# Package: blockr.io (via r-universe)

November 19, 2026

**Title** Interactive File Import and Export Blocks

**Version** 0.1.0.9015

**Description** Extends 'blockr.core' with interactive blocks for reading and writing data files. Supports CSV, Excel, Parquet, RDS, and other formats through a graphical interface without writing code directly. Includes file browser integration and configurable import/export options.

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

**BugReports** https://github.com/BristolMyersSquibb/blockr.io/issues

**License** GPL (>= 3)

**Depends** R (>= 4.1.0)

**Encoding** UTF-8

**Language** en-US

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**VignetteBuilder** knitr

**Remotes** BristolMyersSquibb/blockr.core, nbenn/reactives

**Imports** blockr.core (>= 0.1.4), htmltools, jsonlite, readxl, shiny, readr, rio, shinyjs, writexl, zip

**Suggests** arrow, haven, knitr, nanoparquet, reactives, rlang, rmarkdown, testthat (>= 3.0.0), withr

**Config/testthat/edition** 3

**Config/testthat/parallel** true

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

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

**Date/Publication** 2026-10-03 15:08:52 UTC

**RemoteUrl** https://github.com/BristolMyersSquibb/blockr.io

**RemoteRef** HEAD

**RemoteSha** acadbb26f1f7de9a649056d5bd44fbfe352ca477

# Contents

container_kinds . . . 2
container_list_tables . . . 3
container_read_expr . . . 3
file_category . . . 4
file_extensions . . . 4
file_policy . . . 5
format_opt . . . 6
format_options . . . 7
format_options_ui . . . 8
format_options_update . . . 8
format_options_values . . . 9
format_read_expr . . . 9
gear_band_ui . . . 10
member_options . . . 10
new_data_dir_option . . . 11
new_download_block . . . 12
new_read_block . . . 13
new_write_block . . . 15
path_input . . . 18
read_file_expr . . . 19
register_container . . . 20
register_format . . . 21
resolve_data_dir . . . 22
source_options . . . 23
write_file_expr . . . 23
write_formats . . . 24

**Index** **25**

---

`container_kinds` *What containers are registered*

---

## Description

The keys the container registry currently holds (`"directory"`, `"zip"`, ...). A block that offers container reading should gate its affordances on this, so nothing offers an action that is guaranteed to fail.

## Usage

```
container_kinds()
```

## Value

Character vector of container keys.

## container_list_tables

*Discover the tables in a container*

### Description

Resolves `path` (a directory, or a file whose extension has a container entry) and returns the table names its entry can see, without reading table data where the format allows. `container_table_info()` returns the same discovery with each table's display label (`"CSV 2.1 MB"`, `"12 rows"`, `""` when the entry has none), for blocks that render a picker.

### Usage

```
container_list_tables(path)

container_table_info(path)
```

### Arguments

| | |
|---|---|
| `path` | Character. A resolved location. |

### Value

For `container_list_tables()`, a character vector of table names. For `container_table_info()`, a data frame with columns `name` and `label`.

## container_read_expr *Read expression for a container*

### Description

Resolves `path` against the container registry and returns one unevaluated expression that evaluates to a bare named list of data frames, restricted to `tables`.

### Usage

```
container_read_expr(path, tables = NULL, ...)
```

### Arguments

| | |
|---|---|
| `path` | Character. A resolved location. |
| `tables` | Character vector of table names to read, or `NULL` for all. |
| `...` | Uniform member options, forwarded by the entry into each member's format entry, which picks the parameters it understands: `container_read_expr(dir, sep = ";")` reads every csv in the folder as semicolon-separated and leaves the other formats untouched. |

### Value

A language object that, when evaluated, produces a named list of data frames.

---

`file_category` *Categorizes a file by its extension into a broad format family that determines reader dispatch and UI adaptation.*

---

### Description

Categorizes a file by its extension into a broad format family that determines reader dispatch and UI adaptation.

### Usage

```
file_category(path)
```

### Arguments

| | |
|---|---|
| `path` | Character. File path. |

### Value

One of `"csv"`, `"excel"`, `"arrow"`, `"other"`.

