Package: pmtiles (via r-universe)

May 29, 2026

Type Package

Title R Interface to PMTiles

Version 0.1.2

Date 2025-11-30

Description Provides R bindings to the PMTiles command-line interface for working with PMTiles archives. PMTiles is a single-file archive format for tiled data that enables efficient cloud-native storage and retrieval of map tiles. This package wraps the go-pmtiles CLI to provide functions for inspecting, extracting, converting, and serving PMTiles archives.

License MIT + file LICENSE

Copyright Includes go-pmtiles library (BSD-3-Clause), see inst/COPYRIGHTS

Encoding UTF-8

SystemRequirements tippecanoe (optional, for pm_create())

Imports processx, jsonlite, tools, httpuv, stringi

Suggests mapgl, sf, yyjsonr

RoxygenNote 7.3.2

URL https://github.com/walkerke/pmtiles

BugReports https://github.com/walkerke/pmtiles/issues

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

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

Date/Publication 2025-11-30 18:57:25 UTC

RemoteUrl https://github.com/walkerke/pmtiles

RemoteRef HEAD

RemoteSha 67553858146500fb49f86fb1b7697a8b8cdac7b9

Contents

`pm_cluster` *Cluster a PMTiles archive*

Description

Cluster an unclustered PMTiles archive, optimizing its size and layout. Archives created by tippecanoe, planetiler, and the pmtiles CLI are already clustered and do not need this operation.

Usage

```
pm_cluster(input, no_deduplication = FALSE, verbose = TRUE)
```

Arguments

`input`	Path to input PMTiles file. The file will be modified in place.
`no_deduplication`	Logical. If 'TRUE', skips tile deduplication to speed up clustering. Use this if you know the input has only unique tiles. Default is 'FALSE'.
`verbose`	Logical. If 'TRUE', prints progress information. Default is 'TRUE'.

Value

Invisibly returns the path to the clustered archive.

Examples

```
## Not run:
# Cluster an archive
pm_cluster("archive.pmtiles")

# Cluster without deduplication (faster)
pm_cluster("archive.pmtiles", no_deduplication = TRUE)

## End(Not run)
```

`pm_convert` *Convert MBTiles to PMTiles*

Description

Convert an MBTiles database to PMTiles format. The conversion process automatically deduplicates tiles unless disabled.

Usage

```
pm_convert(
  input,
  output,
  force = FALSE,
  no_deduplication = FALSE,
  tmpdir = NULL,
  verbose = TRUE
)
```

Arguments

`input`	Path to input MBTiles file.
`output`	Path for output PMTiles file.
`force`	Logical. If 'TRUE', removes existing output file if present. Default is 'FALSE'.
`no_deduplication`	Logical. If 'TRUE', skips tile deduplication to speed up conversion. Use this if you know the input has only unique tiles. Default is 'FALSE'.
`tmpdir`	Optional path to a folder for temporary files during conversion. If not specified, uses the system temporary directory.
`verbose`	Logical. If 'TRUE', prints progress information. Default is 'TRUE'.

Value

Invisibly returns the path to the output archive.

Examples

```
## Not run:
# Convert MBTiles to PMTiles
pm_convert("input.mbtiles", "output.pmtiles")

# Convert without deduplication (faster)
pm_convert("input.mbtiles", "output.pmtiles", no_deduplication = TRUE)

# Force overwrite existing file
pm_convert("input.mbtiles", "output.pmtiles", force = TRUE)

## End(Not run)
```

`pm_create` *Create PMTiles or MBTiles from GeoJSON with tippecanoe*

Description

Generate vector tiles from GeoJSON, FlatGeobuf, or CSV input using tippecanoe. This function requires tippecanoe to be installed on your system. See https://github.com/felt/tippecanoe for installation instructions.

Usage

```
pm_create(
  input,
  output,
  layer_name = NULL,
  min_zoom = NULL,
  max_zoom = NULL,
  guess_maxzoom = FALSE,
  smallest_maximum_zoom_guess = NULL,
  base_zoom = NULL,
  extend_zooms_if_still_dropping = FALSE,
  full_detail = NULL,
  low_detail = NULL,
  minimum_detail = NULL,
  extra_detail = NULL,
  exclude = NULL,
  include = NULL,
  exclude_all = FALSE,
  drop_rate = NULL,
  drop_densest_as_needed = FALSE,
  drop_fraction_as_needed = FALSE,
  drop_smallest_as_needed = FALSE,
  drop_lines = FALSE,
```

```
  drop_polygons = FALSE,
  coalesce = FALSE,
  coalesce_smallest_as_needed = FALSE,
  coalesce_densest_as_needed = FALSE,
  coalesce_fraction_as_needed = FALSE,
  cluster_distance = NULL,
  cluster_maxzoom = NULL,
  simplification = NULL,
  no_line_simplification = FALSE,
  simplify_only_low_zooms = FALSE,
  no_tiny_polygon_reduction = FALSE,
  detect_shared_borders = FALSE,
  no_simplification_of_shared_nodes = FALSE,
  preserve_input_order = FALSE,
  reorder = FALSE,
  hilbert = FALSE,
  maximum_tile_bytes = NULL,
  maximum_tile_features = NULL,
  no_feature_limit = FALSE,
  no_tile_size_limit = FALSE,
  generate_ids = TRUE,
  calculate_feature_density = FALSE,
  read_parallel = FALSE,
  attribution = NULL,
  description = NULL,
  buffer = NULL,
  other_options = NULL,
  force = TRUE,
  keep_geojson = FALSE,
  quiet = FALSE
)
```

