

Package: dbVisuals (via r-universe)

October 28, 2026

Type Package

Title Database-Powered Visualizations for Tabular Data

Version 0.1.0

Description Interactive, linked visualizations of tabular data held in 'DuckDB' databases and data files, including spatial data such as 'GiottoDB' spatial omics datasets. A browser workspace built on 'SQL Rooms' creates and links maps, tables and charts without code, and an 'htmlwidgets' viewer combines 'Mosaic' views and 'deck.gl' layers over one 'DuckDB-WASM' instance and one selection, so a brush in any view filters the others. Results of queries on a 'DuckDB' connection reach the browser as 'Arrow' data written by 'DuckDB', without materialising the rows in R. Complements the 'rDeckgl' and 'rMosaic' packages. Both browser components load 'DuckDB-WASM' from a public content delivery network, so using them requires internet access.

License MIT + file LICENSE

Copyright See file inst/COPYRIGHTS for the bundled JavaScript libraries.

URL <https://github.com/dbverse-org/dbVisuals>,
<https://dbverse-org.github.io/dbVisuals/>

BugReports <https://github.com/dbverse-org/dbVisuals/issues>

Encoding UTF-8

Depends R (>= 4.1.0)

Imports callr (>= 3.7.0), DBI (>= 1.1.0), httpuv, jsonlite (>= 1.8.0), openssl, base64enc, htmlwidgets (>= 1.5.4), htmltools (>= 0.5.0), stats, utils

Suggests duckdb (>= 0.8.0), dbplyr, dplyr, arrow (>= 12.0.0), shiny, nanoarrow, adbcdrivermanager, rDeckgl, rMosaic, remotes, testthat (>= 3.0.0), withr, knitr, rmarkdown

Config/Needs/check testthat, knitr, rmarkdown, duckdb, dbplyr, dplyr, arrow

Config/Needs/check-light testthat, knitr, rmarkdown
Config/Needs/website knitr, rmarkdown
VignetteBuilder knitr
Config/roxygen2/version 8.1.0
Config/pak/sysreqs cmake make libuv1-dev libssl-dev zlib1g-dev
Repository <https://dbverse-org.r-universe.dev>
Date/Publication 2026-10-28 00:50:27 UTC
RemoteUrl <https://github.com/dbverse-org/dbVisuals>
RemoteRef HEAD
RemoteSha d831000ab69ec67647d819c33231915161a67031
RemoteSubdir r-package

Contents

dbvisuals_export_htmlwidget	2
install_dbvisuals_deps	4
launchBrowser	5
mgl_data	6
mgl_deckgl	7
mgl_spec	8
mosaic_gl	9
mosaic_glOutput	11
renderMosaic_gl	12
sqlrooms_export_table	13
sqlrooms_register	13
stop_sqlrooms	14
Index	16

dbvisuals_export_htmlwidget

Export a dbVisuals spec as an htmlwidget

Description

Renders a Mosaic or deck.gl specification through rMosaic or rDeckgl and saves the resulting htmlwidget to disk. This is intended for durable exports from dbVisuals specs; the live dbVisuals workspace continues to use the SQLRooms React runtime.

Usage

```
dbvisuals_export_htmlwidget(
  spec,
  file,
  engine = "auto",
  data = NULL,
  data_sql = NULL,
  data_name = "agent_data",
  con = NULL,
  db_path = NULL,
  project = NULL,
  selfcontained = FALSE,
  data_transport = "file",
  width = NULL,
  height = NULL
)

export_htmlwidget(...)
```

Arguments

spec	A widget specification as an R list, JSON string, JSON file path, or an existing htmlwidget. Mosaic specs are rendered with <code>rMosaic</code> ; <code>deck.gl</code> specs are rendered with <code>rDeckgl</code> .
file	Output HTML file path.
engine	Export target. "auto" infers from the spec shape; "rMosaic" and "rDeckgl" select the R package explicitly. "mosaic" and "deckgl" are accepted aliases.
data	Optional named list of data.frames to register for the widget.
data_sql	Optional SQL query to fetch from con, db_path, project, or the active SQL-Rooms database and register as data_name.
data_name	Name used when registering data_sql results.
con	Optional DBI connection used for data_sql or direct DuckDB data nodes in an rDeckgl spec.
db_path	Optional DuckDB database path used to open a temporary read-only connection for export.
project	Optional dbProject object or DBI connection.
selfcontained	Passed to <code>htmlwidgets::saveWidget()</code> . Defaults to FALSE so large exports can keep dependencies and Arrow payloads as files.
data_transport	Data transport passed to <code>rMosaic::mosaic()</code> or <code>rDeckgl::deckgl()</code> . Defaults to "file" to avoid base64 Arrow payloads in JSON.
width, height	Optional widget dimensions.
...	Arguments passed to <code>dbvisuals_export_htmlwidget()</code> .

