Package: gdalvector (via r-universe)

October 26, 2026

Title Modern GDAL Vector Data Workflows

Version 0.0.3

Description Opinionated system for working with modern geospatial vector data formats using GDAL.

License MIT + file LICENSE

URL https://github.com/jimbrig/gdalvector, http://docs.jimbrig.com/gdalvector/

BugReports https://github.com/jimbrig/gdalvector/issues

Depends R (>= 4.2)

Imports bit64, cli, climenu, crayon, DBI, dplyr, fs, gdalraster, geos, glue, gpkg, httr2, jsonvalidate, nanoparquet, openssl, processx, ps, purrr, rlang, RSQLite, rvest, sf, stats, stringr, terra, tibble, tidyr, tidyselect, tools, utils, vapour, withr, xml2, yyjsonr

Suggests knitr, rmarkdown, spelling, testthat (>= 3.1.0)

VignetteBuilder knitr

Config/Needs/website quarto

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 8.0.0

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev make libicu-dev libuv1-dev libxml2-dev libzstd-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev libnode-dev

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

Date/Publication 2026-07-02 15:37:53 UTC

RemoteUrl https://github.com/jimbrig/gdalvector

RemoteRef HEAD

RemoteSha f7daa62b8b9c2d8588b32e07b6e924fa1d022b82

Contents

as_config_option . . . 4
as_gdal_args . . . 4
as_gdal_config_opts . . . 6
as_gdal_creation_opts . . . 7
as_gdal_open_opts . . . 8
as_gdal_vsi_opts . . . 9
as_gdalg . . . 10
check_available_ram . . . 11
check_inherits . . . 11
check_not_empty . . . 13
check_string . . . 13
checks . . . 14
fgb_config_opts . . . 14
fgb_creation_opts . . . 15
fgb_open_opts . . . 16
fgb_validate_spatial_index_ram . . . 17
gdal_config . . . 17
gdal_config_links . . . 18
gdal_config_opts . . . 18
gdal_creation_opts . . . 19
gdal_drivers . . . 19
gdal_open_opts . . . 20
gdal_opts . . . 21
gdal_render . . . 22
gdal_vector_driver_capabilities . . . 22
gdal_vector_driver_config_opts . . . 23
gdal_vector_driver_creation_opts . . . 24
gdal_vector_driver_open_opts . . . 25
gdal_vector_driver_opts . . . 26
gdal_vsi_opts . . . 28
gdb_config_opts . . . 28
gdb_creation_opts . . . 29
gdb_open_opts . . . 31
gpkg_config_opts . . . 32
gpkg_creation_opts . . . 33
gpkg_open_opts . . . 35
gpkg_prelude_pragmas . . . 37
gpq_creation_opts . . . 38
gpq_file_info . . . 40
gpq_geo_metadata . . . 41
gpq_inspect . . . 41
gpq_meta . . . 42
gpq_open_opts . . . 42
gpq_row_groups . . . 43
gpq_schema . . . 44
gpq_schema_info . . . 45

is_vsi_path . . . 45
local_hash . . . 46
ping . . . 46
read_renviron . . . 47
remote_download . . . 47
remote_exists . . . 48
remote_hash . . . 49
remote_head . . . 49
remote_last_modified . . . 50
remote_list . . . 51
remote_size . . . 52
shp_config_opts . . . 52
shp_creation_opts . . . 54
shp_open_opts . . . 55
sql_where_valid_geom . . . 57
sys_available_ram . . . 57
sys_error_code . . . 58
sys_num_cpus . . . 59
sys_os . . . 60
sys_path . . . 60
sys_pid . . . 61
sys_platform . . . 62
sys_which . . . 62
validate_gdal_opts . . . 63
vsi_azure . . . 64
vsi_curl . . . 64
vsi_exists . . . 65
vsi_from_uri . . . 65
vsi_handlers . . . 66
vsi_meta . . . 66
vsi_path . . . 67
vsi_size . . . 67
vsi_strip . . . 68
vsi_sync . . . 68
vsi_type . . . 69
vsi_zip . . . 69
vsi_zip_curl . . . 70
xml_parse_gdal_driver_config_opts . . . 70
xml_parse_gdal_options . . . 71

Index **73**

as_config_option *Convert GDAL Configuration Options to a Config-Option Vector*

Description

Render a gdal_config_opts() or gdal_vsi_opts() object to a named character vector c(NAME = "VALUE"), the form consumed by gdalraster::set_config_option(). Configuration options are ignored by the GDAL algorithm API and must be applied to the process/session this way (or via the CLI --config flag, see as_gdal_args()).