Arguments

input	An sf object, path to a GeoJSON/FlatGeobuf/CSV file, or a list for multi-layer output. For multi-layer PMTiles, provide either: • A list of sf objects or file paths (all layers share the same options) • A list of pm_layer objects (each layer can have different options)
output	Path to output file (.pmtiles or .mbtiles)
layer_name	Name for the layer in the tileset. If NULL, derived from input filename or a random string for sf objects (tippecanoe -l). For multi-layer input, can be a character vector of names (one per layer), or names will be derived from list names.
min_zoom	Minimum zoom level (tippecanoe -Z, default 0)
max_zoom	Maximum zoom level (tippecanoe -z, default 14)
guess_maxzoom	If TRUE, guess appropriate maxzoom based on feature density (tippecanoe -zg)

`smallest_maximum_zoom_guess`
: Use specified zoom if lower maxzoom is guessed (tippecanoe –smallest-maximum-zoom-guess)

`base_zoom`
: Zoom at and above which all points are included (tippecanoe -B). If NULL, defaults to maxzoom.

`extend_zooms_if_still_dropping`
: Increase maxzoom if features still being dropped (tippecanoe -ae)

`full_detail`
: Detail at max zoom (default 12, for 4096 tile resolution, tippecanoe -d)

`low_detail`
: Detail at lower zooms (default 12, tippecanoe -D)

`minimum_detail`
: Minimum detail if tiles too big (default 7, tippecanoe -m)

`extra_detail`
: Generate tiles with extra detail for precision (tippecanoe –extra-detail)

`exclude`
: Character vector of attribute names to exclude (tippecanoe -x)

`include`
: Character vector of attribute names to include, excluding all others (tippecanoe -y)

`exclude_all`
: If TRUE, exclude all attributes and encode only geometries (tippecanoe -X)

`drop_rate`
: Rate at which features dropped at zoom levels below basezoom (default 2.5, tippecanoe -r). Use "g" for auto-guess.

`drop_densest_as_needed`
: Reduce feature spacing if tile too large (tippecanoe -as)

`drop_fraction_as_needed`
: Drop fraction of features to keep under size limit (tippecanoe -ad)

`drop_smallest_as_needed`
: Drop smallest features to keep under size limit (tippecanoe -an)

`drop_lines`
: Apply dot-dropping to lines (tippecanoe -al)

`drop_polygons`
: Apply dot-dropping to polygons (tippecanoe -ap)

`coalesce`
: Coalesce consecutive features with same attributes (tippecanoe -ac)

`coalesce_smallest_as_needed`
: Combine smallest features into nearby ones (tippecanoe -aN)

`coalesce_densest_as_needed`
: Combine densest features into nearby ones (tippecanoe -aD)

`coalesce_fraction_as_needed`
: Combine fraction of features into nearby ones (tippecanoe -aS)

`cluster_distance`
: Cluster points within distance of each other (tippecanoe -K, max 255)

`cluster_maxzoom`
: Max zoom for clustering (tippecanoe -k). Use "g" to set to maxzoom - 1.

`simplification`
: Multiply tolerance for line/polygon simplification (tippecanoe -S, default ~1)

`no_line_simplification`
: Don’t simplify lines and polygons (tippecanoe -ps)

`simplify_only_low_zooms`
: Don’t simplify at maxzoom (tippecanoe -pS)

`no_tiny_polygon_reduction`
: Don’t combine tiny polygons into squares (tippecanoe -pt)

`detect_shared_borders`
: Detect and simplify shared polygon borders identically (tippecanoe -ab)

`no_simplification_of_shared_nodes`
: Don't simplify nodes where lines converge/diverge (tippecanoe -pn)

`preserve_input_order`
: Preserve original input order instead of geographic order (tippecanoe -pi)

`reorder`
: Reorder features to put same attributes in sequence (tippecanoe -ao)

`hilbert`
: Use Hilbert Curve order instead of Z-order (tippecanoe -ah)

`maximum_tile_bytes`
: Maximum compressed tile size in bytes (default 500K, tippecanoe -M)

`maximum_tile_features`
: Maximum features per tile (default 200,000, tippecanoe -O)

`no_feature_limit`
: Don't limit tiles to 200,000 features (tippecanoe -pf)

`no_tile_size_limit`
: Don't limit tiles to 500K bytes (tippecanoe -pk)

`generate_ids`
: Add feature IDs to features without them (tippecanoe -ai)

`calculate_feature_density`
: Add tippecanoe_feature_density attribute (tippecanoe -ag)

`read_parallel`
: Use multiple threads for line-delimited GeoJSON (tippecanoe -P)

`attribution`
: Attribution text for tileset (tippecanoe -A)

`description`
: Description for tileset (tippecanoe -N)

`buffer`
: Buffer size in screen pixels (default 5, tippecanoe -b)

`other_options`
: Character vector of additional tippecanoe options not covered by other parameters. Example: c("-pf", "-pk", "–coalesce")

`force`
: If TRUE, overwrite existing output file (default TRUE, tippecanoe -f)

`keep_geojson`
: If TRUE, keep temporary GeoJSON file for sf objects

`quiet`
: If TRUE, suppress progress messages (tippecanoe -q)

Details

This function wraps the tippecanoe command-line tool. Tippecanoe must be installed separately:

- macOS: `brew install tippecanoe`
- Ubuntu: `sudo apt-get install tippecanoe`
- From source: https://github.com/felt/tippecanoe

The function handles sf objects by converting them to temporary GeoJSON files. For faster GeoJSON writing with large datasets, install the `yyjsonr` package, which can be significantly faster than the default `sf::st_write()`.