---

`file_extensions` *Supported file extensions*

---

### Description

Returns a character vector of file extensions (without dots) readable in single mode: the registry's named entries plus the rio fallback's claims. Useful for sibling packages that need to filter or validate file paths before passing them to blockr.io. The set grows exactly when the registry does – a format registered via `register_format()` shows up here, and with it in the file browser filter and upload accept lists.

### Usage

```
file_extensions()
```

### Value

Character vector of file extensions (without dots)

`file_policy` *File-access verification for path-based blocks*

## Description

Read/write blocks (blockr.io `read`/`write`, blockr.dm `dm-read`/`dm-write`) resolve a user-supplied path and then read from or write to it. By default any resolvable path is allowed: `data_dir` is a UX convenience, not a security boundary, and an absolute path or a `..` traversal walks straight out of it. These helpers let a *deployment* restrict which paths a running app may touch, without each block re-implementing the check.

## Usage

```
resolve_and_check(path, mode = c("read", "write"))

within_dirs(...)
```

## Arguments

| | |
|---|---|
| `path` | A path the block is about to read from or write to. Resolved relative to `data_dir` by the calling block before it reaches here. |
| `mode` | Either `"read"` or `"write"`, selecting which verifier option applies. |
| `...` | Allowed root folders. A path is accepted if it equals one of the roots or sits inside one of them. |

## Details

The deployment sets one or both of the options `blockr.verify_read_path` / `blockr.verify_write_path` to a function of a single resolved path. A block about to use a path calls `resolve_and_check()`, which normalizes the path (resolving `.`, `..` and symlinks) and then calls the deployment's verifier on the *normalized* form. The verifier returns normally to allow the path, or `stop()`s to reject it; the block surfaces the message and does not read/write. When the option is unset there is no checking, so existing apps are unaffected.

Read and write are separate options so a deployment can make writing stricter than reading (or scope only one). Point both options at the same function to apply one rule to both.

`within_dirs()` builds the common verifier: a folder allowlist that rejects anything outside the given roots (trailing-slash- and symlink-safe). A deployment that needs more — per-extension rules, custom messages — writes its own function instead.

Normalization happens here, once, rather than inside the deployment's verifier, so the block checks exactly the path it then uses: a verifier that normalized internally could approve one form while the block touched another. The verifier therefore receives an already-resolved path and only decides yes/no.

## Value

`resolve_and_check()` returns the normalized path, or signals the verifier's error. `within_dirs()` returns a verifier function suitable for the options.

### Honest ceiling

This stops a user picking the wrong path in a read/write block. Concretely, the check resolves `..`, `.`, `//` and symlinks (via `normalizePath()`, then a lexical pass that also collapses `..` in a not-yet-existing tail that `normalizePath()` leaves alone on Linux) before testing the allowlist, so traversal and symlink-escape (including the `symlink/..` cancellation trick) are rejected. It does **not** defend against:

- **arbitrary R** in function/transform blocks (`read.csv()`, `system()`, ...) — out of reach of any in-app path check;
- a **symlink swapped between check and read** (a TOCTOU race) — symlinks are resolved at check time, not re-validated at the moment of the read.

The hard boundary for both is a read-only container filesystem except the study mount; this option is the friendly, granular layer on top.

### Examples

```
# In app.R: restrict reads to one study folder, writes to its output dir.
## Not run:
options(
  blockr.verify_read_path  = blockr.io::within_dirs("/data/study123"),
  blockr.verify_write_path = blockr.io::within_dirs("/data/study123/out")
)

## End(Not run)

# within_dirs() rejects paths outside its roots:
verify <- within_dirs(tempdir())
ok <- file.path(tempdir(), "ok.csv")
file.create(ok)                                    # normalizePath resolves it
verify(ok)                                         # allowed (returns NULL)
tryCatch(verify("/etc/passwd"), error = conditionMessage)
```

---

## `format_opt` *Declare a format's reader options*

---

### Description

The option kinds an entry may declare through `register_format()`'s `options` argument. Each returns one spec: a label, a default, and enough shape for a block to render a field and read it back.

### Usage

```
opt_choice(label, choices, default, create = FALSE)

opt_text(label, default = "", placeholder = NULL)
```

```
opt_number(label, default = 0, placeholder = NULL)

opt_flag(label, default = TRUE)
```

### Arguments

| | |
|---|---|
| label | Field label, shown to the user. |
| choices | Named character vector of choices (names are labels). |
| default | The value the reader uses when the option is unset. |
| create | For opt_choice(), whether the user may type a value that is not in choices. |
| placeholder | Hint shown in an empty field. Defaults to naming the default value. |

### Details

A value equal to the declared default is dropped when a block collects its options, so a block nobody has touched carries no options at all and serializes clean.