Usage

```
as_config_option(x, ...)

## S3 method for class 'gdal_config_opts'
as_config_option(x, ...)

## S3 method for class 'gdal_vsi_opts'
as_config_option(x, ...)

## Default S3 method:
as_config_option(x, ...)
```

Arguments

x	A gdal_config_opts() or gdal_vsi_opts() object.
...	Passed to methods.

Value

A named character vector.

Examples

```
as_config_option(gdal_config_opts(CPL_DEBUG = "ON"))
```

as_gdal_args *Convert GDAL Options to Algorithm Arguments*

Description

Render a gdal_opts() object to the form consumed by the GDAL algorithm API (gdalraster::gdal_alg() / gdalraster::gdal_run()).

Usage

```
as_gdal_args(x, ...)

## S3 method for class 'gdal_opts'
as_gdal_args(x, cli = TRUE, long = FALSE, with_format = FALSE, ...)

## S3 method for class 'character'
as_gdal_args(x, ...)

## S3 method for class 'list'
as_gdal_args(x, ...)

## Default S3 method:
as_gdal_args(x, ...)
```

Arguments

x	A gdal_opts() object (or a character vector / list, passed through).
...	Passed to methods.
cli	Logical; emit interleaved CLI tokens (TRUE, default) or a bare KEY=VALUE vector.
long	Logical; use long flag names (--open-option) rather than aliases (--oo).
with_format	Logical; prepend the --input-format/--output-format flag and driver when known (open/creation only).

Details

For repeated options (--oo/--co/--lco), GDAL requires each value to be preceded by its own flag - values are never comma-packed (a packed value would corrupt options such as PRELUDE_STATEMENTS that themselves contain ;/,). Accordingly:

- cli = TRUE (default) emits a flat token vector c("--open-option", "K=V", "--open-option", "K2=V2", ...), suitable as the args to gdalraster::gdal_alg().
- cli = FALSE emits an unnamed c("K=V", ...) vector, suitable for a single alg$setArg(<flag>, .) call.

Value

A character vector.

Examples

```
as_gdal_args(gdal_open_opts(LIST_ALL_TABLES = FALSE, driver = "GPKG"))
as_gdal_args(gdal_open_opts(LIST_ALL_TABLES = FALSE), cli = FALSE)
```

`as_gdal_config_opts` *Coerce to GDAL Configuration Options*

Description

Coerce a named list, a `KEY=VALUE` character vector, a driver-metadata tibble, or an existing `gdal_config_opts` to the `gdal_config_opts()` class.

Usage

```
as_gdal_config_opts(x, ..., driver = NULL, call = rlang::caller_env())

## Default S3 method:
as_gdal_config_opts(x, ..., driver = NULL, call = rlang::caller_env())

## S3 method for class 'gdal_config_opts'
as_gdal_config_opts(x, ..., driver = NULL, call = rlang::caller_env())

## S3 method for class 'list'
as_gdal_config_opts(x, ..., driver = NULL, call = rlang::caller_env())

## S3 method for class 'character'
as_gdal_config_opts(x, ..., driver = NULL, call = rlang::caller_env())

## S3 method for class 'tbl_df'
as_gdal_config_opts(x, ..., driver = NULL, call = rlang::caller_env())
```

Arguments

`x`	Object to coerce.
`...`	Unused; for method extensibility.
`driver`	Optional GDAL driver short name to attach.
`call`	The execution environment of a currently running function, e.g. `caller_env()`. The function will be mentioned in error messages as the source of the error. See the `call` argument of `abort()` for more information.

