Package: blockr.dplyr (via r-universe)

May 26, 2026

Title Interactive 'dplyr' Data Transformation Blocks

Version 0.1.0.9001

Description Extends 'blockr.core' with interactive blocks for visual data wrangling using 'dplyr' and 'tidyr' operations. Users can build data transformation pipelines through a graphical interface without writing code directly. Includes blocks for filtering, selecting, mutating, summarizing, joining, and arranging data, with support for complex expressions, grouping operations, and real-time validation.

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

BugReports https://github.com/BristolMyersSquibb/blockr.dplyr/issues

License GPL (>= 3)

Depends R (>= 4.1.0)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports shiny, blockr.core (>= 0.1.2), dplyr, tidyr, htmltools, jsonlite, utils

Suggests shinytest2, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Config/testthat/parallel true

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

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

Date/Publication 2026-05-03 06:16:23 UTC

RemoteUrl https://github.com/bristolmyerssquibb/blockr.dplyr

RemoteRef HEAD

RemoteSha ae87868b121d222010634c6f108867bfc96f65e8

Contents

`blockr_blocks_css_dep` *HTML dependency for blockr-blocks.css (shared block layout styles)*

Description

Exported for reuse by other blockr packages that reuse the shared block-container / row / popover styles.

Usage

```
blockr_blocks_css_dep()
```

Value

An `htmltools::htmlDependency`.

blockr_core_js_dep *HTML dependency for blockr-core.js (namespace + shared utilities)*

Description

Exported for reuse by other blockr packages that build on the shared JS namespace (e.g. blockr.dm).

Usage

```
blockr_core_js_dep()
```

Value

An `htmltools::htmlDependency`.

blockr_input_dep *HTML dependency for blockr-input component (code autocomplete)*

Description

Exported for reuse by other blockr packages that embed the shared expression-input component.

Usage

```
blockr_input_dep()
```

Value

An `htmltools::tagList` of `htmlDependency` objects.

blockr_select_dep *HTML dependency for blockr-select component*

Description

Exported for reuse by other blockr packages that embed the shared select component (e.g. blockr.dm's table pickers).

Usage

```
blockr_select_dep()
```

Value

An `htmltools::tagList` of `htmlDependency` objects.

build_column_picker_meta *Build a lightweight column summary for pickers (name + label)*

Description

Returns one entry per column: `list(name, label)`. Skips the type detection and `anyNA()` sweep that `build_column_summary()` does — those are only needed by the filter block's type-aware condition UI. All other picker blocks only need the name for the option value and the optional label for the secondary text slot, so this helper is the lean path.

Usage

```
build_column_picker_meta(df)
```

Arguments

`df`	A data frame

Details

`label` is read from `attr(col, "label")` (e.g. ADaM datasets). It is an empty string when the attribute is missing or not a length-1 character.

Value

A list of `list(name, label)` entries, one per column.

build_column_summary *Build lightweight column summary for JS (names + types only)*

Description

Returns `name`, `type`, `hasNA`, and `label` for each column — no unique values. Sent on data arrival so the filter UI can render column dropdowns instantly. Unique values are fetched on-demand via `build_column_values()`.

Usage

```
build_column_summary(df)
```

Arguments

`df`	A data frame

Details

`label` is read from `attr(col, "label")` (e.g. ADaM datasets). It is an empty string when the attribute is missing or not a length-1 character.

Exported as part of the public API so other blockr packages (e.g. blockr.dm) can reuse the same column-metadata protocol.

Value

A list of column summary objects

`build_column_values` *Build full column metadata for a single column*

Description

Computes type, range (numeric), unique values, NA/empty presence, and label for one column. Called on-demand when the user selects a column in the filter block.

Usage

```
build_column_values(df, col)
```

Arguments

`df`	A data frame
`col`	Column name (character)

Details

Exported as part of the public API so other blockr packages (e.g. blockr.dm) can reuse the same column-metadata protocol.

Value

A column info object (list)

column_picker_ui *Column Picker UI / Server*

Description

Simple selectize-based column picker module used by downstream packages (e.g. blockr.extra).

Usage

```
column_picker_ui(id, label = "Columns", choices = NULL, selected = NULL)

column_picker_server(id, get_choices, initial_value = NULL)
```

Arguments

id	Module namespace id
label	Label for the input
choices	Initial choices
selected	Initial selection
get_choices	Reactive returning the available column names.
initial_value	ReactiveVal holding the initial selection.

