Package: blockr.assistant (via r-universe)

November 19, 2026

Title An LLM Assistant Extension for 'blockr'

Version 0.1.0.9000

Description Building on the docking layout manager provided by 'blockr.dock', this extension hosts an 'ellmer'-powered chat interface alongside a 'blockr' board. The chat is equipped with tools that allow a large language model to inspect and manipulate board state -- adding, removing and modifying blocks, links and stacks -- via the same update channel used by interactive user actions.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

Imports blockr.core (>= 0.1.4), blockr.dock (>= 0.1.3), blockr.ui, bsicons, btw, ellmer, evaluate, glue, grDevices, htmltools, jsonlite, later, promises, R6, shiny, shinychat (>= 0.5.0), shinyWidgets, typedjson, yaml

Suggests chromote, httr2, knitr, rmarkdown, shinytest2, testthat (>= 3.0.0), withr

Remotes BristolMyersSquibb/blockr.core, BristolMyersSquibb/blockr.dock, BristolMyersSquibb/blockr.ui

Config/testthat/edition 3

URL https://bristolmyerssquibb.github.io/blockr.assistant/

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libxml2-dev libssl-dev libx11-dev xz-utils zlib1g-dev libclang-dev

Repository https://cynkra.r-universe.dev

Date/Publication 2026-11-05 09:28:28 UTC

RemoteUrl https://github.com/BristolMyersSquibb/blockr.assistant

RemoteRef HEAD

RemoteSha fcbdb2756a8965915159b7b5ec51197ae458c2c8

Contents

default_system_prompt 2
describe_block 3
describe_board 4
describe_result 5
describe_stack 6
new_assistant_extension 7

Index **10**

`default_system_prompt` *Default assistant system prompt*

Description

Builds the three-section system prompt the assistant ships by default: an intro / conventions block, a one-line-per-skill catalogue of the deployment's globally-scoped skills, and a compact board summary.

Usage

```
default_system_prompt(
  board = NULL,
  client = NULL,
  view_data = NULL,
  skills = NULL,
  focus = NULL,
  in_view = NULL,
  last_active = NULL,
  ...
)
```

Arguments

`board`	Reactive containing the live board, as supplied to the extension server. `NULL` omits the board section.
`client`	An `ellmer::Chat`. Unused by this composer, and passed for the benefit of custom ones – the model and token state are read off it.
`view_data`	Reactive holding `blockr.dock`'s live all-views layout (a `list(views, grids)`), as supplied to the extension server, or `NULL`. Read for the board section's view summary, falling back to the committed `board` when `NULL` (before every view has reported its layout).
`skills`	The available skills, as supplied to the extension server. `NULL` omits the skill catalogue. Only globally-scoped skills are listed here; block- and extension-scoped ones surface through the tools that describe their target.

`focus`	Character vector of block IDs the user has singled out as what they are working on, as added with the `@` under the extension's composer.
`in_view`	Character vector of the block IDs in the current view, in screen order, as the extension reads them off the dock.
`last_active`	The ID of the block the user last clicked in the current view, or `NULL`, including when the user left it out.
	The focus section, built from `focus`, `in_view` and `last_active`, is omitted when none of them names a block still on `board`.
`...`	Forward-compatibility slot for future inputs.

Details

The registered tools are deliberately not catalogued here. They reach the model as a structured tool manifest alongside the system prompt, so a prompt-side listing duplicates them – and the per-block-type tools come and go within a conversation, so any such listing would go stale mid-turn.

Each argument is optional; the corresponding section is omitted when its input is `NULL`, so `default_system_prompt()` at the REPL returns just the intro block – useful for inspecting what the default looks like without mounting a board.

This is the default value of `new_assistant_extension`'s `system_prompt` argument. Custom functions passed in its place receive `(board, client, ...)`; the `...` is forward-compatibility headroom for future phases adding inputs (accept `...` in custom functions so the call site can grow without breaking you).

Value

A character scalar.

`describe_block` *Describe a block for the LLM*

Description

Generic backing the `describe_block` assistant tool. The default method `describe_block.block` reports the block's class, name, arguments, external-control declaration, and incoming links. The bulk of the description comes from `base::format()` on the block; incoming links are computed from the board snapshot supplied alongside. Block authors override their class when the default isn't enough.

Usage

```
describe_block(x, board, id, ..., state = NULL)

## S3 method for class 'block'
describe_block(x, board, id, ..., state = NULL)
```

Arguments

x	A block.
board	The current board snapshot, for resolving link metadata.
id	The id under which the block lives on the board (the name attribute of the enclosing blocks object – block objects themselves do not carry an id because ids must be unique while block-level fields are user-supplied and need not be).
...	Passed on to methods.
state	Live block state as a named list covering the block's constructor inputs, rendered in place of the values the block was constructed with. The block object carries only its constructor frame, which is fixed at construction, so the NULL default reports load-time values however long ago a commit or an edit in the block's own UI moved them on.

Value

Character vector of lines (consistent with summarise_result() and describe_stack()). The tool that consumes this collapses with paste(..., collapse = "\n") before returning to ellmer.

