Package: gdalviz (via r-universe)

July 12, 2026

Title Visualize Modern GDAL Pipelines

Version 0.1.0

Author Jimmy Briggs [aut, cre]

Maintainer Jimmy Briggs <jimmy.briggs@jimbrig.com>

Description Produces interactive or static visualizations of modern GDAL CLI pipelines.

License MIT + file LICENSE

URL http://docs.jimbrig.com/gdalviz/, https://github.com/jimbrig/gdalviz

Imports cli, crayon, DiagrammeR, g6R, htmlwidgets, jsonlite, processx, rlang, tibble, utils

Suggests knitr, quarto, spelling, testthat (>= 3.0.0)

VignetteBuilder quarto

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 8.0.0

BugReports https://github.com/jimbrig/gdalviz/issues

Config/pak/sysreqs cmake libglpk-dev make libicu-dev libuv1-dev libxml2-dev libx11-dev zlib1g-dev

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

Date/Publication 2026-07-12 02:18:07 UTC

RemoteUrl https://github.com/jimbrig/gdalviz

RemoteRef HEAD

RemoteSha 439104e65f19b51793563e0bdb96e60a05c997b1

Contents

`as_gdalg` *Convert a pipeline to a GDALG specification list*

Description

Convert a pipeline to a GDALG specification list

Usage

```
as_gdalg(x, relative_paths = TRUE, gdal_version = NULL)
```

Arguments

`x`	A `gdalviz_pipeline`, or a string/path accepted by `read_gdalg()`.
`relative_paths`	Logical for the `relative_paths_relative_to_this_file` member. Defaults to `TRUE`.
`gdal_version`	Optional GDAL version string to record.

Value

A named list with `type`, `command_line`, and optional members, suitable for serialization to a `*.gdalg.json` file.

contract_step *Look up a single pipeline step in the contract*

Description

Look up a single pipeline step in the contract

Usage

```
contract_step(command, contract = gdalviz_contract())
```

Arguments

command	Step command name (e.g. "reproject").
contract	A gdalviz_contract. Defaults to the bundled contract.

Value

The step definition list, or NULL if the command is unknown.

gdalviz_contract *GDAL vector pipeline contract registry*

Description

Loads the bundled gdal vector pipeline --json-usage snapshot and turns it into a lookup of pipeline steps and their arguments. This is the single source of truth for which steps and arguments are valid, their types, choices, and documentation URLs.

Usage

```
gdalviz_contract(path = NULL, refresh = FALSE)
```

Arguments

path	Optional path to a --json-usage JSON snapshot. Defaults to the snapshot bundled with the package.
refresh	If TRUE, bypass the cache and reload.

Value

A gdalviz_contract object: a named list of step definitions.

gdalviz_refresh_contract

Regenerate the GDAL pipeline contract snapshot from the installed GDAL

Description

Runs `gdal <type> pipeline --json-usage` against the GDAL CLI on the user's `PATH` (or an explicit binary) and writes the result as a new contract snapshot, so the contract registry matches the locally installed GDAL version instead of the snapshot bundled with the package.

Usage

```
gdalviz_refresh_contract(
  path = NULL,
  type = c("vector", "raster"),
  gdal = NULL
)
```

Arguments

`path`	Destination for the snapshot JSON. Defaults to the bundled snapshot location, replacing it for the current installation.
`type`	Pipeline type (`"vector"` or `"raster"`).
`gdal`	Path to the `gdal` binary. Defaults to `gdal` on the `PATH`.

Value

The refreshed `gdalviz_contract`, invisibly.

lint_pipeline

Lint a GDAL pipeline against the GDAL contract

Description

Produces a structured issue table for unknown steps, unknown arguments, missing required arguments, and value/type mismatches.

Usage

```
lint_pipeline(x, contract = gdalviz_contract())
```

Arguments

`x`	A `gdalviz_pipeline`, or a string/path accepted by `read_gdalg()`.
`contract`	A `gdalviz_contract`.

Value

A tibble with one row per lint issue.

`parse_pipeline` *Parse a GDAL vector pipeline into a structured object*

Description

Accepts a raw pipeline command line (with or without the `gdal [vector|raster] pipeline` prefix) and parses it into an ordered list of steps, resolving nested pipelines (`[ ... ]`) and associating arguments with their values using the GDAL contract to disambiguate boolean flags.

Usage

```
parse_pipeline(x, contract = gdalviz_contract())
```

Arguments

`x`	A pipeline command-line string.
`contract`	A `gdalviz_contract` used to disambiguate argument types.

Value

A `gdalviz_pipeline` object.

`parse_script` *Parse a shell or powershell script into a pipeline*

Description

Normalizes a `gdal ... pipeline` invocation written as a shell or powershell script – joining line continuations, stripping comments, and collapsing here-strings / heredocs into single tokens – then parses it with `parse_pipeline()`.