### Value

A blockr_format_opt object.

### See Also

register_format(), format_options()

---

## format_options *The options a format declares*

---

### Description

Looks up x (a path, or a bare extension) in the single-mode registry and returns the entry's declared option specs, or an empty list when the format declares none – which is the answer for most formats, and the reason a block should ask rather than assume.

### Usage

```
format_options(x)
```

### Arguments

| | |
|---|---|
| x | Character. A file path or a bare extension. |

### Value

Named list of format_opt specs, possibly empty.

## format_options_ui *Fields for a set of declared options*

### Description

Renders one field per spec, laid out as this package's settings bands are. Values come from a block's option list; anything absent shows its declared default.

### Usage

```
format_options_ui(specs, ns, values = list())
```

### Arguments

| | |
|---|---|
| specs | Named list of format_opt specs, e.g. from `source_options()`. |
| ns | Namespace function, `shiny::NS()` of the calling module's id. Field input ids are the spec names, namespaced. |
| values | Named list of current option values (deviations only). |

### Value

A htmltools::tag, or `NULL` when there are no specs.

## format_options_update *Push option values back into declared option fields*

### Description

For an external write: moves the fields to match `values` without re-rendering them (which would drop focus mid-edit).

### Usage

```
format_options_update(session, specs, values = list())
```

### Arguments

| | |
|---|---|
| session | The module's `session`. |
| specs | Named list of format_opt specs. |
| values | Named list of current option values. |

### Value

Invisible `NULL`.

## format_options_values

*Read declared option fields back off the inputs*

### Description

The inverse of `format_options_ui()`: collects the fields into the named list a block carries as its options, dropping every value still at its declared default so an untouched block stays empty.

### Usage

```
format_options_values(input, specs)
```

### Arguments

| | |
|---|---|
| `input` | The module's `input`. |
| `specs` | Named list of format_opt specs. |

### Value

Named list of option values; empty when nothing deviates.

## format_read_expr

*Read expression for a single file*

### Description

Resolves `path`'s extension against the registry (named entries first, then the rio fallback) and returns the entry's unevaluated read expression. This is what the read block calls when it builds its expression, and what a container's builder calls per member.

### Usage

```
format_read_expr(path, ...)
```

### Arguments

| | |
|---|---|
| `path` | Character. A resolved file path – never a user-typed one. |
| `...` | Format options forwarded to the entry (e.g. `sep`, `sheet`, `skip`). |

### Value

A language object that, when evaluated, reads the file into a data frame.

## gear_band_ui *Gear button and settings band*

### Description

The gear-and-band affordance blockr.io's blocks use for advanced settings: a small gear button, and an in-flow panel that opens beneath it (class-driven via `blockrIoGearToggle()`, so it pushes content down rather than overlaying it). Exported so sibling packages can mount the same affordance with the same styling; the required JS and CSS ride along as an html dependency.

### Usage

```
gear_band_ui(gear_id, band_id, ..., band_label = "Settings")
```

### Arguments

| | |
|---|---|
| `gear_id`, `band_id` | Element ids (namespace them with `shiny::NS()`). The two are wired: the button toggles the band and carries the `aria-controls` linkage. |
| `...` | Band content. |
| `band_label` | Accessible label for the band region. |

### Details

Lay fields out with `blockr-settings__grid` / `blockr-settings__field` divs and `blockr-label` labels, matching this package's own bands.

### Value

A `htmltools::tagList()` with the gear row and the band; place them together at the top of the block section they configure.

## member_options *The options a set of files understands*

### Description

The union of the members' declared options, which is what a container offers: uniform member options are handed to every member, so an option any member takes is settable and the members that do not take it ignore it – exactly what the emitted read expression does. Containers holding files of other formats declare `options = function(path)` in terms of this.

### Usage

```
member_options(paths)
```

### Arguments

| | |
|---|---|
| `paths` | Character vector of member paths (or names; only the extension is read). |

### Value

Named list of format_opt specs, possibly empty.

