Package 'blockr.dock'

July 13, 2026

Title A Docking Layout Manager for 'blockr'

Version 0.1.2

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

Imports blockr.core (>= 0.1.3), bsicons, shiny, shinyjs, bslib, glue, dockViewR (>= 0.2.1), htmltools, cli, shinyWidgets, jsonlite, utils

Suggests testthat (>= 3.0.0), DT, colorspace, methods, withr, shinytest2, chromote, xml2, knitr, rmarkdown

Config/testthat/edition 3

Collate 'action-class.R' 'action-block.R' 'action-link.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-grid.R' 'dock-layout.R' 'dock-stack.R' 'dock-view.R' 'ext-class.R' 'ext-delta.R' 'ext-edit.R' 'ext-ui.R' 'panel-id.R' 'panel-ops.R' 'panel-ref.R' 'plugin-block.R' 'plugin-serdes.R' 'sidebar-server.R' 'sidebar-block.R' 'sidebar-link.R' 'sidebar-stack.R' 'utils-dock.R' 'utils-misc.R' 'utils-pkg.R' 'utils-serdes.R' 'utils-serve.R' 'utils-ui.R'

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation no

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

Maintainer Nicolas Bennett <nicolas@cynkra.com>

Repository CRAN

Date/Publication 2026-07-13 12:40:02 UTC

Contents

`blk` *Typed panel references*

Description

`blk()` and `ext()` construct references to a block or extension panel for the `views$mod` panel-op grammar (see board_update). They are the public currency: a caller names a block or extension by its **id**, never the `block_panel-` / `ext_panel-` wire prefix – the ref is the codec, and `as.character()` on one yields the canonical panel-id encoding.

Usage

```
blk(id, near = NULL, side = NULL, size = NULL)

ext(id, near = NULL, side = NULL, size = NULL)

is_panel_ref(x)

## S3 method for class 'panel_ref'
as.character(x, ...)
```

Arguments

id	A block or extension id (not the wire-prefixed panel id).
near	A ref or bare id to anchor placement against.
side	Placement direction relative to near: one of within, left, right, above, below.
size	Target size ratio in (0, 1) – consumed by resize, and recorded on add for when the set_size floor lands (#320).
x	An object.
...	Ignored.

Details

Each ref optionally carries its own placement hint, so a verb's operands are an unnamed list of self-describing refs: add = list(blk("a", near = "b", side = "right"), ext("dag")). Which hint fields are meaningful depends on the verb – add consumes near / side / size, move consumes near / side – and a hint on a ref used where no placement happens (rm, select, a near anchor, or the dock_grid() / panels() authoring DSL) is a loud error. Because the hints are constructor arguments, a misspelled one (blk("a", sise = 0.4)) fails at the call site with R's own unused-argument error, before any payload exists.

Bare id strings are accepted as sugar wherever a ref is, resolved block-first with a hard error only on a true cross-namespace clash (an id that is both a block and an extension), which then demands a typed ref.

Value

blk() / ext() return a panel_ref. as.character() on one returns its canonical panel id, and is_panel_ref() returns a boolean.

Examples

```
blk("my_block", near = "other_block", side = "right")
ext("dag")
as.character(blk("my_block"))
```

---

blks_metadata *Get block metadata*

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"))

block_status_badge(status, error_count = 0L)
```

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
status	A block eval status. waiting, unset and failed carry a badge; ready carries none; dormant is indeterminate; any other value yields no badge. size is the coloured dot's pixel diameter and ring its white outline width, both shared so the dock card icon and the DAG node badge render identically.
error_count	Number of error conditions the block has raised. A positive count promotes the badge to failed, catching render-phase errors that leave the eval status ready.

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.
- block_status_badge(): Derives a block's status badge from its eval status and error count – the single derivation the dock card icon and the blockr.dag node badge share, so they always render the same colour and styling. Returns a styling list (draw the badge), NULL (no badge), or NA (indeterminate: the status is not computed, so leave any existing badge unchanged).

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. block_status_badge() returns a list with color, label, size, ring and ring_color (the badge to draw), NULL for a status with no badge, or NA when the status is indeterminate.

Examples

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

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

block_status_badge("waiting")
```

`dock-grid` *Canonical view grid*

Description

A `dock_grid` is a view's geometry in our compact, dockView-independent form – nested splits and tab groups with sizes normalised to 0-1 ratios and no volatile ids, so two casts of the same layout compare `identical()`. It is authored with dock_grid() and produced by `as_dock_grid()`, which casts another `dock_grid` (identity) or a dock_layout (dockView's client echo) into it and is idempotent. `is_dock_grid()` is the class check; `validate_dock_grid()` returns its input and errors on a malformed or non-canonical grid.

