

Package: lazysf (via r-universe)

October 11, 2026

Title Delayed Read for 'GDAL' Vector Data Sources

Version 0.4.0

Description Lazy read for drawings. A 'dplyr' back end for data sources supported by 'GDAL' vector drivers, that allows working with local or remote sources as if they are in-memory data frames. Basic features work with any drawing format ('GDAL vector data source') supported by the 'gdalraster' package.

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 8.0.0

Depends R (>= 4.1.0)

Imports gdalraster (>= 2.0.0), methods, DBI, tibble, dbplyr (>= 2.0.0), dplyr, wk

URL <https://github.com/hypertidy/lazysf>,
<https://hypertidy.github.io/lazysf/>

BugReports <https://github.com/hypertidy/lazysf/issues>

Suggests bit64, nanoarrow, testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

VignetteBuilder knitr

Collate 'GDALVectorResult.R' 'GDALVectorConnection.R'
'GDALVectorDriver.R' 'connect.R' 'dbplyr.R' 'helper.R'
'lazysf-package.R' 'lazysf.R' 'sql-query-fields.R'
'sql-translation.R' 'zzz.R'

Config/pak/sysreqs libgdal-dev gdal-bin libgeos-dev libicu-dev libxml2-dev libzstd-dev

Repository <https://hypertidy.r-universe.dev>

Date/Publication 2026-07-12 23:01:07 UTC

RemoteUrl <https://github.com/hypertidy/lazysf>

RemoteRef HEAD

RemoteSha 4058b28e3b7666089fa4a5023d7216d66729e9dd

Contents

collect	2
dbConnect,GDALVectorDriver-method	3
GDALSQL	4
lazysf	4
Index	7

collect	<i>Collect a lazy GDAL query into memory</i>
---------	--

Description

Re-export of `dplyr::collect()`. Forces evaluation of a lazy `tbl_GDALVectorConnection` and returns the result as a local tibble. Geometry columns are returned as wk-typed vectors (`wk::wkb`, `wk::wkt`, or `wk::rct`) with CRS attached, and can be passed to `sf::st_as_sf()` for conversion.

Arguments

- `x` A `tbl_GDALVectorConnection` created by `lazysf()`.
- `...` Additional arguments passed to `dplyr::collect()`.

Value

A local tibble.

See Also

`lazysf()`, `dplyr::collect()`

Examples

```
f <- system.file("extdata/multi.gpkg", package = "lazysf", mustWork = TRUE)
lsf <- lazysf(f)
dplyr::collect(lsf)
```

dbConnect, GDALVectorDriver-method
dbConnect

Description

dbConnect for vector data sources readable by GDAL

Usage

```
## S4 method for signature 'GDALVectorDriver'
dbConnect(
  drv,
  DSN = "",
  readonly = TRUE,
  geom_format = getOption("lazysf.geom_format", "WKB"),
  dialect = getOption("lazysf.dialect", "SQLITE"),
  use_arrow = getOption("lazysf.use_arrow", FALSE),
  ...
)
```

Arguments

drv	GDALVectorDriver created by GDALSQL()
DSN	data source name, may be a file, or folder path, database connection string, or URL
readonly	open in readonly mode (TRUE is the only option currently)
geom_format	geometry output format: "WKB" (default), "WKT", "NONE", or "BBOX" (alias "RCT"). Case-insensitive.
dialect	SQL dialect: "SQLITE" (default), "OGRSQL", "INDIRECT_SQLITE", or "" (let GDAL choose). SQLITE is recommended as it supports subqueries (required for dbplyr) and spatial SQL functions.
use_arrow	logical; if TRUE, use GDAL's Arrow C stream interface for reading features (GDAL >= 3.6). Columnar transfer via nanoarrow, typically much faster for larger datasets. Default FALSE.
...	ignored

Details

The 'OGRSQL' available is documented with GDAL: https://gdal.org/en/stable/user/ogr_sql_sqlite_dialect.html

Examples

```
f <- system.file("extdata/multi.gpkg", package = "lazysf", mustWork = TRUE)
db <- dbConnect(GDALSQL(), f)
dbListTables(db)
```

GDALSQL

GDALSQL

Description

GDALSQL driver, use with `DBI::dbConnect()` to open a data source readable by GDAL

Usage

`GDALSQL()`

See Also

`lazysf dbConnect`

Examples

`GDALSQL()`

lazysf

Delayed (lazy) read for GDAL vector

Description

A lazy data frame for GDAL drawings ('vector data sources'). lazysf is DBI compatible and designed to work with dplyr. It should work with any data source (file, url, connection string) readable by GDAL via the gdalraster package.