Value

Invisibly returns a list describing the exported file, engine, dependency directory, and generated data files.

Examples

```
## Not run:
spec <- list(
  plot = list(list(
    mark = "dot",
    data = list(from = "cells"),
    x = "x",
    y = "y"
  ))
)
dbvisuals_export_htmlwidget(
  spec,
  "scatter.html",
  engine = "rMosaic",
  data = list(cells = data.frame(x = 1:10, y = 1:10))
)

## End(Not run)
```

`install_dbvisuals_deps`*Install Complete dbVisuals Dependencies*

Description

Installs all recommended dependencies for dbVisuals to enable full SQL Rooms functionality. Everything comes from CRAN: the database packages (DBI, duckdb) and the visualization backends (rDeckgl, rMosaic).

Usage

```
install_dbvisuals_deps(ask = TRUE)
```

Arguments

ask	Logical. Ask the user for confirmation before installing. Defaults to 'TRUE': interactive sessions are prompted, and non-interactive sessions install nothing unless the caller passes 'ask = FALSE' explicitly.
-----	--

Details

This function installs, all from CRAN:

- DBI, duckdb
- rDeckgl, rMosaic

The function checks which packages are already installed and only installs missing ones. In interactive sessions the user is asked to confirm before any installation starts; in non-interactive sessions the function aborts with a message.

Value

Invisibly returns 'TRUE' if all installations succeeded, 'FALSE' otherwise (e.g., the user declined, or a non-interactive session).

Examples

```
## Not run:  
# Install all recommended dependencies after confirming the prompt  
install_dbvisuals_deps()  
  
# Skip the prompt (e.g. inside an automated environment)  
install_dbvisuals_deps(ask = FALSE)  
  
## End(Not run)
```

launchBrowser

Launch the dbVisuals browser

Description

Opens a browser-based dbVisuals workspace for interactive data exploration with support for deck.gl and Mosaic visualizations. A local, authenticated server provides access to the selected DuckDB database; arbitrary R code execution and AI features are not exposed.

Usage

```
launchBrowser(  
  db_path = NULL,  
  project = NULL,  
  port = 3001,  
  frontend_port = 5173,  
  browser = TRUE,  
  daemon = FALSE,  
  host = "127.0.0.1"  
)
```

Arguments

db_path	Optional path to a DuckDB database file. If NULL, starts a standalone DuckDB-WASM workspace in the browser, independent of R data.
project	Optional 'dbProject' R6 object (from the 'dbProject' package) or a 'DBIConnection'. When supplied, 'db_path' is derived from the connection's underlying DuckDB file. Mutually exclusive with 'db_path'.
port	Port number for the R server when a database file is supplied (default: 3001)
frontend_port	Port number for the frontend server (default: 5173)
browser	Logical. Open browser automatically (default: TRUE)
daemon	Logical. Run server in background (default: FALSE)
host	Host address (default: "127.0.0.1")

Value

Invisibly returns a list with server information.

Examples

```
## Not run:
# Launch a standalone browser workspace
launchBrowser()

# Launch with a specific GiottoDB database
launchBrowser(db_path = "giottodb-visium.db")

# Launch without opening browser
launchBrowser(browser = FALSE)

## End(Not run)
```

mgl_data

Declare a data source for a viewer spec

Description

Two transport modes: * 'type = "table"' (default) — a data.frame from R, serialized as Arrow IPC and registered in DuckDB-WASM under its name. Mosaic / deck.gl panels reference it by that name. * 'type = "duckdb"' — a SQL query against the R-side DuckDB connection. Result is streamed as Arrow IPC (or GeoArrow IPC when 'format = "geoarrow"').

Usage

```
mgl_data(
  type = c("table", "duckdb"),
  df = NULL,
  query = NULL,
  format = c("arrow", "geoarrow", "json")
)
```

Arguments

type	"table" for an in-memory data.frame or "duckdb" for a SQL query against the 'con' passed to [mosaic_gl()].
df	Data frame used when 'type = "table"'.
query	SQL query used when 'type = "duckdb"'.
format	Transport format for DuckDB queries. "geoarrow" will use a WKB polygon envelope for 'cell_ID' + 'geometry' result sets and otherwise falls back to Arrow IPC.

Value

A list describing the data source, for the 'data' field of [mgl_spec()].