Value

Path to output file (invisibly)

See Also

[pm_layer()] for creating layers with per-layer options

Examples

```
## Not run:
library(sf)

# Simple usage with sf object
pm_create(
  my_sf_data,
  "output.pmtiles",
  max_zoom = 14
)

# Complex parcel tileset
pm_create(
  "parcels.geojson",
  "parcels.pmtiles",
  layer_name = "parcels",
  min_zoom = 10,
  max_zoom = 18,
  full_detail = 15,
  preserve_input_order = TRUE,
  no_tiny_polygon_reduction = TRUE,
  coalesce_densest_as_needed = TRUE,
  coalesce_fraction_as_needed = TRUE,
  extend_zooms_if_still_dropping = TRUE,
  simplification = 1,
  detect_shared_borders = TRUE,
  other_options = c("-pf", "-pk", "-ai")
)

# Point clustering
pm_create(
  points_sf,
  "points.pmtiles",
  max_zoom = 14,
  cluster_distance = 10,
  cluster_maxzoom = "g",
  generate_ids = TRUE
)

# With attribute filtering
pm_create(
  roads_sf,
  "roads.pmtiles",
  include = c("name", "highway", "surface"),
  drop_densest_as_needed = TRUE,
  simplification = 10
)
```

```
# Multi-layer with shared options (simple)
pm_create(
  input = list(
    counties = counties_sf,
    tracts = tracts_sf
  ),
  output = "census.pmtiles",
  min_zoom = 2,
  max_zoom = 12
)

# Multi-layer with per-layer options (using pm_layer)
pm_create(
  input = list(
    pm_layer(
      input = counties_sf,
      layer_name = "counties",
      min_zoom = 2,
      max_zoom = 8
    ),
    pm_layer(
      input = tracts_sf,
      layer_name = "tracts",
      min_zoom = 8,
      max_zoom = 14,
      drop_densest_as_needed = TRUE
    )
  ),
  output = "census.pmtiles",
  generate_ids = TRUE
)

## End(Not run)
```

`pm_edit` *Edit PMTiles archive header or metadata*

Description

Modify parts of the PMTiles archive header or replace the JSON metadata. Editing only the header modifies the file in-place, while editing metadata creates a new copy.

Usage

```
pm_edit(input, header_json = NULL, metadata = NULL, verbose = TRUE)
```

Arguments

input	Path to PMTiles file to edit.
header_json	Path to JSON file containing modified header data. Use ‘pm_show(input, header_json = TRUE)‘ to get current header.
metadata	Path to JSON file containing replacement metadata. Use ‘pm_show(input, metadata = TRUE)‘ to get current metadata.
verbose	Logical. If ‘TRUE‘, prints progress information. Default is ‘TRUE‘.

Details

Editable Header Fields

The following header fields can be modified: - ‘tile_type‘: Type of tiles (e.g., "mvt", "png", "jpg") - ‘tile_compression‘: Compression format - ‘minzoom‘: Minimum zoom level - ‘maxzoom‘: Maximum zoom level - ‘bounds‘: Geographic bounds - ‘center‘: Center point and zoom

Other header fields are not editable.

Important Notes

- Editing only the header modifies the file in-place - Writing new metadata requires creating a new archive copy - The new copy will replace the original file

Value

Invisibly returns the path to the edited archive.

Examples

```
## Not run:
# Get current header and metadata
header <- pm_show("archive.pmtiles", header_json = TRUE)
metadata <- pm_show("archive.pmtiles", metadata = TRUE)

# Modify and save to JSON files
jsonlite::write_json(header, "header.json", auto_unbox = TRUE)
jsonlite::write_json(metadata, "metadata.json", auto_unbox = TRUE)

# Edit the archive
pm_edit("archive.pmtiles",
        header_json = "header.json",
        metadata = "metadata.json")

## End(Not run)
```

pm_extract *Extract a subset from a PMTiles archive*

Description

Create a smaller PMTiles archive from a larger one by extracting a subset of zoom levels or a geographic region. The source archive can be local or remote.