Usage

```
as_dock_grid(x, ...)

is_dock_grid(x)

validate_dock_grid(x)
```

Arguments

`x`	Object to cast (a `dock_grid` or a dock_layout), validate, or test.
`...`	Passed on to methods.

Value

`as_dock_grid()` and `validate_dock_grid()` a `dock_grid`; `is_dock_grid()` a boolean.

dock-layout *Dock layout: dockView's native representation*

Description

A dock_layout is dockView's own layout representation – the {grid, panels, activeGroup} payload it renders – the wire form at the client boundary. It is distinct from our dock_grid (a view's canonical geometry) and dock_view() (a view's structure): a dock_layout is what the browser echoes and what a restore pushes back, geometry fused with resolved panel content in dockView's own shape. It is produced and consumed only at the dockView seam and is never stored on the board nor passed through the update lifecycle.

Usage

```
new_dock_layout(x = list())

is_dock_layout(x)

validate_dock_layout(x)

as_dock_layout(x, ...)
```

Arguments

x	Object to wrap, validate, test, or cast.
...	Passed on to methods (e.g. blocks / extensions to resolve panel content when casting a dock_grid).

Details

new_dock_layout() wraps a raw dockView state / payload list (a client echo) as a dock_layout; is_dock_layout() is the class check; validate_dock_layout() returns its input and errors on a malformed shape. Cast across the seam with as_dock_layout() (build the dockView payload from a dock_grid against the board's blocks and extensions), as_dock_grid() (canonical geometry from a layout) and as_dock_view() (membership from a layout).

Value

new_dock_layout(), validate_dock_layout() and as_dock_layout() return a dock_layout; is_dock_layout() a boolean.

dock_grid *Dock grid authoring*

Description

A dock_grid is a view's geometry: the arrangement of its panels into nested splits and tab groups, with sizes. `dock_grid()` is the authoring DSL that builds one; view *membership* (which panels belong) and the display *name* live on the view (see dock_view()), not the grid.

Usage

```
dock_grid(..., orientation = c("horizontal", "vertical"), sizes = NULL)

panels(..., active = NULL)

group(..., sizes = NULL)

default_layout(blocks, extensions)
```

Arguments

`...`	For `dock_grid()` and `group()`, grid children (`blk()` / `ext()` references, bare ids, character vectors, lists, `panels()`, or `group()`). For `panels()`, panel references or ids. A reference used here cannot carry a placement hint. Otherwise reserved for generic consistency.
`orientation`	Top-level split direction; one of `"horizontal"` (default) or `"vertical"`.
`sizes`	Numeric vector parallel to `...`, giving each child's share of the parent (positive; need not sum to 1).
`active`	For `panels()`, the id of the tab to open by default.
`blocks, extensions`	Dock board components to arrange (for `default_layout()`).

Details

Construct a grid with:

- `dock_grid(...)`: the page-level container. Its `...` are the children of the root branch. Bare strings become single-panel leaves, character vectors become tabbed leaves, lists become nested branches. Use `panels()` for a tabbed leaf with an explicit open tab, and `group()` for a branch with explicit sizes.
- `panels(..., active = NULL)`: a tabbed leaf whose tab strip holds the given panel ids. `active` selects the initially-open tab; the first id wins by default. A single-panel `panels()` is permitted but redundant (a bare string is equivalent).
- `group(..., sizes = NULL)`: a branch container. `sizes` is a numeric vector parallel to `...` that overrides the even split.

- `default_layout(blocks, extensions)` produces the default board arrangement (extensions on top, blocks below) as a `list(views, grids)` the constructor consumes.

`dock_grid()` accepts `orientation = "horizontal" | "vertical"` for the top-level split direction and `sizes` for the root-branch ratios. The dockView-native `{grid, panels, activeGroup}` payload dockViewR consumes is a dock_layout, built from a grid against the board's blocks and extensions.