Usage

`lazysf(x, layer, ...)`

S3 method for class 'character'

```
lazysf(
  x,
  layer,
  ...,
  query = NA,
  geom_format = getOption("lazysf.geom_format", "WKB"),
  dialect = getOption("lazysf.dialect", "SQLITE"),
  use_arrow = getOption("lazysf.use_arrow", FALSE)
)
```

S3 method for class 'GDALVectorConnection'

```
lazysf(x, layer, ..., query = NA)
```

Arguments

<code>x</code>	the data source name (file path, url, or database connection string • analogous to a GDAL dsn) or a <code>GDALVectorConnection</code>
<code>layer</code>	layer name; defaults to the first layer
<code>...</code>	ignored
<code>query</code>	SQL query to pass in directly
<code>geom_format</code>	geometry output format, passed to <code>dbConnect()</code>
<code>dialect</code>	SQL dialect, passed to <code>dbConnect()</code>
<code>use_arrow</code>	logical; if TRUE, use GDAL's Arrow C stream interface for reading features. Passed to <code>dbConnect()</code> .

Details

Lazy means that the usual behaviour of reading the entirety of a data source into memory is avoided. Printing the output results in a preview query being run and displayed (the top few rows of data).

The output of `lazysf()` is a `'tbl_GDALVectorConnection'` that extends `tbl_dbf` and may be used with functions and workflows in the normal DBI way, see [GDALSQL\(\)](#) for the lazysf DBI support.

The kind of query that may be run will depend on the type of format, see the list on the GDAL vector drivers page. For some details see the [GDALSQL vignette](#).

When `dplyr` is attached the lazy data frame can be used with the usual verbs (filter, select, distinct, mutate, transmute, arrange, left_join, pull, collect etc.). To see the result as a SQL query rather than a data frame preview use `dplyr::show_query()`.

To obtain an in memory data frame use an explicit `collect()`. Geometry columns in the result are wk-typed vectors (`wk::wkb`, `wk::wkt`, or `wk::rct`) with CRS attached. To convert to an sf data frame, collect first then call `sf::st_as_sf()`: `lazysf(dsn) |> collect() |> sf::st_as_sf()`.

As well as `collect()` it's also possible to use `tibble::as_tibble()` or `as.data.frame()` or `pull()` which all force computation and retrieve the result.

Value

a `'tbl_GDALVectorConnection'`, extending `'tbl_lazy'` (something that works with `dplyr` verbs, and only shows a preview until you commit the result via `collect()`) see Details

Examples

```
## a multi-layer file
f <- system.file("extdata/multi.gpkg", package = "lazysf", mustWork = TRUE)
lazysf(f)

## Geopackage (an actual database, so with SELECT we must be explicit re geom-column)
nc <- system.file("extdata/nc.gpkg", package = "lazysf", mustWork = TRUE)
lazysf(nc)
lazysf(nc, query = "SELECT AREA, FIPS, geom FROM nc WHERE AREA < 0.1")
lazysf(nc, layer = "nc") |> dplyr::select(AREA, FIPS, geom) |> dplyr::filter(AREA < 0.1)
```

```
## the famous ESRI Shapefile (not an actual database)
shdb <- system.file("extdata/nc.shp", package = "lazysf", mustWork = TRUE)
shp <- lazysf(shdb)
library(dplyr)
shp |>
  filter(NAME %LIKE% 'A%') |>
  mutate(abc = 1.3) |>
  select(abc, NAME) |>
  arrange(desc(NAME))
```

Index

`collect`, [2](#)
`collect()`, [5](#)

`dbConnect()`, [5](#)
`dbConnect`, `GDALVectorDriver`-method, [3](#)
`DBI::dbConnect()`, [4](#)
`dplyr::collect()`, [2](#)

`GDALSQL`, [4](#)
`GDALSQL()`, [5](#)

`lazysf`, [4](#)
`lazysf()`, [2](#)