Value

A `gdal_config_opts()` object.

as_gdal_creation_opts *Coerce to GDAL Creation Options*

Description

Coerce a named list, a KEY=VALUE character vector, a driver-metadata tibble, or an existing gdal_creation_opts to the gdal_creation_opts() class.

Usage

```
as_gdal_creation_opts(
  x,
  ...,
  driver = NULL,
  level = c("layer", "dataset"),
  call = rlang::caller_env()
)

## Default S3 method:
as_gdal_creation_opts(
  x,
  ...,
  driver = NULL,
  level = c("layer", "dataset"),
  call = rlang::caller_env()
)

## S3 method for class 'gdal_creation_opts'
as_gdal_creation_opts(
  x,
  ...,
  driver = NULL,
  level = c("layer", "dataset"),
  call = rlang::caller_env()
)

## S3 method for class 'list'
as_gdal_creation_opts(
  x,
  ...,
  driver = NULL,
  level = c("layer", "dataset"),
  call = rlang::caller_env()
)

## S3 method for class 'character'
as_gdal_creation_opts(
```

```
  x,
  ...,
  driver = NULL,
  level = c("layer", "dataset"),
  call = rlang::caller_env()
)

## S3 method for class 'tbl_df'
as_gdal_creation_opts(
  x,
  ...,
  driver = NULL,
  level = c("layer", "dataset"),
  call = rlang::caller_env()
)
```

Arguments

x	Object to coerce.
...	Unused; for method extensibility.
driver	Optional GDAL driver short name to attach.
level	Creation-option level: "layer" (--lco) or "dataset" (--co).
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.

Value

A gdal_creation_opts() object.

---

as_gdal_open_opts *Coerce to GDAL Open Options*

Description

Coerce a named list, a KEY=VALUE character vector, a driver-metadata tibble, or an existing gdal_open_opts to the gdal_open_opts() class.

Usage

```
as_gdal_open_opts(x, ..., driver = NULL, call = rlang::caller_env())

## Default S3 method:
as_gdal_open_opts(x, ..., driver = NULL, call = rlang::caller_env())

## S3 method for class 'gdal_open_opts'
```

```
as_gdal_open_opts(x, ..., driver = NULL, call = rlang::caller_env())

## S3 method for class 'list'
as_gdal_open_opts(x, ..., driver = NULL, call = rlang::caller_env())

## S3 method for class 'character'
as_gdal_open_opts(x, ..., driver = NULL, call = rlang::caller_env())

## S3 method for class 'tbl_df'
as_gdal_open_opts(x, ..., driver = NULL, call = rlang::caller_env())
```

Arguments

x	Object to coerce.
...	Unused; for method extensibility.
driver	Optional GDAL driver short name to attach.
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.

Value

A gdal_open_opts() object.

---

as_gdal_vsi_opts *Coerce to GDAL VSI Options*

Description

Coerce a named list, a KEY=VALUE character vector, or an existing gdal_vsi_opts to the gdal_vsi_opts() class. VSI options are path-scoped, config-like settings applied via gdalraster::vsi_set_path_option().

Usage

```
as_gdal_vsi_opts(x, ..., vsi_path = NULL, call = rlang::caller_env())

## Default S3 method:
as_gdal_vsi_opts(x, ..., vsi_path = NULL, call = rlang::caller_env())

## S3 method for class 'gdal_vsi_opts'
as_gdal_vsi_opts(x, ..., vsi_path = NULL, call = rlang::caller_env())

## S3 method for class 'list'
as_gdal_vsi_opts(x, ..., vsi_path = NULL, call = rlang::caller_env())

## S3 method for class 'character'
as_gdal_vsi_opts(x, ..., vsi_path = NULL, call = rlang::caller_env())
```

Arguments

x	Object to coerce.
...	Unused; for method extensibility.
vsi_path	Optional VSI path prefix the options apply to (e.g. "/vsis3/bucket").
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.

Value

A gdal_vsi_opts() object.