---

## `new_data_dir_option` *Data directory board option*

---

### Description

A board-level option that sets a default data directory for read and write blocks. When set, file paths entered in blocks are resolved relative to this directory.

### Usage

```
new_data_dir_option(
  value = blockr_option("data_dir", ""),
  category = "Data",
  ...
)
```

### Arguments

| | |
|---|---|
| `value` | Character. Initial directory path. Default: empty string (no data directory). |
| `category` | Character. Option category for UI grouping. |
| `...` | Forwarded to `blockr.core::new_board_option()`. |

### Value

A `board_option` object.

`new_download_block` *Download-only file export block*

## Description

A variadic block that lets the user download one or more data frames as a file in the browser. Intended as a lightweight counterpart to `new_write_block()` for the common case where no server-side save is needed.

## Usage

```
new_download_block(filename = "", format = "csv", args = list(), ...)
```

## Arguments

| | |
|---|---|
| `filename` | Character. Optional fixed filename (without extension). If empty (default), a timestamped filename is generated. |
| `format` | Character. One of the values in `write_formats()`: `"csv"`, `"excel"`, `"parquet"`, or `"feather"`. Default: `"csv"`. |
| `args` | Named list of format-specific writing parameters (same as `new_write_block()`). Only relevant values for the selected format are used. |
| `...` | Forwarded to `blockr.core::new_transform_block()`. |

## Details

Multi-input behavior matches `new_write_block()`: multiple inputs produce a multi-sheet Excel file for `format = "excel"`, or a ZIP archive for CSV, Parquet, and Feather.

Adding a new format (e.g. SAS) only requires extending `write_formats()`, `format_extension()`, and the dispatch in `write_expr()` - both `new_write_block()` and `new_download_block()` pick it up automatically.

## Value

A blockr transform block exposing a download button.

## Examples

```
if (interactive()) {
  library(blockr.core)
  serve(new_download_block())
}
```

`new_read_block` *Unified file reading block*

## Description

A single block for reading files in various formats with smart UI that adapts based on detected file type. Supports "From Browser" (upload) and "Location" (path/URL input) modes with persistent storage for uploaded files.

## Usage

```
new_read_block(
  path = character(),
  source = "upload",
  combine = "auto",
  args = list(),
  ...
)
```

## Arguments

| | |
|---|---|
| `path` | Character vector of file paths to pre-load. Accepts local paths and URLs. When provided, automatically switches to "path" mode regardless of the source parameter. |
| `source` | Either "upload" for file upload widget or "path" for path/URL input. Default: "upload". Automatically set based on path parameter. |
| `combine` | Strategy for combining multiple files: "auto", "rbind", "cbind", "first" |
| `args` | Named list of format-specific reading parameters. Only specify values that differ from defaults. Which parameters a file accepts is declared by its format's registry entry, so `format_options()` is the live answer and the settings band is generated from it. As shipped:<br>• **For CSV files:** `sep` (default: ","), `quote` (default: '""'), `encoding` (default: "UTF-8"), `skip` (default: 0), `n_max` (default: Inf), `col_names` (default: TRUE)<br>• **For Excel files:** `sheet` (default: NULL), `range` (default: NULL), `skip` (default: 0), `n_max` (default: Inf), `col_names` (default: TRUE)<br>• **Everything else:** nothing to set, and no settings affordance shown |
| `...` | Forwarded to `blockr.core::new_data_block()` |

## Details

**File Handling Modes:**

The block supports two modes:

**From Browser mode** (upload):

- User uploads files from their computer via the browser

- Files are copied to persistent storage directory (upload_path)
- State stores permanent file paths
- Works across R sessions with state restoration

**Location mode** (path):

- User enters a file path or URL in a text input with autocomplete
- The path is committed (and the file read) on Enter, blur, or a dropdown selection — never while typing; an "Enter" chip shows while the typed path is not yet applied
- For server paths: reads directly from original location
- For URLs: downloads to a temporary file each time
- When a board-level data directory is set, paths are resolved relative to it

**Smart Adaptive UI:**

After file selection the block asks the registry what this format allows (`format_options()`) and generates the settings band from the answer:

- **CSV/TSV:** Delimiter, quote character, encoding, skip, max rows, header
- **Excel:** Sheet, cell range, skip, max rows, header
- **Other formats:** nothing to set, so no gear at all

A format registered by another package (`register_format()`) declares its own options and gets fields here without this block changing.