Examples

```
mgl_data(type = "table", df = data.frame(x = 1:3, y = 3:1))
mgl_data(type = "duckdb", query = "SELECT x, y FROM points")
```

mgl_deckgl

Build a deck.gl input node for a Mosaic spec

Description

Produces 'list(input = "deckgl", id = ..., filterBy = ..., layers = ..., concat = ..., width = ..., height = ...)'. Drop it into a Mosaic spec anywhere an 'input = "menu"' would be valid.

Usage

```
mgl_deckgl(
  id,
  layers,
  concat = NULL,
  filter_by = NULL,
  tools = c("pan", "lasso", "+lasso", "clear", "fit"),
  title = NULL,
  width = NULL,
  height = NULL,
  ...
)
```

Arguments

id	Required identifier (used for cleanup + lasso source tag).
layers	List of deck.gl layer specs ('@type', 'data', accessors). Layer 'data' is a string referencing a name from 'mgl_spec(data = ...)'.

concat	Optional concat config: 'list(orient = "hconcat", panes = list(list(x = "umap_1", y = "umap_2"), ...))'. Each pane reuses the same 'layers' but with the position accessor swapped to (pane\$x, pane\$y) — used for the UMAP / PCA dim-pair grid.
filter_by	A Mosaic param reference (character "\$sel") or a pre-built Selection. The deck.gl panel subscribes to it and re-renders when its clauses change.
tools	Toolbar buttons. Defaults to 'c("pan", "lasso", "+lasso", "clear", "fit")'. "lasso" replaces the previous lasso clause, "+lasso" unions the new lasso with the previous one, and "clear" removes this deck.gl panel's lasso clause.
title	Optional title shown above the canvas.
width, height	CSS dimensions for the panel (number → px).
...	Further options passed verbatim (e.g. 'initialViewState').

Value

A list describing the panel, for use inside [mgl_spec()].

Examples

```
panel <- mgl_deckgl(
  id = "map", filter_by = "$sel", width = 400, height = 300,
  layers = list(list(`@@type` = "ScatterplotLayer", id = "pts", data = "pts",
    getPosition = "@@=[x, y]", getRadius = 2,
    radiusUnits = "pixels")))
panel$input
```

mgl_spec

Construct a top-level Mosaic spec for the viewer

Description

Accepts the same shape as Mosaic / rMosaic: meta, data, params, vconcat / hconcat / plot / input ...

Usage

```
mgl_spec(...)
```

Arguments

... Named top-level fields of a Mosaic spec.

Details

Pass each top-level field as a named argument and they will be packaged into a list. To author specs ergonomically just build a regular R list.

Value

A list with class `"mgl_spec"`.

See Also

`[mosaic_gl()]` for a complete linked example.

Examples

```
spec <- mgl_spec(
  data = list(pts = mgl_data(type = "table",
                             df = data.frame(x = 1:3, y = c(2, 4, 3)))),
  plot = list(list(mark = "dot", data = list(from = "pts"), x = "x", y = "y"))
)
names(spec)
```

mosaic_gl

Render a linked Mosaic + deck.gl visualization

Description

Bundles `vgplot` and `deck.gl` into one browser scope, with a shared DuckDB-WASM coordinator and `'vg.Selection.crossfilter()'` so that any panel that brushes/toggles/lassoes contributes a clause that filters every other panel.

Usage

```
mosaic_gl(
  spec,
  con = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL,
  data_transport = c("auto", "inline", "file"),
  data_dir = NULL
)
```

Arguments

<code>spec</code>	A spec built with <code>[mgl_spec()]</code> (or a plain list with the same structure).
<code>con</code>	Optional DBI connection to an R-side DuckDB used to hydrate <code>'mgl_data(type = "duckdb")'</code> references. Pass an existing connection so spatial / nanoarrow / gearrow extensions you've already configured are reused. The connection is never disconnected by this package, in a Shiny session or otherwise: the caller owns its lifetime.
<code>width, height</code>	Widget dimensions.
<code>element_id</code>	Optional DOM id.

data_transport How query-backed sources (`mgl_data(type = "duckdb")`) reach the browser. `"inline"` embeds each result set as base64 in the widget JSON; `"file"` streams it into `'data_dir'` and the page fetches the file by relative URL, so nothing is copied into the HTML; `"auto"` (the default) picks `"file"` when `'data_dir'` is given and `"inline"` otherwise. Both transports let DuckDB write the file itself (COPY ... FORMAT ARROWS when the community 'nanoarrow' extension loads, else a record-batch stream through `'arrow::write_ipc_stream()'`, else COPY ... FORMAT PARQUET), so the rows are never materialised in R. Each envelope records the rung used in `'exportMethod'`; `'options(dbVisuals.export_methods = "copy_parquet")'` (or any subset of `"copy_arrows"`, `"record_batch_stream"`, `"copy_parquet"`) restricts the ladder, mainly for diagnostics. `'type = "table"'` data frames are always embedded inline because they already live in R.

data_dir Directory that receives the exported files under `'data_transport = "file"'`; when omitted a session temporary directory is used. The files travel with the widget as an html dependency attachment, so the RStudio Viewer, Shiny and `'htmlwidgets::saveWidget(selfcontained = FALSE)'` all resolve them (`'selfcontained = TRUE'` is not supported for file transport).