---

describe_board *Describe a board for the LLM*

Description

Generic backing the system prompt's board summary. The default method describe_board.board reports the board's blocks, links, stacks and options; describe_board.dock_board adds the view and extension summaries via NextMethod(). Board sub-classes extend the summary by supplying their own method.

Usage

```
describe_board(b, markers, ...)

## S3 method for class 'board'
describe_board(b, markers, ...)

## S3 method for class 'dock_board'
describe_board(b, markers, ..., view_data = NULL)
```

Arguments

b	A board.
markers	Named character vector of per-block markers carrying the block's eval status and captured conditions, such as [stale] [1 error], as produced by block_markers().
...	Passed to methods.

`view_data` Reactive holding blockr.dock's live all-views layout, or `NULL` to read views from the committed board. Views are a `dock_board` concept, so this is consumed only by the `dock_board` method – where it appears in the signature – not by the base `board` method.

Value

Character vector of lines, collapsed with `paste(..., collapse = "\n")` by `summarise_board()`.

`describe_result` *Describe a block result for the LLM*

Description

Generic backing the result summaries the assistant feeds the model: the `get_block_result` tool and the post-apply review it sends itself. The default method delegates to `btw::btw_this()`. A package contributing an unusual result type can add a method to describe it directly, in blockr terms, instead of supplying a `btw::btw_this()` method.

Usage

```
describe_result(x, ...)

## Default S3 method:
describe_result(x, ...)

## S3 method for class 'evaluate_evaluation'
describe_result(x, ...)

## S3 method for class 'recordedplot'
describe_result(x, ...)

## S3 method for class 'source'
describe_result(x, ...)

## S3 method for class 'condition'
describe_result(x, ...)
```

Arguments

`x` A block result (any R object).

`...` Passed on to methods (e.g. `btw::btw_this()`).

Details

Methods for recorded plots ship here, since a plot block built on `blockr.core::new_plot_block()` evaluates to recordings and the default renders their display list – a list of graphics primitives, not a description of the chart. They name the class and count the recordings, and say when a block evaluated without drawing. Neither describes the chart itself: `inspect_results` renders it, by drawing it on a device.

The `evaluate::evaluate()` method describes each component through the generic rather than reading the container as one shape, so an evaluation mixing plots with output and conditions is handled by whatever methods exist for its parts – including ones another package adds. That is what keeps a warning's message from being reduced to a count of warnings. To describe a whole result differently, give it a class of its own and register a method on that, which takes precedence over all of these.

Methods need not bound their output or guard their own errors: the internal `summarise_result()` wrapper caps the text before it reaches the prompt and turns a failed description into a surfaced error message. It is what the tool and the review actually call.

Value

Character vector of lines, consistent with `describe_block()` and `describe_stack()`; the caller collapses with `paste(collapse = "\n")`.

`describe_stack` *Describe a stack for the LLM*

Description

Generic backing the `description` column of the `list_stacks` tool. The default method `describe_stack.stack` summarises the base stack fields (name and comma-separated block ids). Stack classes that carry extra attributes (e.g. `blockr.dag::dag_stack` with a colour) override their class to surface those attributes.

Usage

```
describe_stack(x, ...)

## S3 method for class 'stack'
describe_stack(x, ...)
```

Arguments

`x`	A `stack`.
`...`	For future use.

Value

Character vector of lines (consistent with `summarise_result()` and `describe_block()`). The tool that consumes this collapses with `paste(..., collapse = "\n")` before returning to ellmer.

new_assistant_extension *Assistant extension*

Description

Mounts an ellmer-powered chat panel on a blockr.dock board. The chat client is built from the board's llm_model option and wired with the read and mutation tools; the system prompt is refreshed on every materialized board change so the model always sees the current shape of the board.

Usage

```
new_assistant_extension(
  system_prompt = default_system_prompt,
  threads = NULL,
  ...
)
```

Arguments

system_prompt	Either a function (called each refresh with (board, client, ...) to build the prompt) or a character scalar (used verbatim, no refresh). Defaults to the exported default_system_prompt function.
threads	Optional named list of conversation records to seed the history with, keyed by conversation id. Defaults to no threads. This is how a restored board seeds the chats it was saved with.
...	Forwarded to blockr.dock::new_dock_extension().

Details

Tools of the deployment's own come from the blockr.assistant_tools option, set in the app's startup script to a list of ellmer::tool() definitions, and every client is built with them next to the assistant's. Like chat_save_turns below, they describe the deployment rather than the board, so a saved board carries none of them. A name already in use, by one of the assistant's own tools (the typed add_<type> and modify_<type> included) or by another tool in the list, fails the client build with an error the app shows, rather than leaving one of the two tools unreachable.

The system_prompt argument controls the prompt the model sees:

- A **function** is called on every refresh with (board, client, ...) and must return a character scalar. The default default_system_prompt() composes a four-section prompt (intro / tool catalogue / skill catalogue / board summary); a caller can pass any function of the same shape.
- A **character scalar** is used verbatim as a static prompt – no refresh, no auto-appended catalogue or board summary. The deal is "give up dynamic context, gain full prompt control". Block- and extension-scoped skills are unaffected: those ride in tool return values rather than the prompt.