**Multi-file Support:**

When multiple files are selected:

- **"auto"**: Attempts rbind, falls back to first file if incompatible
- **"rbind"**: Row-binds files (requires same columns)
- **"cbind"**: Column-binds files (requires same row count)
- **"first"**: Uses only the first file

## Value

A blockr data block that reads file(s) and returns a data.frame.

## External control

`path`, `source`, `combine` and `args` are externally controllable (see `blockr.core::external_ctrl_vars()`), so a board update, an assistant or a parent app can retarget the block with a `mod` delta instead of replacing it. This holds because the block's expression is a pure function of that state: writing `path` moves the read, and the path field, the type badge and the settings band follow. A path that does not resolve is reported on the badge and as a block error rather than failing the constructor, so a board restores even when its data has not landed yet.

## Configuration

The following settings are retrieved from options and not stored in block state:

- **upload_path**: Directory for persistent file storage. Set via `options(blockr.upload_path = "/path")` or environment variable `BLOCKR_UPLOAD_PATH`. Default: `tools::R_user_dir("blockr", "data")`

### Examples

```
# Create a read block for a CSV file
csv_file <- tempfile(fileext = ".csv")
write.csv(mtcars[1:5, ], csv_file, row.names = FALSE)
block <- new_read_block(path = csv_file)
block

# With custom CSV parameters
block <- new_read_block(
  path = csv_file,
  args = list(n_max = 3)
)

if (interactive()) {
  # Launch interactive app
  serve(new_read_block())
}
```

---

## new_write_block *Unified file writing block*

---

### Description

A variadic block for writing dataframes to files in various formats. Accepts multiple input dataframes and handles single files, multi-sheet Excel, or ZIP archives depending on format and number of inputs.

### Usage

```
new_write_block(
  directory = "",
  filename = "",
  format = "csv",
  auto_write = FALSE,
  args = list(),
  mode = NULL,
  ...
)
```

### Arguments

| | |
|---|---|
| `directory` | Character. Default directory for file output. When non-empty, enables server-side writing. Can be configured via `options(blockr.write_dir = "/path")` or environment variable `BLOCKR_WRITE_DIR`. Default: `""` (empty – download-only until user sets a path). |
| `filename` | Character. Optional fixed filename (without extension).<br>• **If provided**: Writes to the same file path on every save (overwrite) |

- **If empty** (default): Manual saves and downloads generate a timestamped filename (e.g., `data_20250127_143022.csv`); auto-write uses a fixed `data.{ext}` file so repeated writes overwrite instead of littering the directory

`format` Character. Output format: "csv", "excel", "parquet", or "feather". Default: "csv"

`auto_write` Logical. When TRUE, automatically writes files when data changes (requires a non-empty directory). When FALSE (default), the user must click "Save to Server", and each click writes exactly once.

`args` Named list of format-specific writing parameters. Only specify values that differ from defaults. Available parameters:

- **For CSV files:** `sep` (default: ","), `quote` (default: TRUE), `na` (default: "")
- **For Excel/Arrow:** Minimal options needed (handled by underlying packages)

`mode` [**Deprecated**] Previously selected between "browse" and "download" tabs. Now ignored – both download and server-save are always available. Kept for backwards compatibility; emits a deprecation warning when non-NULL.

`...` Forwarded to `blockr.core::new_transform_block()`

## Details

### Variadic Inputs:

This block accepts multiple dataframe inputs (1 or more) similar to `bind_rows_block`. Inputs can be numbered ("1", "2", "3") or named ("sales_data", "inventory"). Input names are used for sheet names (Excel) or filenames (multi-file ZIP).