Usage

```
parse_script(
  text,
  shell = c("bash", "powershell"),
  contract = gdalviz_contract()
)
```

Arguments

text	The script text (a single string or a character vector of lines).
shell	Source shell: "bash" (default) or "powershell".
contract	A gdalviz_contract.

Value

A gdalviz_pipeline.

---

pipeline_dot *Generate the DOT specification for a pipeline graph*

Description

Generate the DOT specification for a pipeline graph

Usage

```
pipeline_dot(graph, direction = c("TB", "LR"), theme = c("light", "dark"))
```

Arguments

graph	A gdalviz_graph from pipeline_graph().
direction	Layout direction: "TB" (vertical, default) or "LR".
theme	"light" or "dark".

Value

A length-one character string of DOT.

---

pipeline_graph *Build a renderer-agnostic graph model from a parsed pipeline*

Description

Converts a parse_pipeline() result into a graph of nodes and edges, classifying each step, rendering its arguments as code, generating a plain-language description, and propagating the feature-stream state (CRS, geometry type, field schema, validity, ordering) along the pipeline.

Usage

```
pipeline_graph(x, contract = gdalviz_contract(), merge_repeated = TRUE)
```

Arguments

x	A gdalviz_pipeline (from parse_pipeline()) or a string/path accepted by read_gdalg().
contract	A gdalviz_contract.
merge_repeated	Merge runs of 3+ consecutive identical commands (e.g. the one-field-at-a-time set-field-type chains needed for schema overrides) into a single stacked node. Defaults to TRUE.

Value

A gdalviz_graph: a list with nodes and edges tibbles.

---

pipelineFlowOutput *Shiny bindings for pipeline_flow widgets*

Description

Output and render functions for using render_reactflow() widgets within Shiny applications.

Usage

```
pipelineFlowOutput(outputId, width = "100%", height = "560px")

renderPipelineFlow(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

outputId	Output variable to read from.
width, height	CSS dimensions for the widget container.
expr	An expression that returns a pipeline_flow widget.
env	The environment in which to evaluate expr.
quoted	Whether expr is a quoted expression.

Value

pipelineFlowOutput() returns a Shiny output element; renderPipelineFlow() returns a Shiny render function.

read_gdalg *Read a GDAL pipeline from a GDALG file or raw string*

Description

Accepts a path to a GDALG JSON file (`{"type":"gdal_streamed_alg", "command_line": "..."}`), a path to a raw pipeline text file, or a raw pipeline string, and returns a parsed `gdalviz_pipeline`.

Usage

```
read_gdalg(x, contract = gdalviz_contract())
```

Arguments

`x`	A path to a GDALG/JSON/text file, or a raw pipeline string.
`contract`	A `gdalviz_contract`.

Value

A `gdalviz_pipeline`.

read_script *Read a shell or powershell script file into a pipeline*

Description

Read a shell or powershell script file into a pipeline

Usage

```
read_script(path, shell = NULL, contract = gdalviz_contract())
```

Arguments

`path`	Path to a `.sh` / `.ps1` script.
`shell`	Source shell: `"bash"` (default) or `"powershell"`.
`contract`	A `gdalviz_contract`.

Value

A `gdalviz_pipeline`.

render_command_line *Render a pipeline as a canonical GDALG command line*

Description

Serializes a parsed `parse_pipeline()` result back into a single-line `command_line` string, using the same minimal quoting that GDAL itself uses when it writes `*.gdalg.json` files. The result round-trips: parsing the output again yields an equivalent pipeline.

Usage

```
render_command_line(x, prog = TRUE, type = NULL)
```

Arguments

`x`	A `gdalviz_pipeline`, or a string/path accepted by `read_gdalg()`.
`prog`	Logical or `NULL`. Whether to emit the leading `gdal <type> pipeline` program prefix. Defaults to `TRUE`.
`type`	Pipeline type (`"vector"` or `"raster"`). Defaults to the type detected during parsing, falling back to `"vector"`.

Value

A length-one character vector.

render_diagrammer *Render a pipeline graph as a Graphviz diagram (static-capable)*

Description

Builds a DOT specification with card-style HTML nodes (verb header, code body, plain-language description), category colors, and state-annotated edges, rendered via DiagrammeR. Suitable for static export to SVG/PNG.

Usage

```
render_diagrammer(graph, direction = c("TB", "LR"), theme = c("light", "dark"))
```

Arguments

`graph`	A `gdalviz_graph` from `pipeline_graph()`.
`direction`	Layout direction: `"TB"` (vertical, default) or `"LR"`.
`theme`	`"light"` or `"dark"`.

Value

A DiagrammeR `grViz` htmlwidget.

render_g6 *Render a pipeline graph as an interactive g6 (AntV G6) widget*

Description

Produces a modern, interactive node-link diagram using a dagre (hierarchical) layout. Nodes are card-style, colored by category, labelled with the step verb and a plain-language description; edges carry state-change badges.

Usage