Usage

```
pm_extract(
  input,
  output,
  bbox = NULL,
  region = NULL,
  minzoom = NULL,
  maxzoom = NULL,
  bucket = NULL,
  download_threads = 4,
  overfetch = 0.05,
  dry_run = FALSE,
  verbose = TRUE
)
```

Arguments

input	Path to input PMTiles archive (local or remote URL).
output	Path for the output PMTiles archive.
bbox	Numeric vector of bounding box coordinates in the form ‘c(min_lon, min_lat, max_lon, max_lat)‘. Mutually exclusive with ‘region‘.
region	Path to a GeoJSON file containing a Polygon, MultiPolygon, Feature, or FeatureCollection defining the area of interest. Mutually exclusive with ‘bbox‘.
minzoom	Minimum zoom level to extract (inclusive). Default is 0.
maxzoom	Maximum zoom level to extract (inclusive). If not specified, extracts all zoom levels from the source.
bucket	Optional remote bucket specification if ‘input‘ is remote.
download_threads	Number of parallel download threads for remote archives. Default is 4.
overfetch	Ratio of extra data to download to minimize number of requests. For example, 0.05 means 5 percent overfetch. Default is 0.05.
dry_run	Logical. If ‘TRUE‘, calculates tiles to extract without actually downloading them. Default is ‘FALSE‘.
verbose	Logical. If ‘TRUE‘, prints progress information. Default is ‘TRUE‘.

Details

Extracting a full sub-pyramid from zoom 0 to ‘maxzoom‘ is an efficient operation. However, using a ‘minzoom‘ > 0 may require many more requests and should only be used when necessary.

Value

Invisibly returns the path to the output archive.

Examples

```
## Not run:
# Extract zoom levels 0-10
pm_extract("large.pmtiles", "subset.pmtiles", maxzoom = 10)

# Extract by bounding box
pm_extract(
  "large.pmtiles",
  "bbox_subset.pmtiles",
  bbox = c(-122.5, 37.7, -122.3, 37.9)
)

# Extract by GeoJSON region
pm_extract(
  "large.pmtiles",
  "region_subset.pmtiles",
  region = "boundary.geojson",
  maxzoom = 12
)

# Extract from remote archive
pm_extract(
  "large.pmtiles",
  "local_copy.pmtiles",
  bucket = "s3://my-bucket",
  maxzoom = 10,
  download_threads = 8
)

## End(Not run)
```

---

pm_layer *Define a layer for multi-layer PMTiles creation*

Description

Creates a layer specification for use with ‘pm_create()‘ when building multi-layer PMTiles with per-layer tippecanoe options. Each layer can have its own zoom levels, drop rates, simplification settings, and other options.

Usage

```
pm_layer(
  input,
  layer_name,
  min_zoom = NULL,
  max_zoom = NULL,
  guess_maxzoom = NULL,
  smallest_maximum_zoom_guess = NULL,
  base_zoom = NULL,
  extend_zooms_if_still_dropping = NULL,
  full_detail = NULL,
  low_detail = NULL,
  minimum_detail = NULL,
  extra_detail = NULL,
  exclude = NULL,
  include = NULL,
  exclude_all = NULL,
  drop_rate = NULL,
  drop_densest_as_needed = NULL,
  drop_fraction_as_needed = NULL,
  drop_smallest_as_needed = NULL,
  drop_lines = NULL,
  drop_polygons = NULL,
  coalesce = NULL,
  coalesce_smallest_as_needed = NULL,
  coalesce_densest_as_needed = NULL,
  coalesce_fraction_as_needed = NULL,
  cluster_distance = NULL,
  cluster_maxzoom = NULL,
  simplification = NULL,
  no_line_simplification = NULL,
  simplify_only_low_zooms = NULL,
  no_tiny_polygon_reduction = NULL,
  detect_shared_borders = NULL,
  no_simplification_of_shared_nodes = NULL,
  preserve_input_order = NULL,
  reorder = NULL,
  hilbert = NULL,
  maximum_tile_bytes = NULL,
  maximum_tile_features = NULL,
  no_feature_limit = NULL,
  no_tile_size_limit = NULL,
  generate_ids = NULL,
  calculate_feature_density = NULL,
  read_parallel = NULL,
  buffer = NULL,
  other_options = NULL
)
```

Arguments

input	An sf object, or path to a GeoJSON, FlatGeobuf, or CSV file
layer_name	Name for this layer in the tileset (required)
min_zoom	Minimum zoom level (tippecanoe -Z)
max_zoom	Maximum zoom level (tippecanoe -z)
guess_maxzoom	If TRUE, guess appropriate maxzoom based on feature density (tippecanoe -zg)
smallest_maximum_zoom_guess	Use specified zoom if lower maxzoom is guessed (tippecanoe –smallest-maximum-zoom-guess)
base_zoom	Zoom at and above which all points are included (tippecanoe -B). If NULL, defaults to maxzoom.
extend_zooms_if_still_dropping	Increase maxzoom if features still being dropped (tippecanoe -ae)
full_detail	Detail at max zoom (default 12, for 4096 tile resolution, tippecanoe -d)
low_detail	Detail at lower zooms (default 12, tippecanoe -D)
minimum_detail	Minimum detail if tiles too big (default 7, tippecanoe -m)
extra_detail	Generate tiles with extra detail for precision (tippecanoe –extra-detail)
exclude	Character vector of attribute names to exclude (tippecanoe -x)
include	Character vector of attribute names to include, excluding all others (tippecanoe -y)
exclude_all	If TRUE, exclude all attributes and encode only geometries (tippecanoe -X)
drop_rate	Rate at which features dropped at zoom levels below basezoom (tippecanoe -r). Use "g" for auto-guess.
drop_densest_as_needed	Reduce feature spacing if tile too large (tippecanoe -as)
drop_fraction_as_needed	Drop fraction of features to keep under size limit (tippecanoe -ad)
drop_smallest_as_needed	Drop smallest features to keep under size limit (tippecanoe -an)
drop_lines	Apply dot-dropping to lines (tippecanoe -al)
drop_polygons	Apply dot-dropping to polygons (tippecanoe -ap)
coalesce	Coalesce consecutive features with same attributes (tippecanoe -ac)
coalesce_smallest_as_needed	Combine smallest features into nearby ones (tippecanoe -aN)
coalesce_densest_as_needed	Combine densest features into nearby ones (tippecanoe -aD)
coalesce_fraction_as_needed	Combine fraction of features into nearby ones (tippecanoe -aS)
cluster_distance	Cluster points within distance of each other (tippecanoe -K, max 255)