---

as_gdalg *Coerce to GDALG*

Description

Coerce to GDALG

Usage

```
as_gdalg(x, ..., call = rlang::caller_env())
```

Arguments

x	An object to coerce to class "gdalg".
...	Additional arguments passed to specific methods.
call	The calling environment, used for error messages.

Value

An object of class "gdalg".

check_available_ram *Check Available RAM*

Description

Checks that the provided value x (in bytes) does not exceed the available system RAM as returned by sys_available_ram(). If the check fails, an error is thrown.

Usage

```
check_available_ram(x, arg = rlang::caller_arg(x), call = rlang::caller_env())
```

Arguments

x	The object to check.
arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.

Value

If the check passes, invisibly returns the provided x value. If the check fails, a condition error is thrown indicating that the provided value exceeds available system RAM.

check_inherits *Class Inheritence Checks*

Description

These functions perform checks that assert the underlying class of objects passed to them.

- check_inherits(): checks that object x is of class class using base::inherits()
- check_inherits2(): checks that object x is of class class using base::.class2()
- check_inherits_any(): checks that object x is at least one of the provided classes via rlang::inherits_any()
- check_inherits_all(): checks that object x is all of the provided classes via rlang::inherits_all()

If validation fails for any of these functions, an error is thrown via check_abort() displaying a friendly error message.

Usage

```
check_inherits(
  x,
  class,
  arg = rlang::caller_arg(x),
  call = rlang::caller_env()
)

check_inherits2(
  x,
  class,
  arg = rlang::caller_arg(x),
  call = rlang::caller_env()
)

check_inherits_any(
  x,
  classes,
  arg = rlang::caller_arg(x),
  call = rlang::caller_env()
)

check_inherits_all(
  x,
  classes,
  arg = rlang::caller_arg(x),
  call = rlang::caller_env()
)
```

Arguments

`x`	The object to check.
`class`, `classes`	The name of the class or classes to use during checking.
`arg`	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
`call`	The execution environment of a currently running function, e.g. `caller_env()`. The function will be mentioned in error messages as the source of the error. See the `call` argument of `abort()` for more information.

Value

If checks pass, invisibly returns the provided `x` object. If checks fail, a condition error is thrown.

check_not_empty *Check Not Empty*

Description

Checks the provided x is not "empty" via rlang::is_empty().

Usage

```
check_not_empty(x, arg = rlang::caller_arg(x), call = rlang::caller_env())
```

Arguments

x	The object to check.
arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.

Value

If checks pass, invisibly returns the provided x object. If checks fail, a condition error is thrown.

check_string *Check String (Scalar)*

Description

Checks the provided x is a scalar string via rlang::check_string().

Usage

```
check_string(x, arg = rlang::caller_arg(x), call = rlang::caller_env())
```

Arguments

x	The object to check.
arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.

Value

If checks pass, invisibly returns the provided x object. If checks fail, a condition error is thrown.

checks *Check Functions*

Description

Collection of various checking functions primarily used for incorporating argument validation checks for package functions.

These check functions act as assertions, and will either return the provided objects invisibly, or throw exceptions.

fgb_config_opts *FlatGeobuf Configuration Options*

Description

The `FlatGeobuf` driver exposes no documented configuration options. This constructor exists for interface symmetry with the other driver option families; it warns and returns an empty `gdal_config_opts()` object.

Usage

```
fgb_config_opts(...)
```

Arguments

...	Ignored (no configuration options are available for this driver).

Value

An (empty) `gdal_config_opts()` object for the `FlatGeobuf` driver.

See Also

`fgb_open_opts()`, `fgb_creation_opts()`

Examples

```
fgb_config_opts()
```

fgb_creation_opts *FlatGeobuf Creation Options*

Description

Construct a layer-level gdal_creation_opts() object for the FlatGeobuf driver. Only options you supply are emitted; values are validated against the driver's registered metadata.