### File Output Behavior:

**Single input:**

- Writes single file in specified format
- Filename: `{filename}.{ext}` or `data_{timestamp}.{ext}`

**Multiple inputs + Excel:**

- Single Excel file with multiple sheets
- Sheet names derived from input names

**Multiple inputs + CSV/Arrow:**

- Single ZIP file containing individual files
- Each file named from input names

### Filename Behavior:

**Fixed filename** (`filename = "output"`):

- Reproducible path: always writes to `{directory}/output.{ext}`
- Overwrites the file on every save (every data change with auto-write)
- Ideal for automated pipelines

**Empty filename** (`filename = ""`):

- Manual saves and downloads: unique files `{directory}/data_YYYYMMDD_HHMMSS.{ext}` – preserves history, prevents accidental overwrites
- Auto-write: fixed `{directory}/data.{ext}`, overwritten on each change – a timestamp would create one file per upstream change

**Download vs Server Save:**

Both options are always available in a flat layout (no tabs):

**Download to Browser:**

- Always available via the download button
- Triggers a download to your browser's download folder

**Save to Server:**

- Active when a server directory path is set (non-empty)
- User types a directory path (committed with Enter, blur, or a dropdown selection – an "Enter" chip shows while the typed path is not yet applied) in the path input
- The target directory is created at write time if missing
- In manual mode each "Save to Server" click writes exactly once; later data changes never rewrite the file
- Files persist on server; when running locally, this is your computer's file system

**Pipeline Behavior:**

The block passes its first input through unchanged. Only with `auto_write = TRUE` does the block expression itself contain the write (so exported code reproduces the auto-write); manual saves and downloads happen in their handlers and keep the expression a pure passthrough.

## Value

A blockr transform block that writes dataframes to files

## Examples

```
# Create a write block for CSV output
block <- new_write_block(
  directory = tempdir(),
  filename = "output",
  format = "csv"
)
block

# Write block for Excel with auto-timestamp
block <- new_write_block(
  directory = tempdir(),
  filename = "",
  format = "excel"
)

if (interactive()) {
```

```
  # Launch interactive app
  serve(new_write_block())
}
```

---

`path_input` *Path input widget*

---

## Description

A Shiny module that provides a text input with server-side file/directory autocomplete. Used by the read and write blocks to replace shinyFiles browser widgets.

## Usage

```
path_input_ui(
  id,
  prefix = NULL,
  upload_id = NULL,
  placeholder = NULL,
  required = FALSE
)

path_input_server(
  id,
  data_dir = reactive(""),
  mode = c("file", "directory"),
  extensions = NULL,
  policy = NULL,
  value = NULL
)
```

## Arguments

| | |
|---|---|
| `id` | Module namespace ID. |
| `prefix` | Optional initial prefix text shown before the input (typically the data directory path). |
| `upload_id` | Optional ID of a hidden Shiny `fileInput` to wire up for upload-icon click and drag-and-drop. When non-NULL, an upload icon button is rendered inside the input field and the container gets a `data-upload-target` attribute pointing to this ID. |
| `placeholder` | Placeholder text for the input. Defaults to an upload-aware file hint; pass e.g. "Enter directory path..." for directory-mode inputs. |
| `required` | Whether the field must be filled. When `TRUE`, an empty field carries a soft amber "needs a value" cue via the canonical `.blockr-field--required-empty` class (mirroring blockr.viz's required-empty mapping affordance) that clears once a value is entered. |

| | |
|---|---|
| `data_dir` | Reactive returning the current data directory path (from board options). Empty string means no data directory. |
| `mode` | Either `"file"` or `"directory"`. Controls which entries are selectable in the autocomplete dropdown. |
| `extensions` | Optional character vector of file extensions (without dots) to show in autocomplete. Defaults to `NULL`, which shows all rio-supported formats. Use e.g. `"rtf"` to restrict to RTF files only. |
| `policy` | Which deployment file-access verifier applies to the directory-listing endpoint: `"read"` or `"write"` (see file_policy). Defaults to `"read"` for file mode and `"write"` for directory mode. |
| `value` | Optional reactive giving the path the field should show. Supply it and the module keeps the widget in step with it – including after a dock panel mounts, which is the case a caller pushing on its own cannot get right (see the note in the body). |

### Value

`path_input_ui()` returns a `tagList` with the widget HTML. `path_input_server()` returns a `reactive` containing the committed path text value.

### Commit model

Typing never commits: keystrokes only drive the autocomplete dropdown. The reactive value returned by `path_input_server()` updates when the user *commits* — by pressing Enter, leaving the field (blur), or selecting a dropdown entry. While the typed text differs from the committed value, the field shows an "Enter " chip; committing collapses it to a faded check mark. This follows the blockr design-system text-commit convention (decided 2026-07-02) and keeps half-typed paths from reaching the pipeline.