`cluster_maxzoom`
: Max zoom for clustering (tippecanoe -k). Use "g" to set to maxzoom - 1.

`simplification`
: Multiply tolerance for line/polygon simplification (tippecanoe -S)

`no_line_simplification`
: Don't simplify lines and polygons (tippecanoe -ps)

`simplify_only_low_zooms`
: Don't simplify at maxzoom (tippecanoe -pS)

`no_tiny_polygon_reduction`
: Don't combine tiny polygons into squares (tippecanoe -pt)

`detect_shared_borders`
: Detect and simplify shared polygon borders identically (tippecanoe -ab)

`no_simplification_of_shared_nodes`
: Don't simplify nodes where lines converge/diverge (tippecanoe -pn)

`preserve_input_order`
: Preserve original input order instead of geographic order (tippecanoe -pi)

`reorder`
: Reorder features to put same attributes in sequence (tippecanoe -ao)

`hilbert`
: Use Hilbert Curve order instead of Z-order (tippecanoe -ah)

`maximum_tile_bytes`
: Maximum compressed tile size in bytes (tippecanoe -M)

`maximum_tile_features`
: Maximum features per tile (tippecanoe -O)

`no_feature_limit`
: Don't limit tiles to 200,000 features (tippecanoe -pf)

`no_tile_size_limit`
: Don't limit tiles to 500K bytes (tippecanoe -pk)

`generate_ids`
: Add feature IDs to features without them (tippecanoe -ai)

`calculate_feature_density`
: Add tippecanoe_feature_density attribute (tippecanoe -ag)

`read_parallel`
: Use multiple threads for line-delimited GeoJSON (tippecanoe -P)

`buffer`
: Buffer size in screen pixels (tippecanoe -b)

`other_options`
: Character vector of additional tippecanoe options not covered by other parameters. Example: c("–coalesce")

Value

A list with class '"pm_layer"' containing the input and all specified options.

See Also

[pm_create()]

Examples

```
## Not run:
library(sf)

# Create a multi-layer PMTiles with per-layer options
pm_create(
  input = list(
    pm_layer(
      input = counties_sf,
      layer_name = "counties",
      min_zoom = 2,
      max_zoom = 8
    ),
    pm_layer(
      input = tracts_sf,
      layer_name = "tracts",
      min_zoom = 8,
      max_zoom = 14,
      drop_densest_as_needed = TRUE
    )
  ),
  output = "census.pmtiles",
  generate_ids = TRUE
)

## End(Not run)
```

`pm_serve` *Serve PMTiles files via local HTTP server with CORS*

Description

Start a local HTTP server to serve PMTiles files with CORS headers enabled. This allows PMTiles to be consumed by web maps (like mapgl) using the PMTiles.js client library. The server runs in the background and can be stopped with ‘pm_stop_server()‘.

Usage

```
pm_serve(path, port = 8080, background = TRUE)
```

Arguments

`path`	Path to a directory containing PMTiles files, or a single PMTiles file. If a single file, its directory will be served.
`port`	Port number for the HTTP server. Default is 8080.
`background`	Logical. If ‘TRUE‘, runs server in background and returns immediately. If ‘FALSE‘, blocks until interrupted. Default is ‘TRUE‘.

Details

This function serves the **raw PMTiles files** with CORS headers, allowing them to be consumed by PMTiles.js in the browser. This works with mapgl's PMTiles protocol and ‘pm_view()‘ for quick visualization.

For a file at ‘tiles/data.pmtiles‘, it will be available at: ‘http://localhost:PORT/data.pmtiles‘

File size: Works well for files up to ~1GB. For larger files, consider ‘pm_serve_zxy()‘ or serving from a Node.js server like http-server.

The server uses httpuv with custom CORS headers and HTTP range request support. When ‘background = TRUE‘, the server runs as a background daemon.

Value

If ‘background = TRUE‘, returns a list with: - ‘url‘: Base URL of the server - ‘port‘: Port number - ‘dir‘: Directory being served - ‘daemon‘: Server daemon ID (for stopping)

If ‘background = FALSE‘, blocks until interrupted and returns nothing.