Usage

```
fgb_creation_opts(
  spatial_index = NULL,
  temporary_dir = NULL,
  title = NULL,
  description = NULL,
  ...,
  .set_defaults = FALSE
)
```

Arguments

spatial_index	Value for SPATIAL_INDEX. Logical TRUE/FALSE (coerced to "YES"/"NO") controlling whether a packed Hilbert R-tree spatial index is written. GDAL default "YES".
temporary_dir	Value for TEMPORARY_DIR (path to an existing directory for temporary files; only used when SPATIAL_INDEX = TRUE. "/vsimem/" may be used for in-memory temporaries).
title	Value for TITLE (GDAL >= 3.9); dataset title (should be relatively short).
description	Value for DESCRIPTION (GDAL >= 3.9); dataset description (free-form long text).
...	Additional NAME = value options passed through verbatim alongside the typed arguments. They are coerced and validated against the driver metadata in the same way, and take precedence over a typed argument that sets the same option.
.set_defaults	Logical. If TRUE, options left unset (NULL) are filled with the driver's documented GDAL metadata defaults (via the relevant gdal_vector_driver_*_opts_defaults()); user-supplied values always take precedence. Defaults to FALSE.

Value

A layer-level gdal_creation_opts() object for the FlatGeobuf driver.

See Also

fgb_open_opts(), gdal_creation_opts()

- FlatGeobuf Home Page

- FlatGeobuf GDAL Driver
  - FlatGeobuf GDAL Open Options
  - FlatGeobuf GDAL Layer Creation Options

Examples

```
fgb_creation_opts(spatial_index = TRUE, title = "Parcels")
```

`fgb_open_opts` *FlatGeobuf Open Options*

Description

Construct a `gdal_open_opts()` object for the `FlatGeobuf` driver. Only options you supply are emitted; values are validated against the driver's registered metadata.

Usage

```
fgb_open_opts(verify_buffers = NULL, ..., .set_defaults = FALSE)
```

Arguments

`verify_buffers`	Value for `VERIFY_BUFFERS`. Logical `TRUE`/`FALSE` (coerced to `"YES"`/ `"NO"`) controlling whether flatbuffer integrity is verified on read. `"YES"` (the GDAL default) guards against corrupt data at a small performance cost; `"NO"` is faster but unsafe on malformed files. `NULL` (default) leaves it unset.
`...`	Additional `NAME = value` options passed through verbatim alongside the typed arguments. They are coerced and validated against the driver metadata in the same way, and take precedence over a typed argument that sets the same option.
`.set_defaults`	Logical. If `TRUE`, options left unset (`NULL`) are filled with the driver's documented GDAL metadata defaults (via the relevant `gdal_vector_driver_*_opts_defaults()`); user-supplied values always take precedence. Defaults to `FALSE`.

Value

A `gdal_open_opts()` object for the `FlatGeobuf` driver.

See Also

`fgb_creation_opts()`, `gdal_open_opts()`

- FlatGeobuf Home Page
- FlatGeobuf GDAL Driver
  - FlatGeobuf GDAL Open Options
  - FlatGeobuf GDAL Layer Creation Options

Examples

```
fgb_open_opts()
fgb_open_opts(verify_buffers = FALSE)
fgb_open_opts(.set_defaults = TRUE)
```

`fgb_validate_spatial_index_ram` *Validate Spatial Index RAM Requirement*

Description

Validates if the available RAM is sufficient to build a spatial index for a FlatGeobuf file based on the number of features and an estimated RAM requirement of 83 bytes per feature.

Usage

```
fgb_validate_spatial_index_ram(fgb_dsn, force = FALSE, quiet = FALSE)
```

Arguments

`fgb_dsn`	The data source name (DSN) of the FlatGeobuf file to validate.
`force`	Logical indicating whether to force the feature count (default: `FALSE`).
`quiet`	Logical indicating whether to suppress success messages (default: `FALSE`).

Value

Returns `TRUE` if sufficient RAM is available to build the spatial index, and `FALSE` otherwise. Also provides informative messages about the RAM requirements and availability.

`gdal_config` *GDAL Configuration*

Description

Creates a session-wide, global GDAL Configuration object of class `"gdal_config"`.