---

## `read_file_expr` *Create a read expression for a single file*

---

### Description

Convenience wrapper that detects the file category from the path and returns an unevaluated R expression for reading the file. Useful for sibling packages that construct pipelines programmatically.

### Usage

```
read_file_expr(path, ...)
```

### Arguments

| | |
|---|---|
| `path` | Character. Path to a single file. |
| `...` | Additional parameters forwarded to the reader (e.g. `sep`, `sheet`, `skip`). |

### Value

A language object (unevaluated call) that, when evaluated, reads the file into a data frame.

---

`register_container` *Register a container*

---

### Description

A container is a location that holds several tables: a directory, a zip, a workbook read sheet by sheet. A container entry produces a bare named list of data frames – no class. Whatever the consuming block wants the list to become (a dm, via `dm::new_dm()`) is its own wrap, made outside this package.

### Usage

```
register_container(opens, list_tables, read, options = NULL)
```

### Arguments

| | |
|---|---|
| `opens` | Character vector of file extensions (without dots), or the string `"directory"`. Directories are not a special case in the lookup, only a key that is not an extension. |
| `list_tables` | `function(path)` returning the table names available at `path`, cheaply: discovery must not read table data where the format allows it (a zip's central directory, a workbook's sheet index). Formats whose only index is the data itself (an `.rds`) may read; they are local files read once at configure time. Return a character vector, or a data frame with a `name` column and an optional `label` column carrying a short display string per table (`"CSV 2.1 MB"`). |
| `read` | `function(path, tables, ...)` returning one unevaluated expression that evaluates to a named list of data frames, restricted to `tables` (`NULL` means all). The expression may compose member reads via `format_read_expr()` at build time, and may only call reader packages. `...` carries uniform member options: an entry that holds files of other formats should forward it into each member's `format_read_expr()` call, where the member's entry picks the parameters it understands (so `sep = ";"` reaches every csv in a folder and everything else ignores it). |
| `options` | What the container accepts through `read`'s `...`: a named list of format_opt specs, or a `function(path)` returning one, since a container that holds files of other formats cannot know its options until it is pointed somewhere (`member_options()` over its members is the usual answer). `NULL` for a container with nothing to tune. |

### Value

Invisible `NULL`, called for its side effect.

### See Also

`register_format()`, `container_list_tables()`, `container_read_expr()`, format_opt

---

`register_format` *Register a file format*

---

### Description

Teaches blockr.io a file format at load time. The format then works everywhere blockr.io's format knowledge is consulted: the read block's dispatch, `file_extensions()` (and with it the file browser filter and upload accept lists), and any package that asks the registry rather than carrying its own extension list.

### Usage

```
register_format(extensions, read, options = NULL, url_ok = FALSE)
```

### Arguments

| | |
|---|---|
| `extensions` | Character vector of file extensions (without dots). |
| `read` | `function(path, ...)` returning an unevaluated expression that reads `path` into a data frame. The expression may only call the reader package (e.g. `bquote(your.pkg::read_special(.(path)))`), never blockr. `...` carries format options forwarded by the calling block. `path` is a resolved location and may be a language object (a container splicing member reads against a run-time prefix), so splice it, do not inspect it. |
| `options` | Named list of format_opt specs declaring what `read` accepts through `...`, or `NULL` for a format with nothing to tune. Blocks generate their settings fields from this and show no settings affordance at all when it is empty, so a format declaring options gets them offered everywhere without any block changing. |
| `url_ok` | Does this format's reader accept an `https://` location where it accepts a path? `readr`'s delimited readers do; `readxl` and `arrow` do not. Blocks reading a URL download it to a temp file so that detection, size checks and a failed fetch all behave, and a format that declares `url_ok = TRUE` gets the URL back in the EXPRESSION it emits, so exported code names the source instead of a temp path that exists on one machine for one session. Default `FALSE`, which is the safe answer for a reader that cannot open a connection. |

### Details

Call from your package's `.onLoad()`. The registering namespace is recorded automatically for collision messages. Registration never throws; two packages claiming the same extension and mode error at first lookup, naming both.

### Value

Invisible `NULL`, called for its side effect.

