Package: GiottoIngest (via r-universe)

November 17, 2026

Title Rust Ingest Backend for GiottoDisk Stores

Version 0.1.0

Description Prototype Rust backend that reads spatial omics inputs fast and writes GiottoDisk point tile stores, reproducing the R writer's tiles, rows and column types. One shared core (per-tile row order, tile layout, parquet writer) with one module per input format: parquet (Xenium, Atera preview, MERSCOPE, bin points) and zarr (Atera v1). The tile plan stays tilework's own; only its counting moves to Rust.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/rextendr/version 0.5.0

SystemRequirements Cargo (Rust's package manager), rustc >= 1.81.0

Depends R (>= 4.2)

Imports methods, tilework

Suggests arrow, DBI, duckdb, GiottoDisk, terra, dplyr, future, rlang, testthat (>= 3.0.0)

Remotes drieslab/tilework

Config/roxygen2/version 8.1.1

Config/testthat/edition 3

Config/pak/sysreqs libgdal-dev gdal-bin libgeos-dev libproj-dev libsqlite3-dev libclang-dev

Repository https://giotto-suite.r-universe.dev

Date/Publication 2026-11-17 16:42:52 UTC

RemoteUrl https://github.com/giotto-suite/GiottoIngest

RemoteRef HEAD

RemoteSha c42c2342b8a15479e38f47c42ec5b302f0792a88

Contents

gi_atera_tx_cells 2
gi_capabilities 3
gi_count_tiles 4
gi_extent 4
gi_route_polygons 5
gi_source 5
gi_tileApply 6
gi_write_point_tiles 6
gi_write_points 7

Index **9**

`gi_atera_tx_cells` *Stream Atera grid cells into tile partitions*

Description

The zarr input: Rust counterpart of GiottoDisk's per-grid-cell Atera v1 transcript writer (`.atera_tx_cell_worker()`). The caller supplies GiottoDisk's Atera tile plan and lookups (`.atera_tx_tile_plan()`, `.atera_tx_lookups()`). Each grid cell is decoded, routed to its `s` x `s` store tiles and written as `tile_index=<k>/part-NNNNN-0.parquet` under `root`; `threads` cells are decoded at once in this process.

Usage

```
gi_atera_tx_cells(
  zarr_path,
  tiles,
  ords,
  s,
  tile_index,
  gene_names,
  cw_gene,
  root,
  qv_threshold = 20,
  flip = TRUE,
  window_rows = 4194304,
  buffer_rows = 4194304,
  threads = 1L
)
```

Arguments

`zarr_path`	path to `transcripts.zarr.zip`
`tiles`	grid cell names (`"i,j"`), largest first
`ords`	zero-based grid cell ordinals, for `transcript_id`

s	tiles per side of each grid cell's split
tile_index	list of store tile indices per grid cell (s^2 each)
gene_names, cw_gene	the archive's gene names and codeword to gene mapping
root	directory the tile_index= partitions are written under
qv_threshold	minimum quality score kept (NULL keeps all)
flip	negate y
window_rows	rows decoded per read
buffer_rows	rows buffered per grid cell before a write
threads	grid cells decoded at once

Value

one list per grid cell: tile_index, counts, and bounds (s^2 x 4 matrix, xmin xmax ymin ymax)

---

gi_capabilities *Capabilities of this build*

Description

What this GiottoIngest build can do. GiottoDisk gates each fast path on its capability, so an older build without it falls back to R:

Usage

```
gi_capabilities()

has_capability(name)
```

Arguments

name	capability name

Details

- "points_parquet": point tile stores from parquet sources (gi_source(), gi_write_points()).
- "points_atera_zarr": Atera v1 transcripts.zarr.zip grid cells (gi_atera_tx_cells()).

Value

gi_capabilities(): character vector; has_capability(): TRUE or FALSE

gi_count_tiles *Rows of a point source per tile*

Description

Tile membership follows GiottoDisk's contiguous `getTile()` over the given set of tiles.

Usage

```
gi_count_tiles(src, bounds, threads = 1L)
```

Arguments

`src`	from `gi_source()`
`bounds`	matrix, one row per tile: xmin, xmax, ymin, ymax
`threads`	decoding threads

Value

numeric counts, one per tile

gi_extent *Rows and extent of a point source*

Description

Rows and extent of a point source

Usage

```
gi_extent(src, threads = 1L)
```

Arguments

`src`	from `gi_source()`
`threads`	decoding threads

Value

`list(n, ext)`, `ext` as xmin, xmax, ymin, ymax

gi_route_polygons *Route polygon vertices to tiles*

Description

Each polygon goes to the tile holding its envelope centroid, as GiottoDisk's polygon tile write selects it; its vertex rows are written to one parquet file per tile in `row_index` order.