See Also

[pm_stop_server()], [pm_view()]

Examples

```
## Not run:
# Serve a single PMTiles file
server <- pm_serve("data.pmtiles", port = 8080)
# File available at: http://localhost:8080/data.pmtiles

# Use in mapgl
mapgl::add_pmtiles_source(url = paste0(server$url, "/data.pmtiles"))

# Serve a directory of PMTiles files
server <- pm_serve("tiles/", port = 8080)

# Stop the server when done
pm_stop_server(server)

# Or run in foreground (blocks)
pm_serve("data.pmtiles", background = FALSE)

## End(Not run)
```

pm_serve_zxy *Serve PMTiles as Z/X/Y tile endpoints*

Description

Start a local tile server that serves PMTiles archives as standard Z/X/Y tile endpoints (e.g., '/{tileset}/{z}/{x}/{y}.mvt'). This uses the native 'pmtiles serve' command and works with any map client, not just PMTiles.js.

Unlike 'pm_serve()' which serves raw .pmtiles files for direct consumption by PMTiles.js, this function extracts individual tiles on-demand, making the tiles accessible to any mapping library that supports standard tile URLs.

Usage

```
pm_serve_zxy(
  path = ".",
  bucket = NULL,
  port = 8080,
  cors = "*",
  cache_size = 64,
  public_url = NULL,
  background = FALSE
)
```

Arguments

path	Directory containing PMTiles files, or a specific path prefix. Default is current directory ('"."').
bucket	Optional cloud storage bucket specification (e.g., '"s3://bucket-name"' or '"s3://bucket?endpoint=https://a When specified, serves tiles directly from cloud storage without downloading.
port	Port number for the tile server. Default is 8080.
cors	CORS origins to allow. Can be '"*"' for all origins, a character vector of specific origins, or 'NULL' for no CORS headers. Default is '"*"'.
cache_size	Cache size in megabytes. Default is 64 MB.
public_url	Public-facing URL for TileJSON generation (e.g., '"https://example.com"'). Required for accurate TileJSON metadata.
background	Logical. If 'TRUE', runs server in background using 'processx'. If 'FALSE' (default), runs in foreground (blocking).

Details

Tile Endpoints

The server provides these endpoints:

- **Tiles**: ‘http://localhost:PORT/TILESET/{z}/{x}/{y}.ext‘ - Extension (‘.mvt‘, ‘.png‘, ‘.jpg‘, etc.) is auto-detected from PMTiles metadata - **TileJSON**: ‘http://localhost:PORT/TILESET.json‘ (requires ‘public_url‘) - Returns TileJSON metadata for the tileset

This approach works with any map client and is particularly useful for: - Large PMTiles files (multi-GB) - Serving directly from cloud storage - Clients that don’t support the PMTiles protocol

Cloud Storage

You can serve tiles directly from cloud storage without downloading:

Cloudflare R2: “‘r pm_serve_zxy(bucket = "s3://my-bucket?endpoint=https://account.r2.cloudflarestorage.com&region) “‘

AWS S3: “‘r pm_serve_zxy(bucket = "s3://my-bucket") “‘

Requires appropriate environment variables for authentication (‘AWS_ACCESS_KEY_ID‘, ‘AWS_SECRET_ACCESS_KEY

Background Mode

When ‘background = TRUE‘, the server runs in a background process. Use ‘pm_stop_server()‘ to stop background servers.

Comparison with pm_serve()

- ‘pm_serve()‘: Serves raw .pmtiles files for PMTiles.js (HTTP Range requests) - ‘pm_serve_zxy()‘: Serves individual Z/X/Y tiles for any map client

Use ‘pm_serve_zxy()‘ when you need standard tile URLs or want to serve from cloud storage. Use ‘pm_serve()‘ for quick local preview with PMTiles.js.

Value

If ‘background = FALSE‘, blocks until server is stopped (Ctrl+C). If ‘background = TRUE‘, invisibly returns the ‘processx::process‘ object.

Examples

```
## Not run:
# Serve all PMTiles in current directory
pm_serve_zxy()

# Serve specific directory on custom port
pm_serve_zxy(path = "~/pmtiles", port = 9000)

# Serve from Cloudflare R2
pm_serve_zxy(
 bucket = "s3://my-bucket?endpoint=https://account.r2.cloudflarestorage.com&region=auto",
  public_url = "https://tiles.example.com"
)

# Run in background
server <- pm_serve_zxy(background = TRUE)
# ... do other work ...
pm_stop_server(server)

## End(Not run)
```

pm_show *Show PMTiles archive information*

Description

Inspect a local or remote PMTiles archive and display header information and metadata.

Usage

```
pm_show(
  input,
  bucket = NULL,
  metadata = FALSE,
  header_json = FALSE,
  tilejson = FALSE,
  public_url = NULL
)
```

Arguments

input	Path to a local PMTiles file or URL to a remote archive. Remote archives can be HTTP URLs or cloud storage paths.
bucket	Optional remote bucket specification for cloud storage (e.g., "s3://bucket-name"). See Details for cloud storage usage.
metadata	Logical. If ‘TRUE‘, return only the JSON metadata. Default is ‘FALSE‘.
header_json	Logical. If ‘TRUE‘, return only the header as JSON. Default is ‘FALSE‘.
tilejson	Logical. If ‘TRUE‘, return TileJSON specification. Default is ‘FALSE‘.
public_url	Character. Public base URL for TileJSON (e.g., "https://example.com/tiles"). Only used when ‘tilejson = TRUE‘.