Value

column_picker_ui returns a tagList; column_picker_server returns a reactive holding the current selection.

filter_block_dep *HTML dependency for filter block JS + CSS*

Description

Exported for reuse by blockr packages that embed the filter block's JS UI (e.g. blockr.dm's dm filter block).

Usage

```
filter_block_dep()
```

Value

An htmltools::tagList of htmlDependency objects.

make_filter_expr *Build a dplyr::filter expression from JS conditions*

Description

Exported as part of the public API so other blockr packages (e.g. blockr.dm) can reuse the same filter-condition → expression protocol.

Usage

```
make_filter_expr(conditions, operator = "&", preserve_order = FALSE)
```

Arguments

`conditions`	List of condition objects from JS. Each has a `type` ("values", "numeric", or "expr") and type-specific fields.
`operator`	Global operator: "&" or "|"
`preserve_order`	Reserved; currently unused at this layer.

Value

A language object like `dplyr::filter(.(data), ...)`

new_arrange_block *Arrange block (JS-driven)*

Description

Sort/arrange block with dynamic rows. Each row has a column picker and a direction toggle (ascending/descending). Auto-submits on any change.

Usage

```
new_arrange_block(state = list(columns = list()), ...)
```

Arguments

`state`	List with `columns` (list of objects with `column` and `direction` fields, where direction is "asc" or "desc").
`...`	Additional arguments forwarded to `blockr.core::new_block()`

Examples

```
if (interactive()) {
  library(blockr.core)
  serve(
    new_arrange_block(
      state = list(
        columns = list(
          list(column = "Sepal.Length", direction = "desc")
        )
      )
    ),
    data = list(data = iris)
  )
}
```

`new_bind_cols_block` *Bind cols block (JS-driven, variadic)*

Description

Combine multiple data frames side-by-side using dplyr::bind_cols. Minimal UI with no user-configurable parameters. Variadic block: accepts any number of data inputs via the `...args` pattern.

Usage

```
new_bind_cols_block(...)
```

Arguments

`...`	Additional arguments forwarded to `blockr.core::new_block()`

Examples

```
if (interactive()) {
  library(blockr.core)
  serve(
    new_bind_cols_block(),
    data = list(
      a = data.frame(x = 1:3),
      b = data.frame(y = 4:6)
    )
  )
}
```

new_bind_rows_block *Bind rows block (JS-driven, variadic)*

Description

Stack multiple data frames on top of each other using dplyr::bind_rows. Simple UI with an optional text input for .id column name. Variadic block: accepts any number of data inputs via the `...args` pattern.

Usage

```
new_bind_rows_block(state = list(id_name = ""), ...)
```

Arguments

`state`	List with `id_name` (string, optional .id column name).
`...`	Additional arguments forwarded to `blockr.core::new_block()`

Examples

```
if (interactive()) {
  library(blockr.core)
  serve(
    new_bind_rows_block(
      state = list(id_name = "source")
    ),
    data = list(a = iris[1:5, ], b = iris[6:10, ])
  )
}
```

new_filter_block *Filter block (JS-driven)*

Description

Unified filter block combining simple mode (column + operator + values) and expression mode (free R code) in a single JS-driven UI.

Usage

```
new_filter_block(state = list(conditions = list(), operator = "&"), ...)
```

Arguments

state	List with conditions (array of condition objects) and operator ("&" or "|"). Condition types: • values: column, values (character vector), mode ("include"/"exclude") • numeric: column, op (">", ">=", "<", "<=", "is", "is not"), value • expr: expr (R expression as string)
...	Additional arguments forwarded to blockr.core::new_block()

Examples

```
if (interactive()) {
  library(blockr.core)
  serve(
    new_filter_block(
      state = list(
        conditions = list(
          list(type = "values", column = "Species",
               values = list("setosa"), mode = "include")
        ),
        operator = "&"
      )
    ),
    data = list(data = iris)
  )
}
```

---

new_join_block *Join block (JS-driven)*

Description

Binary join block combining key-based joins (equi and non-equi) with expression mode (free R code for join conditions) in a single JS-driven UI. Takes two data inputs (x and y).

Usage

```
new_join_block(
  state = list(type = "left_join", keys = list(), exprs = list(), suffix_x = ".x",
    suffix_y = ".y"),
  ...
)
```

Arguments

state	List with: • type: join function name ("left_join", "inner_join", "right_join", "full_join", "semi_join", "anti_join") • keys: list of key objects, each with xCol, op, yCol • exprs: character vector of R expression strings for join_by() • suffix_x: suffix for x columns (default ".x") • suffix_y: suffix for y columns (default ".y")
...	Additional arguments forwarded to blockr.core::new_block()

Examples

```
if (interactive()) {
  library(blockr.core)
  serve(
    new_join_block(
      state = list(
        type = "left_join",
        keys = list(
          list(xCol = "id", op = "==", yCol = "id")
        ),
        exprs = list(),
        suffix_x = ".x",
        suffix_y = ".y"
      )
    ),
    data = list(x = iris, y = iris)
  )
}
```

---

new_mutate_block *Mutate block (JS-driven)*

Description

JS-driven mutate block with dynamic mutation rows of name + expression pairs. Each mutation defines a new or modified column using an R expression.