Value

An `htmlwidget`.

Internet access

The widget loads 'DuckDB-WASM' from a public content delivery network (jsDelivr) when it is displayed, so viewing it requires internet access.

Security

In a Shiny session with `'con'` supplied, the widget's query channel runs the SQL that the page sends on `'con'`. That is harmless when the app is used locally, but in an app deployed to other users any visitor can send SQL to that connection, and 'DuckDB' SQL can read files the server process can read. For a deployed app, pass a read-only connection and disable external access first, e.g. `'DBI::dbExecute(con, "SET enable_external_access = false")'` followed by `'DBI::dbExecute(con, "SET lock_configuration = true")'`.

Examples

```
cells <- data.frame(cell_ID = sprintf("c%02d", 1:50),
                    x = stats::runif(50), y = stats::runif(50),
                    total_expr = stats::rpois(50, 20))
spec <- mgl_spec(
  data = list(cells = mgl_data(type = "table", df = cells)),
  params = list(sel = list(select = "crossfilter")),
  vconcat = list(
    list(plot = list(
      list(mark = "rectY", data = list(from = "cells", filterBy = "$sel"),
        x = list(bin = "total_expr"), y = list(count = NULL)),
      list(select = "intervalX", as = "$sel")),
    width = 400, height = 120),
```

```

    mgl_deckgl(
      id = "map", filter_by = "$sel", width = 400, height = 300,
      layers = list(list(`@@type` = "ScatterplotLayer", id = "cells",
                          data = "cells", getPosition = "@@=[x, y]",
                          getRadius = 3, radiusUnits = "pixels")))
    )
  )
  # Building the widget serialises the data frame with 'arrow'.
  if (requireNamespace("arrow", quietly = TRUE)) {
    widget <- mosaic_gl(spec)
    if (interactive()) widget
  }
}

```

mosaic_glOutput

*Shiny output binding for the Mosaic + deck.gl viewer***Description**

Shiny output binding for the Mosaic + deck.gl viewer

Usage

```
mosaic_glOutput(outputId, width = "100%", height = "600px")
```

Arguments

outputId	output variable to read the plot/image from.
width, height	Image width/height. Must be a valid CSS unit (like "100%", "400px", "auto") or a number, which will be coerced to a string and have "px" appended. These two arguments are ignored when inline = TRUE, in which case the width/height of a plot must be specified in renderPlot(). Note that, for height, using "auto" or "100%" generally will not work as expected, because of how height is computed with HTML/CSS.

Value

A Shiny UI element that holds a [mosaic_gl()] widget.

Examples

```

if (interactive() && requireNamespace("shiny", quietly = TRUE) &&
    requireNamespace("arrow", quietly = TRUE)) {
  pts <- data.frame(id = 1:100, x = stats::rnorm(100), y = stats::rnorm(100))
  spec <- mgl_spec(
    data = list(pts = mgl_data(type = "table", df = pts)),
    plot = list(list(mark = "dot", data = list(from = "pts"), x = "x", y = "y")))
  shiny::shinyApp(
    ui = shiny::fluidPage(mosaic_glOutput("viewer")),
    server = function(input, output) {

```

```

    output$viewer <- renderMosaic_gl(mosaic_gl(spec))
  })
}

```

renderMosaic_gl

*Shiny render expression for the Mosaic + deck.gl viewer***Description**

Shiny render expression for the Mosaic + deck.gl viewer

Usage

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

Arguments

expr	An expression that returns a list.
env	The parent environment for the reactive expression. By default, this is the calling environment, the same as when defining an ordinary non-reactive expression. If expr is a quosure and quoted is TRUE, then env is ignored.
quoted	If it is TRUE, then the <code>quote()</code> ed value of expr will be used when expr is evaluated. If expr is a quosure and you would like to use its expression as a value for expr, then you must set quoted to TRUE.

Value

A Shiny render function for use with `[mosaic_glOutput()]`.