Usage

```
gdal_config(...)
```

Arguments

`...`	Configuration options as named `KEY = value` pairs.

Value

gdal_config

gdal_config_links *GDAL Configuration Links*

Description

Links to the official GDAL Configuration Documentation.

- GDAL Configuration Options
  - Setting Configuration Options

See Also

gdal_config_opts()

gdal_config_opts *GDAL Configuration Options*

Description

Construct a gdal_config_opts() object from NAME = value pairs. Configuration options are global, stateful settings applied to the GDAL process (via gdalraster::set_config_option() / the CLI --config flag), and are *not* algorithm arguments. When driver is supplied, values for boolean options are validated against the driver metadata.

Usage

```
gdal_config_opts(..., driver = NULL, .set_defaults = FALSE)
```

Arguments

...	Named configuration options as KEY=VALUE pairs.
driver	Optional GDAL driver short name (e.g. "GPKG") to associate.
.set_defaults	Logical. If TRUE, options left unset (NULL) are filled with the driver's documented GDAL metadata defaults (via the relevant gdal_vector_driver_*_opts_defaults()); user-supplied values always take precedence. Defaults to FALSE.

Value

A gdal_config_opts() object.

Examples

```
gdal_config_opts(CPL_DEBUG = "ON", GDAL_NUM_THREADS = "ALL_CPUS")
```

gdal_creation_opts *GDAL Creation Options*

Description

Construct a `gdal_creation_opts()` object from `NAME = value` pairs. The `level` controls whether these are dataset-creation options (`--co`) or layer-creation options (`--lco`, the default).

Usage

```
gdal_creation_opts(
  ...,
  driver = NULL,
  level = c("layer", "dataset"),
  .set_defaults = FALSE
)
```

Arguments

`...`	Named creation options (`NAME = value`). Logical values are coerced to `"YES"`/`"NO"`.
`driver`	Optional GDAL driver short name to associate.
`level`	Creation-option level, `"layer"` (default) or `"dataset"`.
`.set_defaults`	Logical. If `TRUE`, options left unset (`NULL`) are filled with the driver's documented GDAL metadata defaults (via the relevant `gdal_vector_driver_*_opts_defaults()`); user-supplied values always take precedence. Defaults to `FALSE`.

Value

A `gdal_creation_opts()` object.

Examples

```
gdal_creation_opts(COMPRESSION = "ZSTD", driver = "Parquet")
```

gdal_drivers *GDAL Drivers*

Description

Query the GDAL drivers registered in the active GDAL build.

- `gdal_drivers()`: returns a normalized `tibble::tibble()` of driver metadata (identity, capabilities, supported extensions and SQL dialects), optionally filtered by a name pattern.
- `gdal_driver_names()`: returns just the short driver names.

The driver table is built once from `gdalraster::gdal_formats()` and cached for the session.

Usage

```
gdal_drivers(pattern = NULL, ignore_case = TRUE)

gdal_driver_names(pattern = NULL)
```

Arguments

pattern	Optional character vector of regular-expression patterns. Drivers whose short or long name matches any pattern are returned. NULL (default) returns all drivers.
ignore_case	Logical; match pattern case-insensitively. Defaults to TRUE.

Value

- gdal_drivers(): a tibble::tibble() with one row per driver.
- gdal_driver_names(): a character vector of short driver names.

See Also

gdal_vector_driver_opts(), gdal_vector_driver_capabilities()

Examples

```
gdal_drivers()
gdal_drivers(c("parquet", "geojson"))
gdal_driver_names("gpkg")
```

---

gdal_open_opts *GDAL Open Options*

Description

Construct a gdal_open_opts() object from NAME = value pairs (the GDAL --oo / GDALOpenEx() open-option channel).

Usage

```
gdal_open_opts(..., driver = NULL, .set_defaults = FALSE)
```

Arguments

...	Named open options (NAME = value). Logical values are coerced to "YES"/"NO".
driver	Optional GDAL driver short name to associate.
.set_defaults	Logical. If TRUE, options left unset (NULL) are filled with the driver's documented GDAL metadata defaults (via the relevant gdal_vector_driver_*_opts_defaults()); user-supplied values always take precedence. Defaults to FALSE.

