Package: manby (via r-universe)

November 14, 2026

Title Vector Tilesets Like 'tippecanoe', Several Times Faster

Version 0.2.8

Description Builds vector tilesets in 'PMTiles' and 'MBTiles' from sf data frames, 'DuckDB' tables, 'Arrow' tables and files, with the tiles that 'tippecanoe' makes from the same features. 'manby', a 'Rust' port of 'tippecanoe', is built into the package from CRAN and in 'webR', and elsewhere the package runs the 'manby' program, which it downloads from the package's release on first use.

License BSD_2_clause + file LICENSE

URL https://birkskyum.github.io/manby/, https://github.com/birkskyum/manby

BugReports https://github.com/birkskyum/manby/issues

Depends R (>= 4.0)

Imports jsonlite, nanoarrow, processx, stats, tools, utils

Suggests arrow, DBI, dbplyr, dplyr, duckdb, sf, testthat (>= 3.0.0), withr

SystemRequirements Cargo (Rust's package manager), rustc >= 1.90, to build manby into the package

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Config/pak/sysreqs libzstd-dev libclang-dev

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

Date/Publication 2026-11-14 12:37:01 UTC

RemoteUrl https://github.com/birkskyum/manby

RemoteRef v0.2.8

RemoteSha 99a3f8b32e4b855f597719eba6ab2ce34a9e7f02

RemoteSubdir r

Contents

manby 2
manby_binary 3
serve 3
tile 4
tile_stats 6

Index **7**

`manby` *Runs manby*

Description

Runs manby with arguments as on the command line, such as `manby join`.

Usage

```
manby(...)
```

Arguments

`...`	The arguments.

Value

Nothing. If manby fails, an error of class `manby_error`, with its exit status in `status`: 100 when the tileset is complete only to a lower zoom, as for tippecanoe.

Examples

```
## Not run:
manby("join", "-o", "map.pmtiles", "roads.pmtiles", "places.pmtiles")

## End(Not run)
```

manby_binary *The manby program*

Description

`manby_binary()` gives the path of the manby program that the package runs: the one that the `manby.binary` option or the `MANBY_BINARY` environment variable names, or else the one that `install_manby()` downloaded. If there is neither, it calls `install_manby()`.

Usage

```
manby_binary()

install_manby(version = own_version(), force = FALSE)
```

Arguments

`version`	The version of manby to download. The default is the package's own.
`force`	Download it again, even if it is in the cache.

Details

`install_manby()` downloads manby from the GitHub release of the package's version, checks it against the release's checksums, and keeps it in the package's cache, `tools::R_user_dir()`. manby runs on macOS, Linux and Windows.

Value

The path of the program.

Examples

```
## Not run:
manby_binary()

## End(Not run)
```

serve *Serves tilesets with a map of them*

Description

Serves PMTiles archives, MBTiles databases or directories of tiles with `manby serve`, until `$stop()` or the end of the R session. The map, at `$url`, shows each tileset's layers on a basemap, with the attributes of the features clicked. A tileset made again while it is served shows when the map is reloaded.

Usage

```
serve(tilesets, port = 8080, host = "127.0.0.1", open = interactive())
```

Arguments

`tilesets`	The paths of the tilesets.
`port`	The port, or 0 for any free port.
`host`	The address to listen on.
`open`	Show the map, in the viewer pane or a browser. With manby built into the package, as from CRAN, it serves on threads of R's own process, until `$stop()` or the server is garbage collected.

Details

Each tileset's tiles and TileJSON can also go in a map of one's own, as can a PMTiles archive itself, at its file name after `$url`.

Value

A `manby_server`, with `url`, the map's address, `tilejson`, the TileJSON address of each tileset by name, `stop()`, which ends it, and `alive()`, whether it serves.

Examples

```
## Not run:
server <- serve("roads.pmtiles")
server$tilejson[["roads"]]
server$stop()

## End(Not run)
```

`tile` *Makes a tileset*

Description

Makes the tileset `output` from `data`, with the tiles that tippecanoe makes from the same features.