Usage

```
gi_route_polygons(src, bounds, group, out_dir, tile_files, threads = 1L)
```

Arguments

`src`	from `gi_source()`, over the vertex rows
`bounds`	matrix of the tiles written (non-empty plan tiles)
`group`	polygon id column
`out_dir`	directory for the per-tile files
`tile_files`	file name per tile
`threads`	threads

Value

`list(counts, groups, unassigned)`: vertex rows per tile, polygons, and polygons whose centroid fell in no tile

gi_source *Describe a parquet point source*

Description

The lazy query a GiottoDisk reader builds, reduced to what the Rust passes need: files, coordinate columns, a y flip and numeric row filters.

Usage

```
gi_source(files, x, y, flip_y = FALSE, filters = list())
```

Arguments

`files`	parquet file paths, in the order the reader scans them
`x, y`	coordinate columns
`flip_y`	negate y
`filters`	list of `list(col, op, value)` on numeric columns; `op` one of `>=`, `>`, `<=`, `<`, `==`, `!=`, or `in` with a vector `value` (R's `%in%`). Rows where the column is NA are dropped, as in arrow.

Value

a list used by the `gi_*` functions

`gi_tileApply` *Tile planning with tilework, counted in Rust*

Description

A `tileApply()` method for `gi_source()` objects: every tile of a pass is counted in one Rust scan of the source, under GiottoDisk's contiguous `getTile()` edge rules. With it, tilework's own `tilework::quadtreePlan()` plans a point store from a source, so the plan is GiottoDisk's by construction:

Arguments

`x`	a `gi_source()`
`tiles`	a `freeTilePlan`
`threads`	decoding threads
`...`	ignored

Details

```
src <- gi_source(files, "x_location", "y_location", flip_y = TRUE)
plan <- tilework::quadtreePlan(src, tiles = seed, threshold = threshold,
    threads = 8L)
```

`seed` is the starting `tilePlan` GiottoDisk builds from the source extent (`gi_extent()`). Only counting is supported: `FUN` is ignored, and `sdimx`, `sdimy` and `contiguous` (the source carries its coordinates; the edge rules are always the contiguous ones).

Value

a list with one count per tile

`gi_write_point_tiles` *Write a point tile store's tiles*

Description

Write a point tile store's tiles

Usage

```
gi_write_point_tiles(
  src,
  bounds,
  tile_dirs,
  id_col,
  to_string = character(),
  drop = character(),
  root,
  geo_json,
  spill_rows = 4e+06,
  geom = TRUE,
  threads = 1L
)
```

Arguments

src	from gi_source()
bounds	matrix of the tiles to write (non-empty plan tiles)
tile_dirs	directory name of each tile under root
id_col	feature id column, written as feat_ID
to_string	columns cast to string, as the reader's schema does
drop	columns left out
root	directory the tiles are written under
geo_json	geoparquet metadata written in every tile
spill_rows	rows buffered in memory before spilling to disk
geom	write the WKB geom column and geoparquet metadata, as GiottoDisk's point stores have them. FALSE writes coordinates only (x_index, y_index, and z among the metadata columns): faster and smaller, but not yet a format GiottoDisk reads.
threads	threads

Value

numeric rows per tile

---

gi_write_points *Write a point tile store from a plan*

Description

The one call a reader needs after planning: writes every non-empty tile of plan as GiottoDisk's point store writer does, into root/tile_index=<k>/part-0.parquet with k the tile's index in the plan (empty tiles leave gaps, as in GiottoDisk), and the store's geoparquet metadata in every tile.

Usage

```
gi_write_points(
  src,
  plan,
  root,
  id_col,
  to_string = character(),
  drop = character(),
  geom = TRUE,
  threads = 1L,
  spill_rows = 4e+06
)
```

Arguments

src	from gi_source()
plan	a freeTilePlan from tilework::quadtreePlan() (with its n_records), or a list with bounds (one row per tile: xmin, xmax, ymin, ymax) and n (rows per tile)
root	the store's source_id= directory the tiles go under
id_col	feature id column, written as feat_ID
to_string	columns cast to string, as the reader's schema does
drop	columns left out
geom	write the WKB geom column and geoparquet metadata (GiottoDisk's current point format)
threads	threads
spill_rows	rows buffered in memory before spilling to disk

Details

Only the non-empty tiles are passed on, as R writes them: the contiguous edge rules (which tiles' right and bottom edges are inclusive) are taken over that set.

Value

list(tile_index, counts): the plan indices written and their row counts

Index

gi_atera_tx_cells, 2
gi_atera_tx_cells(), *3*
gi_capabilities, 3
gi_count_tiles, 4
gi_extent, 4
gi_extent(), *6*
gi_route_polygons, 5
gi_source, 5
gi_source(), *3–8*
gi_tileApply, 6
gi_write_point_tiles, 6
gi_write_points, 7
gi_write_points(), *3*

has_capability *(*gi_capabilities*)*, 3

tileApply,gi_source,missing,ANY-method
        *(*gi_tileApply*)*, 6
tilework::quadtreePlan(), *6*, *8*