Value

A gdal_open_opts() object.

Examples

```
gdal_open_opts(LIST_ALL_TABLES = FALSE, driver = "GPKG")
```

---

gdal_opts *GDAL Options*

Description

GDAL options control how the GDAL/OGR library reads, writes, and otherwise processes geospatial data. This package models the distinct GDAL option *channels* as a small family of S3 classes, each backed by a named list of `NAME = "VALUE"` pairs (values stored as their coerced GDAL strings) plus a `driver` attribute, so a set of options is an inert, composable value that can be rendered on demand to whichever form a given consumer needs.

In the GDAL CLI, an option is supplied via one of these argument flags:

```
--config <NAME>=<VALUE>
--open-option / --oo <NAME>=<VALUE>
--creation-option / --co <NAME>=<VALUE>
--layer-creation-option / --lco <NAME>=<VALUE>
```

Classes

All four classes share the `gdal_opts` base (a named `list`) and differ only by channel:

```
gdal_opts
 |- gdal_config_opts                          --config      CPLSetConfigOption()
 |- gdal_open_opts                             --oo          GDALOpenEx()
 |- gdal_creation_opts  (level = layer)      --lco       GDALDatasetCreateLayer()
 |                      (level = dataset)      --co          GDALCreate()
 |- gdal_vsi_opts     (vsi_path)         --config     VSISetPathSpecificOption()
```

- gdal_config_opts(): global, stateful configuration options. Applied to the process/session (via gdalraster::set_config_option()), not as an algorithm argument.
- gdal_open_opts(): driver open options.
- gdal_creation_opts(): driver dataset- or layer-creation options, selected by `level`.
- gdal_vsi_opts(): virtual file system (VSI) path-scoped options (config-like).

Rendering

A `gdal_opts` value is rendered with as_gdal_args() (CLI token vector for gdalraster::gdal_alg() / gdalraster::gdal_run()), as_config_option() (a `NAME = VALUE` character vector for gdalraster::set_config_opti config/VSI only), or gdal_render() (a copy-pasteable shell snippet).

See Also

as_gdal_args(), as_config_option(), gdal_render(); and the typed per-driver builders gpkg_open_opts(), gpq_creation_opts(), shp_open_opts(), fgb_creation_opts(), gdb_open_opts().

---

gdal_render *Render GDAL Options as a Shell Command Snippet*

Description

Render a gdal_opts() object to a copy-pasteable, multi-line shell snippet with one flag/value pair per line and the appropriate line-continuation for the target shell. Only the option flags (and the leading --input-format/--output-format when the driver is known) are rendered; the base gdal invocation and datasets are not included.

Usage

```
gdal_render(x, shell = c("bash", "sh", "pwsh", "cmd"))
```

Arguments

x	A gdal_opts() object.
shell	Target shell dialect controlling quoting and continuation: "bash"/"sh" (\\, single quotes), "pwsh" (`, single quotes), or "cmd" (^, double quotes).

Value

A length-1 character string (embedded newlines), or "" when there are no options.

Examples

```
gdal_render(gdal_creation_opts(COMPRESSION = "ZSTD", driver = "Parquet"), shell = "bash")
```

---

gdal_vector_driver_capabilities
GDAL Vector Driver Capabilities

Description

Return a driver's capability flags (the DCAP_* metadata items) as a named logical vector.

Usage

```
gdal_vector_driver_capabilities(driver)
```

Arguments

driver	Character scalar GDAL driver short name.

Value

A named logical vector, one element per DCAP_* capability (name without the DCAP_ prefix retained as given by GDAL), TRUE where the capability is advertised.

See Also

gdal_drivers()

Examples

```
gdal_vector_driver_capabilities("GPKG")
```

---

gdal_vector_driver_config_opts

GDAL Vector Driver Configuration Options

Description

Accessors for a driver's configuration options (the --config channel). Configuration options are sourced from curated package data, since GDAL does not expose them in driver metadata.