Usage

```
new_mutate_block(
  state = list(mutations = list(list(name = "", expr = "")), by = list()),
  ...
)
```

Arguments

state	List with mutations (array of objects with name and expr strings) and optional by (array of grouping column names). Each mutation becomes a dplyr::mutate() argument.
...	Additional arguments forwarded to blockr.core::new_block()

Examples

```
if (interactive()) {
  library(blockr.core)
  serve(
    new_mutate_block(
      state = list(
        mutations = list(
          list(name = "Sepal.Ratio", expr = "Sepal.Length / Sepal.Width")
        )
      )
    ),
    data = list(data = iris)
  )
}
```

---

new_pivot_longer_block

Pivot longer block (JS-driven)

Description

Reshape data from wide to long format using tidyr::pivot_longer. Multi-select column picker with text inputs for names_to, values_to, names_prefix, and a toggle for values_drop_na. Auto-submits on any change.

Usage

```
new_pivot_longer_block(
  state = list(cols = list(), names_to = "name", values_to = "value", values_drop_na =
    FALSE, names_prefix = ""),
  ...
)
```

Arguments

state	List with cols (character vector of columns to pivot), names_to (string), values_to (string), values_drop_na (logical), and names_prefix (string).
...	Additional arguments forwarded to blockr.core::new_block()

Examples

```
if (interactive()) {
  library(blockr.core)
  serve(
    new_pivot_longer_block(
      state = list(
        cols = list("Sepal.Length", "Sepal.Width"),
        names_to = "measurement",
        values_to = "value",
        values_drop_na = FALSE,
        names_prefix = ""
      )
    ),
    data = list(data = iris)
  )
}
```

`new_pivot_wider_block` *Pivot wider block (JS-driven)*

Description

Reshape data from long to wide format using tidyr::pivot_wider. Multi-selects for names_from, values_from, and id_cols (optional). Text inputs for values_fill, names_sep, names_prefix. Auto-submits on any change.

Usage

```
new_pivot_wider_block(
  state = list(names_from = list(), values_from = list(), id_cols = list(), values_fill =
    NULL, names_sep = "_", names_prefix = ""),
  ...
)
```

Arguments

`state`	List with `names_from` (character vector), `values_from` (character vector), `id_cols` (character vector, optional), `values_fill` (value or NULL), `names_sep` (string), and `names_prefix` (string).
`...`	Additional arguments forwarded to `blockr.core::new_block()`

Examples

```
if (interactive()) {
  library(blockr.core)
  serve(
    new_pivot_wider_block(
```

```
      state = list(
        names_from = list("Species"),
        values_from = list("Sepal.Length"),
        id_cols = list(),
        values_fill = NULL,
        names_sep = "_",
        names_prefix = ""
      )
    ),
    data = list(data = iris)
  )
}
```

`new_rename_block` *Rename block (JS-driven)*

Description

Column rename block with dynamic rows. Each row maps an old column name to a new name. Auto-submits on any change (300ms debounce for text input).

Usage

```
new_rename_block(state = list(renames = list()), ...)
```

Arguments

`state`	List with `renames` (named list where names are new column names and values are old column names).
`...`	Additional arguments forwarded to `blockr.core::new_block()`

Examples

```
if (interactive()) {
  library(blockr.core)
  serve(
    new_rename_block(
      state = list(
        renames = list(sepal_len = "Sepal.Length")
      )
    ),
    data = list(data = iris)
  )
}
```

new_select_block *Select block (JS-driven)*

Description

Column selection block with reorderable multi-select, exclude toggle, and distinct toggle. Auto-submits on any change.

Usage

```
new_select_block(
  state = list(columns = list(), exclude = FALSE, distinct = FALSE),
  ...
)
```

Arguments

`state`	List with `columns` (character vector of column names), `exclude` (logical), and `distinct` (logical).
`...`	Additional arguments forwarded to `blockr.core::new_block()`

Examples

```
if (interactive()) {
  library(blockr.core)
  serve(
    new_select_block(
      state = list(
        columns = list("Sepal.Length", "Species"),
        exclude = FALSE,
        distinct = FALSE
      )
    ),
    data = list(data = iris)
  )
}
```

new_separate_block *Separate block (JS-driven)*

Description

Split a single column into multiple columns using tidyr::separate. Single select for source column, text input for into (comma-separated new column names), text input for sep, toggle pills for remove and convert. Auto-submits on any change.