```
render_g6(
  graph,
  direction = c("TB", "LR", "BT", "RL"),
  theme = c("light", "dark"),
  height = NULL,
  width = "100%"
)
```

Arguments

graph	A gdalviz_graph from pipeline_graph().
direction	Layout direction: "TB" (default), "LR", "BT", "RL".
theme	"light" or "dark".
height, width	Widget dimensions.

Value

A g6 htmlwidget.

render_reactflow *Render a pipeline graph as an interactive React Flow widget*

Description

Produces a modern, interactive dataflow diagram of a GDAL pipeline using React Flow (xyflow). Steps render as card-style nodes colored by category with their arguments inline; edges carry state-change badges (CRS, geometry type, field count); tee branches render as dashed side flows. Clicking a node opens an inspector panel with the full argument list, the propagated stream state, and a link to the GDAL documentation for that step.

Usage

```
render_reactflow(
  graph,
  direction = c("TB", "LR", "BT", "RL"),
  theme = c("light", "dark"),
  minimap = TRUE,
  controls = TRUE,
  legend = TRUE,
  draggable = TRUE,
  width = NULL,
  height = NULL,
  elementId = NULL
)
```

Arguments

graph	A gdalviz_graph from pipeline_graph(), or a pipeline/string/path accepted by it.
direction	Layout direction: "TB" (default), "LR", "BT", "RL".
theme	"light" (default) or "dark".
minimap	Show a minimap overview. Defaults to TRUE.
controls	Show zoom/fit controls. Defaults to TRUE.
legend	Show the category legend. Defaults to TRUE.
draggable	Allow nodes to be repositioned by dragging. Defaults to TRUE.
width, height	Widget dimensions passed to htmlwidgets::createWidget().
elementId	Optional explicit element id for the widget container.

Details

The JavaScript bundle is built from srcjs/ (see srcjs/README.md) and shipped with the package, so no node toolchain is needed at run time.

Value

A pipeline_flow htmlwidget.

Examples

```
## Not run:
gdalg <- system.file(
  "extdata", "pipelines", "tiger_states.gdalg.json",
  package = "gdalviz"
)
pipeline_graph(gdalg) |> render_reactflow(theme = "dark")

## End(Not run)
```

render_script *Render a pipeline as a formatted shell script*

Description

Produces an indented, line-continued `gdal <type> pipeline` invocation for either `bash`/`sh` or `powershell`, with one `! step` per line and shell-appropriate quoting. Optionally reflows large SQL into a heredoc (bash) or here-string (powershell) for readability.

Usage

```
render_script(
  x,
  shell = c("bash", "powershell"),
  indent = "  ",
  type = NULL,
  globals_per_line = 3L,
  sql = c("inline", "block")
)
```

Arguments

`x`	A `gdalviz_pipeline`, or a string/path accepted by `read_gdalg()`.
`shell`	Target shell: `"bash"` (default) or `"powershell"`.
`indent`	Indentation unit for steps. Defaults to two spaces.
`type`	Pipeline type (`"vector"` or `"raster"`).
`globals_per_line`	Number of global options to place per continuation line. Defaults to `3`.
`sql`	Either `"inline"` (default, single quoted token) or `"block"` (multiline heredoc / here-string for `--sql` values).

Value

A length-one character vector containing the full script.

validate_pipeline *Validate a GDAL pipeline against the GDAL contract*

Description

Validates a parsed pipeline and returns a structured validation object. By default it errors when validation fails.

Usage

```
validate_pipeline(x, contract = gdalviz_contract(), strict = TRUE)
```

Arguments

x	A gdalviz_pipeline, or a string/path accepted by read_gdalg().
contract	A gdalviz_contract.
strict	If TRUE (default), abort when validation contains errors.

Value

A gdalviz_validation object with valid and issues.

---

write_gdalg *Write a pipeline to a GDALG JSON file*

Description

Write a pipeline to a GDALG JSON file

Usage

```
write_gdalg(x, path, relative_paths = TRUE, gdal_version = NULL, pretty = TRUE)
```

Arguments

x	A gdalviz_pipeline, or a string/path accepted by read_gdalg().
path	Output path (conventionally ending in .gdalg.json).
relative_paths	Logical for the relative_paths_relative_to_this_file member. Defaults to TRUE.
gdal_version	Optional GDAL version string to record.
pretty	Whether to pretty-print the JSON. Defaults to TRUE.

Value

The path, invisibly.

Index