Value

`dock_grid()` returns a dock_grid object. `panels()` returns a `dock_panels` node and `group()` returns a `dock_group` node – both are grid sub-trees usable inside `dock_grid()` / `group()`. `default_layout()` returns a `list` with `views` (a `dock_views`) and `grids` (a `dock_grids`).

Examples

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

# Panels named with typed references; bare id strings work too
panels(blk("a"), blk("b"), active = blk("b"))

# Branch with explicit child ratios
group(blk("a"), blk("b"), sizes = c(0.3, 0.7))

# An extension panel beside a tabbed block leaf
dock_grid(
  ext("dag"),
  panels(blk("a"), blk("b"), active = blk("b")),
  sizes = c(0.3, 0.7)
)

# Vertical top-level split
dock_grid(blk("a"), blk("b"), orientation = "vertical")
```

---

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 for block / extension cards, plus view_handle_id for a view's DOM container. 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)

as_view_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. as_view_handle_id() maps a view id to its DOM container id (a view_handle_id). 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)))
```

`is_dock_grids` *Dock views: structure and grid*

Description

A `dock_board` stores its views as two independent slots. **Structure** – which panels belong to each view, plus the view names, ids and the active view – is a `dock_views` collection of `dock_view` objects, read with `board_views()`. **Grid** – the geometry of each view (nesting, tab groups, sizes) – is a separate, `NULL`-valid `dock_grids` slot, read with `board_grids()`. 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

```
is_dock_grids(x)

validate_dock_grids(x, views = NULL)

board_grids(x)

board_grids(x) <- value

as_dock_view(x, ...)

dock_view(members = character(), name = NULL)

is_dock_view(x)

validate_dock_view(x)

view_members(x)

is_dock_views(x)

validate_dock_views(x)

view_name(x)
```

```
view_name(x) <- value

view_names(x)

active_view(x)

active_view(x) <- value

board_views(x)

board_views(x) <- value
```

Arguments

x	An object appropriate to the function: a dock_view (for view_name(), view_members()), a dock_views collection (for view_names(), active_view()), a dock_grids, or a dock_board (for the board accessors).
views	A dock_views collection, used to check that grids key known views.
value	Replacement value
...	Forwarded to methods.
members	Ordered character vector of panel ids.
name	Optional display name for the view.

Details

Each view carries a stable, immutable **id** (its key in the collection) distinct from its editable display **name**. This mirrors the id / name separation used for blocks (an immutable id keys the collection; blockr.core::block_name() is an editable label). The id is minted once when the view is created and never changes – rename only rewrites the name attribute, never the key. Use dock_view() to construct a view, view_name() / view_name<-() to read and write its display name, view_names() for all names in a collection, and view_members() for a dock_view's ordered panel-id set.

Multi-view boards are defined by passing views (and optionally grids) to new_dock_board(): views is a named list keyed by view id (minted when absent), each value a dock_view(), a bare character vector of member panel ids, or a list of panel ids. grids is a named list keyed by view id whose values are dock_grid()s; it is optional – a view with no grid entry falls back to a default grid over its members wherever placement geometry is needed. Bare block / extension ids in either slot are resolved to canonical panel ids against the board's blocks and extensions. The initially-active view is chosen by new_dock_board(active =) (a view id), defaulting to the first; it is a property of the collection, never of an individual view. View CRUD is enabled unless the dock is locked (see is_dock_locked()).

Structure and grid are related by total semantics, not containment: a member with no grid entry is an un-landed intent, a grid entry with no membership an inert ghost. Both are legal on a committed board and reconciled only where placement is read (view_grid() prunes ghosts and shows un-landed members via a default) – the board is valid with no grid at all. Referential integrity still holds: every member must reference a block or extension on the board.

Value

board_views() returns a dock_views, board_grids() a dock_grids or NULL, and their setters the modified board invisibly. dock_view() returns a dock_view and view_members() a character vector. is_dock_view() / is_dock_views() / is_dock_grids() return a boolean; validate_dock_view(), validate_dock_views() and validate_dock_grids() return their (validated) input and throw on error. active_view() returns the active view's id, or NULL when no view is active, and active_view<-() the modified collection (or dock_board) invisibly. view_name() returns a view's explicit display name (or NULL), view_name<-() the modified view, and view_names() a character vector of display labels keyed by view id (derived from the id where a view has no explicit name). as_dock_view() returns a dock_view: identity on a dock_view, or a view whose members are a dock_layout's panel ids.