The `state` shape mirrors the constructor: `system_prompt` (when the caller passed a string) round-trips through `blockr.dock`'s ser/des. Function-valued `system_prompt` is omitted from `state` so restore falls back to the constructor default (functions don't serialise robustly across sessions).

Conversations are saved alongside it and restored into the chat when the board is reopened. The chat holds several of them: the history control in the footer opens a drawer listing every thread on this board, with switching, renaming, deletion and model-written titles, and the whole set rides into `state` as recorded turns. How many of the most recent turns are written **per thread** is read at save time from the `blockr.chat_save_turns` option (or the `BLOCKR_CHAT_SAVE_TURNS` environment variable), which takes `0` for none, a positive whole number, or `Inf` for all, and defaults to 50. Setting it to `0` is worth considering where boards are shared, since the file otherwise carries whatever was typed into every thread; at `0` the chat is not read at save time either, not read and then discarded. It describes the deployment rather than the board, so it is neither a constructor argument nor part of `state` – restore reads whatever the file holds.

Turns go in as they stand because blockr.core writes board files with typedjson, which carries what plain JSON cannot. A board holding anything else under `history` opens without its conversation rather than on a guess at one. The raw provider response is stripped before saving, and a thread trimmed to the budget is cut between exchanges rather than inside one, so it never opens on half of a tool call. A thread reaches `state` once shinychat has recorded it, which is once the model has answered, so a question typed but not yet answered when the board is saved is not in the file.

Which thread is open is remembered by the browser rather than in `state`, so a board reopened elsewhere lists its threads without selecting one.

The live conversation is bounded separately, since saving bounds only the file: every turn is resent on every request, so an unbounded session costs more as it goes and eventually exceeds the provider's context window – and because that limit is reached by accumulation, every later message is over it too, leaving the chat dead until the extension is remounted. Once an exchange exceeds the threshold, the older part of the conversation is replaced by a summary the model writes of it, and the recent turns are kept verbatim. The threshold counts what the provider itself billed for the last exchange rather than turns, that being what the context window is actually spent in. A restored board is checked on mount as well, so reopening a long conversation cannot land already over the limit.

That threshold is a **board option**, `chat_compact_tokens`, so a user can retune it during a session – the trade is recall against how soon the chat starts summarising, and the right answer moves with the model, which is itself swappable at runtime. It defaults to `Inf`, which leaves compaction **off**: a threshold that would suit one provider's context window is wrong for another's, and nothing in the API reports that window, so a number picked here would be a guess – silently inert on a small-context model and needlessly destructive on a large one. A deployment that knows its models sets the starting point through the `blockr.chat_compact_tokens` option or the `BLOCKR_CHAT_COMPACT_TOKENS` environment variable, and a user can pick a value for their own session. Contrast `chat_save_turns`, which stays a deployment setting because it governs whether conversations may land in a shared file at all – not a decision to hand to the person whose conversation it is.

How much survives a compaction is the companion board option, `chat_compact_keep` (deployment default `blockr.chat_compact_keep`, 8): the count of most recent turns left verbatim, with everything older becoming the summary. It is offered on doubling rungs to 256, since a turn count needs no `Inf` and stops meaning much at the top – keeping 256 turns verbatim is already barely compacting. The figure is a preference rather than a floor: a conversation that exceeds the threshold in fewer turns than this is still compacted, or the bound would be inert exactly where it is needed.

Compaction rewrites the browser transcript to match, which is what keeps the two honest. `shinychat` appends each turn to the DOM as it arrives and never reads the client back, so turns dropped from the client would otherwise stay on screen unremembered. The same replay fills in a transcript that a board saved before threads existed leaves empty; tool traffic carries no text and is not replayed.

Both the conversation and its size are reachable from the chat's command palette, which lists two built-in commands alongside the user-invocable skills. The `/compact` command runs the same summarise-and-replace on demand, without waiting for the threshold – for a thread that has gone stale rather than large, where a long build has finished and the next question is unrelated to it. The `/clear` command starts a new conversation, as the history drawer's New button does: the thread on screen is kept in the drawer rather than extended by the next answer, and the chat empties along with the turns the model is sent, the token meter, the block focus and any changes staged but never committed.

Value

A `dock_extension` object additionally inheriting from `assistant_extension`.

Examples

```
ext <- new_assistant_extension()
blockr.dock::is_dock_extension(ext)
```

Index

base::format(), *3*
blockr.core::new_plot_block(), *6*
blockr.dock::new_dock_extension(), *7*
btw::btw_this(), *5*

chat_save_turns, *8*

default_system_prompt, 2, *7*
default_system_prompt(), *7*
describe_block, 3
describe_block(), *6*
describe_board, 4
describe_result, 5
describe_stack, 6
describe_stack(), *4*, *6*

ellmer::tool(), *7*
evaluate::evaluate(), *6*

new_assistant_extension, 7