Usage

```
new_separate_block(
  state = list(col = "", into = list(), sep = "[^[:alnum:]]+", remove = TRUE, convert =
    FALSE, extra = "warn", fill = "warn"),
  ...
)
```

Arguments

state	List with `col` (string, source column), `into` (character vector of new column names), `sep` (string or regex), `remove` (logical), `convert` (logical), `extra` (string), and `fill` (string).
...	Additional arguments forwarded to `blockr.core::new_block()`

Examples

```
if (interactive()) {
  library(blockr.core)
  df <- data.frame(x = c("a-1", "b-2", "c-3"), stringsAsFactors = FALSE)
  serve(
    new_separate_block(
      state = list(
        col = "x",
        into = list("letter", "number"),
        sep = "-",
        remove = TRUE,
        convert = FALSE,
        extra = "warn",
        fill = "warn"
      )
    ),
    data = list(data = df)
  )
}
```

`new_slice_block` *Slice block (JS-driven)*

Description

Subset rows by position using dplyr::slice_* family of functions. Single select for type (head/tail/min/max/sample), number input for n, optional column selects for order_by and weight_by, toggle pills for with_ties and replace, multi-select for grouping (.by). Auto-submits on any change.

Usage

```
new_slice_block(
  state = list(type = "head", n = 5L, prop = NULL, order_by = "", with_ties = TRUE,
    weight_by = "", replace = FALSE, rows = "1:5", by = list()),
  ...
)
```

Arguments

state	List with type (string: "head", "tail", "min", "max", "sample"), n (integer), prop (numeric or NULL), order_by (string), with_ties (logical), weight_by (string), replace (logical), and by (character vector).
...	Additional arguments forwarded to blockr.core::new_block()

Examples

```
if (interactive()) {
  library(blockr.core)
  serve(
    new_slice_block(
      state = list(
        type = "head",
        n = 10L,
        prop = NULL,
        order_by = "",
        with_ties = TRUE,
        weight_by = "",
        replace = FALSE,
        by = list()
      )
    ),
    data = list(data = iris)
  )
}
```

---

new_summarize_block *Summarize block (JS-driven)*

Description

JS-driven summarize block with two row types: simple (function + column) and expression (free R code). Includes a group-by section for grouped summaries.

Usage

```
new_summarize_block(state = list(summaries = list(), by = list()), ...)
```

Arguments

state	List with summaries (array of summary objects) and by (character vector of grouping columns). Summary types: • simple: name, func (e.g. "mean"), col (column name) • expr: name, expr (R expression string)
...	Additional arguments forwarded to blockr.core::new_block()

Extending available functions

The list of available summary functions can be extended using the blockr.dplyr.summary_functions option. Set this option to a named character vector where names are display labels and values are function calls with proper namespacing:

```
options(
  blockr.dplyr.summary_functions = c(
    "extract parentheses (paren_num)" = "blockr.topline::paren_num"
  )
)
```

Examples

```
if (interactive()) {
  library(blockr.core)
  serve(
    new_summarize_block(
      state = list(
        summaries = list(
          list(type = "simple", name = "avg_sl", func = "mean",
               col = "Sepal.Length")
        ),
        by = list("Species")
      )
    ),
    data = list(data = iris)
  )
}
```

---

new_unite_block *Unite block (JS-driven)*

Description

Paste together multiple columns into one using tidyr::unite. Text input for new column name, multi-select for columns to unite, text input for separator, toggle pills for remove and na.rm. Auto-submits on any change.

Usage

```
new_unite_block(
  state = list(col = "united", cols = list(), sep = "_", remove = TRUE, na_rm = FALSE),
  ...
)
```

Arguments

state	List with col (string, new column name), cols (character vector of columns to unite), sep (string), remove (logical), and na.rm (logical).
...	Additional arguments forwarded to blockr.core::new_block()

Examples

```
if (interactive()) {
  library(blockr.core)
  serve(
    new_unite_block(
      state = list(
        col = "full_name",
        cols = list("Sepal.Length", "Sepal.Width"),
        sep = "_",
        remove = TRUE,
        na.rm = FALSE
      )
    ),
    data = list(data = iris)
  )
}
```

Index