### See Also

`register_container()`, `format_read_expr()`, format_opt

---

`resolve_data_dir` *Resolve paths against the board's data directory*

---

### Description

Relative paths are taken to be relative to the data directory; absolute ones and URLs are left alone. Vectorized, so a multi-file read resolves in one call.

### Usage

```
resolve_data_dir(paths, data_dir = "")
```

### Arguments

| | |
|---|---|
| `paths` | Character vector of paths, each absolute, relative or a URL. |
| `data_dir` | The board's data directory. `""` (the default) leaves every path alone, which is what a board without one wants. |

### Details

Every block that takes a path has to answer the same question – is this absolute, or is it relative to the board's data directory – and each one that answers it in its own words is a block that can resolve differently from the path widget beside it, reading one file while showing another. Exported so there is one answer.

### Value

`paths`, unnamed, with the relative ones prefixed by `data_dir`.

## source_options *The options a source offers*

### Description

What can be set when reading `path` in the given mode: the format's own options for a single file, and for a container the options its members understand (a folder of CSVs offers the csv options; a folder of parquet offers none). This is what a block gates its settings affordance on – no options, no gear – and what it generates the fields from.

### Usage

```
source_options(path, mode = c("single", "container"))
```

### Arguments

| | |
|---|---|
| `path` | Character. A resolved location. |
| `mode` | `"single"` (one file, one data frame) or `"container"` (a location holding several tables). |

### Value

Named list of format_opt specs, possibly empty.

## write_file_expr *Create a write expression for a single file*

### Description

Convenience wrapper that detects the file format from the path extension and returns an unevaluated R expression for writing the data. Useful for sibling packages that construct pipelines programmatically (e.g. DM blocks).

### Usage

```
write_file_expr(data, path, ...)
```

### Arguments

| | |
|---|---|
| `data` | Character. Name of the data object to write. |
| `path` | Character. Output file path (extension determines format). |
| `...` | Additional parameters forwarded to the writer (e.g. `sep`, `quote`, `na` for CSV files). |

### Value

A language object (unevaluated call) that, when evaluated, writes the data frame to the file.

### Examples

```
write_file_expr("mtcars", "/tmp/cars.csv")
write_file_expr("iris", "/tmp/flowers.xlsx")
write_file_expr("df", "/tmp/data.parquet")
```

---

## write_formats *Supported write formats*

---

### Description

Returns a named character vector of supported output formats for writing data frames. Names are display labels, values are format identifiers. Used to populate format dropdowns in the write block UI.

### Usage

```
write_formats()
```

### Value

A named character vector.

# Index


blockr.core::external_ctrl_vars(), *14*
blockr.core::new_board_option(), *11*
blockr.core::new_data_block(), *13*
blockr.core::new_transform_block(), *12*, *16*

container_kinds, 2
container_list_tables, 3
container_list_tables(), *21*
container_read_expr, 3
container_read_expr(), *21*
container_table_info *(container_list_tables)*, 3
container_table_info(), *3*

file_category, 4
file_extensions, 4
file_extensions(), *21*
file_policy, 5, *19*
format_opt, 6, *7–9*, *11*, *20–23*
format_options, 7
format_options(), *7*, *13*, *14*
format_options_ui, 8
format_options_ui(), *9*
format_options_update, 8
format_options_values, 9
format_read_expr, 9
format_read_expr(), *20*, *22*

gear_band_ui, 10

htmltools::tag, *8*
htmltools::tagList(), *10*

member_options, 10
member_options(), *20*

new_data_dir_option, 11
new_download_block, 12
new_download_block(), *12*
new_read_block, 13
new_write_block, 15
new_write_block(), *12*
normalizePath(), *6*

opt_choice *(format_opt)*, 6
opt_flag *(format_opt)*, 6
opt_number *(format_opt)*, 6
opt_text *(format_opt)*, 6

path_input, 18
path_input_server *(path_input)*, 18
path_input_ui *(path_input)*, 18

read_file_expr, 19
register_container, 20
register_container(), *22*
register_format, 21
register_format(), *4*, *6*, *7*, *14*, *21*
resolve_and_check *(file_policy)*, 5
resolve_and_check(), *5*
resolve_data_dir, 22

shiny::NS(), *8*, *10*
source_options, 23
source_options(), *8*

within_dirs *(file_policy)*, 5
within_dirs(), *5*
write_file_expr, 23
write_formats, 24
write_formats(), *12*