Details

Cloud Storage

PMTiles supports reading from cloud storage buckets: - **S3**: ‘bucket = "s3://BUCKET_NAME"‘ - **S3-compatible** (R2, etc.): ‘bucket = "s3://BUCKET?endpoint=https://example.com&region=auto"‘ - **Azure**: ‘bucket = "azblob://CONTAINER?storage_account=ACCOUNT"‘ - **Google Cloud**: ‘bucket = "gs://BUCKET_NAME"‘

Authentication uses standard cloud provider environment variables (e.g., ‘AWS_ACCESS_KEY_ID‘, ‘AWS_SECRET_ACCESS_KEY‘ for S3).

Value

- If ‘metadata = TRUE‘ or ‘header_json = TRUE‘ or ‘tilejson = TRUE‘: Returns a parsed list from the JSON output - Otherwise: Returns invisible ‘NULL‘ and prints archive information

Examples

```
## Not run:
# Show local archive info
pm_show("path/to/archive.pmtiles")

# Get metadata as list
metadata <- pm_show("archive.pmtiles", metadata = TRUE)

# Show remote archive
pm_show(
  "archive.pmtiles",
 bucket = "s3://my-bucket?endpoint=https://account.r2.cloudflarestorage.com&region=auto"
)

# Get TileJSON
tilejson <- pm_show(
  "archive.pmtiles",
  tilejson = TRUE,
  public_url = "https://example.com/tiles"
)

## End(Not run)
```

`pm_stop_server` *Stop a background PMTiles server*

Description

Stop a PMTiles server that was started with ‘pm_serve()‘ or ‘pm_serve_zxy()‘.

Usage

```
pm_stop_server(server = NULL)
```

Arguments

`server`	A server object returned by ‘pm_serve()‘ or ‘pm_serve_zxy()‘, or a port number. If ‘NULL‘, stops all running PMTiles servers.

Value

Invisibly returns ‘TRUE‘ if server was stopped, ‘FALSE‘ otherwise.

See Also

[pm_serve()], [pm_serve_zxy()]

Examples

```
## Not run:
# Start servers
server1 <- pm_serve("data.pmtiles")
server2 <- pm_serve_zxy(background = TRUE)

# Stop specific server
pm_stop_server(server1)

# Or stop by port
pm_stop_server(8080)

# Or stop all servers
pm_stop_server()

## End(Not run)
```

`pm_tile` *Extract a single tile from a PMTiles archive*

Description

Fetch one tile from a local or remote PMTiles archive and save it to a file.

Usage

```
pm_tile(input, z, x, y, output = NULL, bucket = NULL)
```

Arguments

`input`	Path to a local PMTiles file or URL to a remote archive.
`z`	Integer zoom level.
`x`	Integer tile column.
`y`	Integer tile row.
`output`	Path where the tile should be saved. If 'NULL' (default), returns the raw tile data as a raw vector.
`bucket`	Optional remote bucket specification for cloud storage.

Value

If 'output' is specified, writes tile to file and returns the output path invisibly. If 'output' is 'NULL', returns the tile data as a raw vector.

Examples

```
## Not run:
# Get tile data
tile_data <- pm_tile("archive.pmtiles", z = 0, x = 0, y = 0)

# Save tile to file
pm_tile("archive.pmtiles", z = 5, x = 10, y = 12, output = "tile.mvt")

## End(Not run)
```

`pm_upload` *Upload PMTiles archive to cloud storage*

Description

Upload a local PMTiles archive to cloud storage (S3, Azure, Google Cloud). Requires appropriate authentication via environment variables.

Usage

```
pm_upload(input, remote, bucket, max_concurrency = 2, verbose = TRUE)
```

Arguments

`input`	Path to local PMTiles file to upload.
`remote`	Name for the PMTiles file in cloud storage.
`bucket`	Bucket specification (e.g., "s3://bucket-name"). See Details.
`max_concurrency`	Maximum number of parallel upload threads. Default is 2.
`verbose`	Logical. If ‘TRUE‘, prints progress information. Default is ‘TRUE‘.

Details

Authentication

PMTiles uses standard cloud provider authentication methods:

AWS S3: - Set ‘AWS_ACCESS_KEY_ID‘ and ‘AWS_SECRET_ACCESS_KEY‘ environment variables - Requires write permissions to the bucket

S3-compatible (R2, MinIO, etc.): - Same credentials as S3 - Specify endpoint in bucket parameter: ‘bucket = "s3://bucket?endpoint=https://account.r2.cloudflarestorage.com&region=auto"‘

Azure Blob: - Uses Azure SDK default authentication - ‘bucket = "azblob://container?storage_account=ACCOUNT"‘

Google Cloud Storage: - Uses Application Default Credentials - ‘bucket = "gs://bucket-name"‘

Value

Invisibly returns ‘TRUE‘ on success.

Examples

```
## Not run:
# Set credentials
Sys.setenv(
  AWS_ACCESS_KEY_ID = "your-key-id",
  AWS_SECRET_ACCESS_KEY = "your-secret"
)

# Upload to S3
pm_upload(
  "local.pmtiles",
  "remote.pmtiles",
  bucket = "s3://my-bucket"
)

# Upload to Cloudflare R2
pm_upload(
  "local.pmtiles",
  "remote.pmtiles",
 bucket = "s3://my-bucket?endpoint=https://account.r2.cloudflarestorage.com&region=auto"
)

## End(Not run)
```

`pm_verify` *Verify PMTiles archive structure*

Description

Check that a PMTiles archive is ordered correctly and has correct header information. This verifies the archive structure without checking individual tile contents.