Examples

```
brd <- new_dock_board(
  blocks = c(
    dataset_1 = blockr.core::new_dataset_block(),
    head_1 = blockr.core::new_head_block()
  ),
  views = list(
    analysis = dock_view(c("dataset_1", "head_1"), name = "Analysis"),
    overview = "dataset_1"
  ),
  active = "overview"
)
view_names(board_views(brd))
view_members(board_views(brd)[["analysis"]])
```

---

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 shiny::moduleServer().
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 per-view layout, stored as two independent slots – view structure (`board_views()`) and grid geometry (`board_grids()`). 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(),
  views = NULL,
  grids = NULL,
  active = NULL,
  options = dock_board_options(),
  ctor = NULL,
  pkg = NULL,
  class = character()
)

is_dock_board(x)
```

```
as_dock_board(x, ...)

dock_extensions(x)

dock_extensions(x) <- value

dock_ext_ids(x)

extension_ids(x, class = NULL)

dock_board_options()
```

Arguments

blocks	Set of blocks
links	Set of links
stacks	Set of stacks
...	Further (metadata) attributes
extensions	Dock extensions. Each is keyed by its list name, or by its class stripped of the _extension suffix when unnamed (so new_dag_extension() becomes dag); that key names the extension's panel and is what ext() resolves against.
views	Per-view membership: a named list keyed by view id (minted when absent), each value a dock_view(), or member panels named with blk() / ext() references or bare ids (as a vector or list). NULL yields a single default view over the board's blocks and extensions.
grids	Per-view geometry: a named list keyed by view id whose values are dock_grid()s, or a dock_grids. Optional – a view with no grid entry falls back to a default grid over its members.
active	Id of the initially active view (a key of views). Defaults to the first view. Which view is active is a property of the collection, not of an individual view.
options	Board-level user settings
ctor, pkg	Constructor information (used for serialization)
class	Optional extension subclass; when supplied, only ids of extensions inheriting from it are returned.
x	Board object
value	Replacement value

Details

Multi-view boards pass views (and optionally grids); see dock_view() and dock_grid() for the input forms and board_views() for how they combine. Bare block / extension ids are resolved against the board's blocks and extensions. With views = NULL the board gets a single default view.

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. 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()))
view_members(board_views(brd)[[1L]])
```

---

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,
  description = NULL,
  ctor = sys.parent(),
  pkg = NULL,
  options = new_board_options(),
  external_ctrl = FALSE,
  ...
)

is_dock_extension(x)

validate_extension(x, ...)

extension_ui(x, key, id, ...)

extension_server(x, key, ...)

extension_id(x)
```

```
extension_name(x)

extension_description(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 shiny::moduleServer(). Beyond id, it is called with the board handles board, update, view_data and actions, plus extensions – an environment exposing every extension's server result keyed by ID (each carrying its state), so one extension can read another's state via extensions[[id]]. view_data is a reactive holding the live all-views layout (the committed board is board); it is NULL until every view has reported its layout once, so req() it.
ui	A function with a single argument (ns) returning a shiny.tag
name	Name for extension
class	Extension subclass
description	Optional free-text description of the extension, surfaced as consumer-neutral metadata (e.g. to the AI assistant)
ctor	Constructor function name
pkg	Package to look up ctor
options	Board options supplied by an extension
external_ctrl	Set up external control (experimental). FALSE (the default) opts out; TRUE exposes every constructor input as externally controllable; a character vector names a subset of them.
...	Further attributes
x	Extension object

key	Container-assigned extension id (list key)
id	Namespace ID

Value

The constructors new_dock_extension() and new_dock_extension(), as do the coercion function as_dock_extension() and as_dock_extension(), 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), extension_description() (a string or NULL) and extension_ctor() (an object that inherits from blockr_ctor).

Examples

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

---

new_dock_stack *Colored stacks*

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

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

`panel_obj_ids` *Panel IDs of a grid or layout*

Description

`layout_panel_ids()` returns the canonical panel IDs (`block_panel-...` / `ext_panel-...`) a dock_grid or dock_layout references; `panel_obj_ids()` strips those prefixes back to the bare block / extension IDs.

Usage

```
panel_obj_ids(ids)

layout_panel_ids(layout)
```

Arguments

`ids`	Character vector of panel IDs.
`layout`	A dock_grid or dock_layout.

Value

Character vectors of IDs.

Index