; view membership is a layout concern only.

Usage

```
new_dock_layouts(...)

validate_dock_layouts(x)

dock_layouts(...)

dock_view(..., active = FALSE)

is_dock_layouts(x)

view_ids(x)

active_view(x)

active_view(x) <- value

view_can_crud(x)

dock_layouts(x) <- value

board_views(x)
```

```
as_dock_layouts(x, ...)
```

Arguments

...	Generic consistency
x	Object
active	Logical; mark this view as the initially active one. At most one view in a dock_layouts() may be active.
value	Replacement value

Details

Multiple views are defined via `dock_layouts()`, which accepts named list elements – each a (possibly nested) list of block and extension IDs (the same format accepted by `create_dock_layout(grid = ...)`). A view can be marked as the initially active one by tagging its spec with `attr(view, "active") <- TRUE`, conveniently produced by `dock_view()`. If no view is tagged, the first one is used. View CRUD is enabled unless the dock is locked (see `is_dock_locked()`).

Value

`dock_layouts()` returns a `dock_layouts` object. `dock_view()` returns a list (the view spec) with the active attribute set when requested. `is_dock_layouts()` returns a boolean. `active_view()` returns a string and `active_view<-()` returns the modified `dock_layouts` object invisibly. `view_ids()` returns all IDs (block + extension) found in a layout specification. The `view_can_crud()` helper returns `FALSE` when the dock is locked.

Examples

```
# Explicit constructor (first view is active by default)
ly <- dock_layouts(
  Analysis = list("dataset_1", "head_1"),
  Overview = list("dag_extension")
)
is_dock_layouts(ly)
active_view(ly)

# Mark a specific view as initially active
ly2 <- dock_layouts(
  Analysis = list("dataset_1"),
  Overview = dock_view("dag_extension", active = TRUE)
)
active_view(ly2)
```

new_dock_stack	<i>Colored stacks</i>
----------------	-----------------------

Description

While stacks created via `blockr.core::new_stack()` do not keep track of a color attribute, a `dock_stack` object does. Such objects can be created via `new_dock_stack()`. The color attribute can be extracted using `stack_color()` and set with `stack_color<-()`. A new color suggestion, based on existing colors, is available through `suggest_new_colors()`.

Usage

```
new_dock_stack(..., color = suggest_new_colors())

is_dock_stack(x)

stack_color(x)

suggest_new_colors(colors = character(), n = 1)

stack_color(x) <- value

as_dock_stack(x, ...)
```

Arguments

<code>...</code>	Passed to <code>blockr.core::new_stack()</code>
<code>color</code>	String-valued color value (using hex encoding)
<code>x</code>	object
<code>colors</code>	Currently used color values
<code>n</code>	Number of new colors to generate
<code>value</code>	Replacement value

Value

The constructor `new_dock_stack()` returns a "dock_stack" object, which is a stack object as returned by `blockr.core::new_stack()`, with an additional color attribute. Inheritance can be checked using `is_dock_stack()`, which returns a scalar logical and the color attribute can be set and retrieved using `stack_color<-()` (returns the modified stack object invisibly) and `stack_color()` (returns a string), respectively. Stack objects may be coerced to "dock_stack" using `as_dock_stack()` and finally, a utility function `suggest_new_colors()` which returns a character vector of new colors, based on an existing palette.

```
new_edit_board_extension
      Edit board extension
```

Description

A simplistic example of an extension which can be used for manipulating the board via a table-based UI. Mainly relevant for testing purposes.

Usage

```
new_edit_board_extension(...)
```

Arguments

```
...          Forwarded to new_dock_extension()
```

Value

A board extension object that additionally inherits from edit_board_extension.

Examples

```
ext <- new_edit_board_extension()
is_dock_extension(ext)
```

```
show_panel          UI utilities
```

Description

Exported utilities for manipulating dock panels (i.e. displaying panels).