Usage

```
tile(data, output, ..., layer = NULL, force = TRUE, h3_sql = NULL)
```

Arguments

`data`	The features: a file name, a table, or a named list of layers.
`output`	The tileset to write, such as `"roads.pmtiles"`.
`...`	tippecanoe's options. A named argument is a long option, so `maximum_zoom = 14` is `--maximum-zoom=14`, `drop_densest_as_needed = TRUE` is `--drop-densest-as-needed`, and a vector repeats an option. Other arguments are passed as they are, such as `"-zg"`.
`layer`	The name of the layer, for a single input. Without one, the layer is named after the file, or for a table after the tileset.
`force`	Overwrite an existing tileset.
`h3_sql`	Aggregates for the hexagons of `--h3`, as a named vector of DuckDB SQL, such as `c(p90 = "quantile_cont(price, 0.9)")`. DuckDB computes each over the points of each H3 cell, with its spatial and H3 extensions, and the hexagons get them as attributes of those names. The points of a Parquet file are taken to be in WGS 84.

Details

`data` is a file that manby reads, such as GeoJSON, FlatGeobuf, CSV, GeoParquet, GeoPackage or Shapefile, a file in another format that GDAL reads, or a table:

- An sf data frame. One in a CRS other than WGS 84 or Web Mercator is reprojected to WGS 84 first.
- A DuckDB table from dplyr, such as `dplyr::tbl(con, "roads")` or `dplyr::tbl(con, dplyr::sql("SELECT ...")`), which DuckDB writes to a temporary Parquet file, so it need not fit in memory.
- An Arrow table or stream, or a data frame, with its geometry in a GeoArrow column or in a column named `geometry` in well-known binary.

A named list makes a layer of each of these. Other tables are written to a temporary Arrow file for manby to read.

A file in a format that GDAL reads and manby does not, such as KML, GPX, GML, OpenStreetMap (`.osm.pbf`), SQLite, a File Geodatabase (`.gdb`) or a spreadsheet, is read with sf, or with DuckDB's spatial extension if sf is not installed, and reprojected to WGS 84. The layer's name picks the source's layer of that name, if it has one, and otherwise the first.

Value

`output`, invisibly.

Examples

```
## Not run:
nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"))
tile(nc, "nc.pmtiles", layer = "counties", maximum_zoom = "g")

con <- DBI::dbConnect(duckdb::duckdb())
roads <- dplyr::tbl(con, dplyr::sql(
```

```
  "SELECT name, class, geometry FROM 'roads.parquet' WHERE class = 'motorway'"
))
tile(roads, "roads.pmtiles", maximum_zoom = 14, drop_densest_as_needed = TRUE)

tile(list(roads = roads, places = "places.parquet"), "map.pmtiles", "-zg")

tile(places, "places.pmtiles", maximum_zoom = 12, h3 = TRUE,
  h3_sql = c(p90 = "quantile_cont(price, 0.9)", kinds = "count(DISTINCT kind)"))

tile(list(points = "liechtenstein.osm.pbf", lines = "liechtenstein.osm.pbf"),
  "liechtenstein.pmtiles", maximum_zoom = 14, drop_densest_as_needed = TRUE)

## End(Not run)
```

`tile_stats` *The sizes of a tileset's tiles*

Description

How big a PMTiles archive's, an MBTiles database's or a directory's tiles are at each zoom, as stored, and the largest tiles. With `layers`, also how many features of each layer the tiles at each zoom hold, which shows where dropping and coalescing took effect.

Usage

```
tile_stats(tileset, layers = FALSE, largest = 10)
```

Arguments

`tileset`	The path of the tileset.
`layers`	Count the features of each layer, from every tile.
`largest`	How many of the largest tiles to list.

Value

A list of data frames: `zooms`, with the tiles at each zoom and their total, mean, median, 99th percentile and largest size in bytes; `largest`, the largest tiles; and with `layers`, `layers`, the tiles and features of each layer at each zoom.

Examples

```
## Not run:
tile_stats("roads.pmtiles", layers = TRUE)

## End(Not run)
```

Index

install_manby *(*manby_binary*)*, 3

manby, 2
manby_binary, 3

serve, 3

tile, 4
tile_stats, 6
tools::R_user_dir(), *3*