Examples

```

if (interactive() && requireNamespace("shiny", quietly = TRUE) &&
    requireNamespace("arrow", quietly = TRUE)) {
  pts <- data.frame(id = 1:100, x = stats::rnorm(100), y = stats::rnorm(100))
  spec <- mgl_spec(
    data = list(pts = mgl_data(type = "table", df = pts)),
    plot = list(list(mark = "dot", data = list(from = "pts"), x = "x", y = "y")))
  shiny::shinyApp(
    ui = shiny::fluidPage(mosaic_glOutput("viewer")),
    server = function(input, output) {
      output$viewer <- renderMosaic_gl(mosaic_gl(spec))
    })
}

```

sqlrooms_export_table *Export Table to SQL Rooms*

Description

Materializes an in-memory ‘data.frame’ into the DuckDB database backing the current SQL Rooms session. When a ‘project’ is supplied (or one was attached at ‘launchBrowser()’ time) the table is written through the project’s writable connection. Otherwise the data.frame is stashed in a package-level cache for later retrieval (back-compat behavior).

Usage

```
sqlrooms_export_table(name, data, project = NULL, overwrite = TRUE)
```

Arguments

name	Character. Table name to write.
data	A ‘data.frame’.
project	Optional. A ‘dbProject’ R6 object or ‘DBIConnection’. Defaults to whatever was passed to [launchBrowser()].
overwrite	Logical. Overwrite an existing table (default ‘TRUE’).

Details

For lazy ‘tbl’ objects (dbplyr / dbMatrix / dbSpatial), use [sqlrooms_register()] instead, which creates a SQL VIEW with no data movement.

Examples

```
## Not run:
df <- data.frame(x = 1:10, y = rnorm(10))
sqlrooms_export_table("my_table", df)

## End(Not run)
```

sqlrooms_register	<i>Register a lazy tbl or dbMatrix/dbSpatial as a VIEW in the SQL Rooms database</i>
-------------------	--

Description

Creates a SQL ‘VIEW’ (optionally ‘TEMPORARY’) over ‘x’ in the connection associated with the current SQL Rooms session. No data is materialized: the view is a lazy alias that the SPA / ‘api/arrow-query’ can query.

Usage

```
sqlrooms_register(name, x, project = NULL, temporary = FALSE, overwrite = TRUE)
```

Arguments

name	Character. View name to create.
x	A lazy 'tbl', dbMatrix/dbSpatial, or data.frame.
project	Optional. A 'dbProject' R6 object or 'DBIConnection'. Defaults to whatever was passed to [launchBrowser()].
temporary	Logical. Create a 'TEMPORARY VIEW' (default 'FALSE'). Persisted (non-temporary) views are written to the database file and are visible to the SPA after the next page reload.
overwrite	Logical. Use 'CREATE OR REPLACE VIEW' (default 'TRUE').

Details

Accepted inputs: - 'tbl_lazy' / 'tbl_sql': rendered via 'dbplyr::sql_render()'. - 'dbMatrix' / 'dbSpatial' (or any S4 with a '[' extractor returning a 'tbl'): the underlying lazy tbl is extracted via 'x[]'. - 'data.frame': forwarded to [sqlrooms_export_table()] (materializes).

Value

Invisibly, a 'dplyr::tbl()' pointing at the new view, or 'TRUE'.

Examples

```
## Not run:
library(dplyr)
tbl(con, "cells") |>
  filter(x > 0) |>
  sqlrooms_register("cells_pos", x = _)

## End(Not run)
```

stop_sqlrooms

Stop SQL Rooms Processes

Description

Stops SQL Rooms background processes started by [launchBrowser()]. This helper stops both the R database server and the frontend server.

Usage

```
stop_sqlrooms(dashboard = NULL)
```

Arguments

dashboard Optional list returned by [launchBrowser()]. If NULL, stops the most recent SQL Rooms processes tracked in the package environment.

Value

Invisibly returns a list with logical flags: r_server_stopped, frontend_stopped.

Examples

```
## Not run:  
result <- launchBrowser(browser = FALSE)  
stop_sqlrooms(result)  
  
## End(Not run)
```

Index

dbvisuals_export_htmlwidget, [2](#)

export_htmlwidget
 (dbvisuals_export_htmlwidget),
 [2](#)

htmlwidgets::saveWidget(), [3](#)

install_dbvisuals_deps, [4](#)

launchBrowser, [5](#)

mgl_data, [6](#)

mgl_deckgl, [7](#)

mgl_spec, [8](#)

mosaic_gl, [9](#)

mosaic_glOutput, [11](#)

quote(), [12](#)

renderMosaic_gl, [12](#)

sqlrooms_export_table, [13](#)

sqlrooms_register, [13](#)

stop_sqlrooms, [14](#)