Usage

```
show_panel(id, board, dock, type = c("block", "extension"))
```

Arguments

id	Object ID
board	Board object
dock	Object available as dock in extensions
type	Either "block" or "extensions", depending on what kind of panel should be shown

show_panel

15

Value

NULL, invisibly

Index

`action_id` (`new_action`), 4
`action_triggers` (`new_action`), 4
`active_view` (`new_dock_layouts`), 11
`active_view<-` (`new_dock_layouts`), 11
`as_block_handle_id` (`dock_id`), 3
`as_block_panel_id` (`dock_id`), 3
`as_dock_board` (`new_dock_board`), 6
`as_dock_extensions`
 (`new_dock_extension`), 8
`as_dock_handle_id` (`dock_id`), 3
`as_dock_layout` (`new_dock_layout`), 10
`as_dock_layouts` (`new_dock_layouts`), 11
`as_dock_panel_id` (`dock_id`), 3
`as_dock_stack` (`new_dock_stack`), 13
`as_ext_handle_id` (`dock_id`), 3
`as_ext_panel_id` (`dock_id`), 3
`as_obj_id` (`dock_id`), 3

`blk_color` (`blks_metadata`), 2
`blk_icon_data_uri` (`blks_metadata`), 2
`blks_metadata`, 2
`block_input_select` (`new_action`), 4
`block_registry_selectize` (`new_action`), 4
`blockr.core::new_board()`, 6
`blockr.core::new_stack()`, 13
`board_actions` (`new_action`), 4
`board_select` (`new_action`), 4
`board_views` (`new_dock_layouts`), 11

`create_dock_layout` (`new_dock_layout`), 10

`default_grid` (`new_dock_layout`), 10
`dock_board_options` (`new_dock_board`), 6
`dock_ext_ids` (`new_dock_board`), 6
`dock_extensions` (`new_dock_board`), 6
`dock_extensions<-` (`new_dock_board`), 6
`dock_id`, 3
`dock_layout` (`new_dock_board`), 6
`dock_layout<-` (`new_dock_board`), 6
`dock_layouts` (`new_dock_layouts`), 11

`dock_layouts()`, 6
`dock_layouts<-` (`new_dock_layouts`), 11
`dock_view` (`new_dock_layouts`), 11
`dock_view()`, 12

`extension_block_callback`
 (`new_dock_extension`), 8
`extension_ctor` (`new_dock_extension`), 8
`extension_id` (`new_dock_extension`), 8
`extension_name` (`new_dock_extension`), 8
`extension_server` (`new_dock_extension`), 8
`extension_ui` (`new_dock_extension`), 8

`is_action` (`new_action`), 4
`is_action_generator` (`new_action`), 4
`is_dock_board` (`new_dock_board`), 6
`is_dock_extension` (`new_dock_extension`), 8

`is_dock_extensions`
 (`new_dock_extension`), 8
`is_dock_layout` (`new_dock_layout`), 10
`is_dock_layouts` (`new_dock_layouts`), 11
`is_dock_stack` (`new_dock_stack`), 13

`new_action`, 4
`new_dock_board`, 6
`new_dock_extension`, 8
`new_dock_extensions`
 (`new_dock_extension`), 8
`new_dock_layout`, 10
`new_dock_layouts`, 11
`new_dock_stack`, 13
`new_edit_board_extension`, 14

`shiny::moduleServer()`, 5, 9
`shiny::reactiveVal()`, 5
`shiny::selectizeInput()`, 6
`shiny::updateSelectizeInput()`, 6
`show_panel`, 14
`stack_color` (`new_dock_stack`), 13

`stack_color<- (new_dock_stack)`, 13
`suggest_new_colors (new_dock_stack)`, 13
`validate_dock_layout (new_dock_layout)`,
10
`validate_dock_layouts`
 `(new_dock_layouts)`, 11
`validate_extension`
 `(new_dock_extension)`, 8
`validate_extensions`
 `(new_dock_extension)`, 8
`view_can_crud (new_dock_layouts)`, 11
`view_ids (new_dock_layouts)`, 11