Usage

```
pm_verify(input)
```

Arguments

`input`	Path to a local PMTiles file.

Value

Invisibly returns ‘TRUE‘ if verification succeeds, throws an error otherwise.

Examples

```
## Not run:
# Verify an archive
pm_verify("archive.pmtiles")

## End(Not run)
```

`pm_version` *Get PMTiles CLI version*

Description

Display the version information of the PMTiles command-line tool.

Usage

```
pm_version()
```

Value

Character string containing version information.

Examples

```
## Not run:
pm_version()

## End(Not run)
```

`pm_view` *Quick viewer for PMTiles archives*

Description

Quickly visualize a PMTiles archive or TileJSON endpoint on an interactive map using mapgl. Automatically detects the tile type and applies appropriate styling. For local files, automatically starts a background server.

Usage

```
pm_view(
  input,
  source_layer = NULL,
  style = NULL,
  port = 8080,
  layer_type = c("auto", "fill", "line", "circle"),
  fill_color = "#088",
  fill_opacity = 0.5,
  line_color = "#088",
  line_width = 1,
  circle_color = "#088",
  circle_radius = 5,
  inspect_features = FALSE,
  highlight = FALSE,
  promote_id = NULL,
  ...
)
```

Arguments

input	Path to a local PMTiles file, URL to a remote archive, or TileJSON endpoint URL.
source_layer	Name of the source layer to display. If 'NULL' (default), automatically uses the first layer found in the metadata.
style	Base map style. Can be a mapgl style function like 'mapgl::openfreemap_style("positron")' or a style name. Default is '"positron"'.
port	Port for local server (only used for local files). Default is 8080.
layer_type	Type of layer to add: '"fill"', '"line"', '"circle"', or '"auto"'. Default is '"auto"' which detects based on geometry type.
fill_color	Fill color for polygons. Default is '"#088"'.
fill_opacity	Fill opacity. Default is 0.5.
line_color	Line color. Default is '"#088"'.
line_width	Line width. Default is 1.
circle_color	Circle color for points. Default is '"#088"'.
circle_radius	Circle radius for points. Default is 5.
inspect_features	Logical. If 'TRUE', shows tooltips with feature attributes. Default is 'FALSE'.
highlight	Logical. If 'TRUE', enables hover highlighting effects. Requires the tileset to have an ID property configured. Default is 'FALSE'.
promote_id	Optional. Column name to promote as the feature ID for hover highlighting. Only used when 'highlight = TRUE'. If 'NULL', assumes the tileset already has an ID property configured during tile generation.
...	Additional arguments passed to the layer function.

Details

Local vs Remote Files

Local files: If ‘input‘ is a local file path, ‘pm_view()‘ automatically: 1. Starts a background PMTiles server via ‘pm_serve()‘ 2. Extracts metadata to determine source layer and geometry type 3. Creates an appropriate map visualization

The server runs in the background and will be automatically stopped when the R session ends. You can manually stop it with ‘pm_stop_server()‘.

Remote files: If ‘input‘ is a URL (starts with ‘http://‘ or ‘https://‘), uses the URL directly without starting a local server.

TileJSON endpoints: If ‘input‘ ends with ‘.json‘, it’s treated as a TileJSON endpoint. The function fetches the TileJSON metadata and uses ‘add_vector_source()‘ to display the tiles.

Geometry Detection

When ‘layer_type = "auto"‘, the function inspects the metadata to determine the geometry type and applies appropriate styling: - Polygon/MultiPolygon → fill layer - LineString/MultiLineString → line layer - Point/MultiPoint → circle layer

Feature Inspection and Highlighting

Set ‘inspect_features = TRUE‘ to show tooltips with feature attributes on hover. This works for any tileset.

Set ‘highlight = TRUE‘ to enable hover highlighting effects (features change color/style when hovered). **Important**: This requires the tileset to have an ID property configured. You can either: - Configure IDs during tile generation (recommended) - Use ‘promote_id = "column_name"‘ to promote an existing attribute as the ID

If the tileset lacks proper IDs, highlighting will not work even if enabled.

Value

A mapgl map object.

Examples

```
## Not run:
# View a local PMTiles file
pm_view("data.pmtiles")

# Customize the display
pm_view("data.pmtiles",
        fill_color = "#ff0000",
        fill_opacity = 0.7)

# View remote PMTiles
pm_view("https://example.com/tiles.pmtiles")

# View TileJSON endpoint
pm_view("http://localhost:8080/tiles.json")

# Use specific source layer
pm_view("data.pmtiles", source_layer = "buildings")
```

```
# Enable feature inspection with tooltips
pm_view("data.pmtiles", inspect_features = TRUE)

# Enable hover highlighting (requires ID property in tileset)
pm_view("data.pmtiles",
        inspect_features = TRUE,
        highlight = TRUE,
        promote_id = "id")

# Further customize with mapgl
pm_view("data.pmtiles") |>
  mapgl::fit_bounds(c(-122.5, 37.7, -122.3, 37.9))

## End(Not run)
```

Index