- gdal_vector_driver_config_opts(): the configuration-option table for driver.
- gdal_vector_driver_config_opts_defaults(): name to default (all, or one when opt_name given).
- gdal_vector_driver_config_opts_values(): name to allowed values (only constrained options).
- gdal_vector_driver_config_opts_types(): name to data_type.

Usage

```
gdal_vector_driver_config_opts(driver)

gdal_vector_driver_config_opts_defaults(driver, opt_name = NULL)

gdal_vector_driver_config_opts_values(driver, opt_name = NULL)

gdal_vector_driver_config_opts_types(driver)
```

Arguments

driver	Character scalar GDAL driver short name.
opt_name	Optional single option name. When supplied, returns the value for that option only; otherwise returns the full named result.

Value

A tibble::tibble(), named character vector, or named list (see gdal_vector_driver_opts()).

See Also

gdal_config_opts(), gdal_vector_driver_opts()

Examples

```
gdal_vector_driver_config_opts("GPKG")
gdal_vector_driver_config_opts_values("GPKG")
```

---

gdal_vector_driver_creation_opts

GDAL Vector Driver Creation Options

Description

Accessors for a driver's creation options, parsed from both the dataset-level (DMD_CREATIONOPTIONLIST, --co) and layer-level (DS_LAYER_CREATIONOPTIONLIST, --lco) metadata. Use sub_type to restrict to one level.

- gdal_vector_driver_creation_opts(): the creation-option table for driver.
- gdal_vector_driver_creation_opts_defaults(): name to default (all, or one when opt_name given).
- gdal_vector_driver_creation_opts_values(): name to allowed values (only constrained options).
- gdal_vector_driver_creation_opts_types(): name to data_type.

Usage

```
gdal_vector_driver_creation_opts(driver, sub_type = NULL)

gdal_vector_driver_creation_opts_defaults(
  driver,
  opt_name = NULL,
  sub_type = NULL
)

gdal_vector_driver_creation_opts_values(
  driver,
  opt_name = NULL,
  sub_type = NULL
)

gdal_vector_driver_creation_opts_types(driver, sub_type = NULL)
```

Arguments

driver	Character scalar GDAL driver short name.
sub_type	Optional creation level to restrict to: "dataset" or "layer".
opt_name	Optional single option name. When supplied, returns the value for that option only; otherwise returns the full named result.

Value

A tibble::tibble(), named character vector, or named list (see gdal_vector_driver_opts()).

See Also

gdal_creation_opts(), gdal_vector_driver_opts()

Examples

```
gdal_vector_driver_creation_opts("Parquet", sub_type = "layer")
gdal_vector_driver_creation_opts_values("GPKG", sub_type = "layer")
```

---

gdal_vector_driver_open_opts

GDAL Vector Driver Open Options

Description

Accessors for a driver's open options (the --oo / GDALOpenEx() channel), parsed from the driver's DMD_OPENOPTIONLIST metadata.

- gdal_vector_driver_open_opts(): the open-option table for driver.
- gdal_vector_driver_open_opts_defaults(): name to default (all, or one when opt_name given).
- gdal_vector_driver_open_opts_values(): name to allowed values (only constrained options).
- gdal_vector_driver_open_opts_types(): name to data_type.

Usage

```
gdal_vector_driver_open_opts(driver)

gdal_vector_driver_open_opts_defaults(driver, opt_name = NULL)

gdal_vector_driver_open_opts_values(driver, opt_name = NULL)

gdal_vector_driver_open_opts_types(driver)
```

Arguments

driver	Character scalar GDAL driver short name.
opt_name	Optional single option name. When supplied, returns the value for that option only; otherwise returns the full named result.

Value

A tibble::tibble(), named character vector, or named list (see gdal_vector_driver_opts()).

See Also

gdal_open_opts(), gdal_vector_driver_opts()

Examples

```
gdal_vector_driver_open_opts("GPKG")
gdal_vector_driver_open_opts_values("GPKG", "LIST_ALL_TABLES")
```

---

gdal_vector_driver_opts

GDAL Vector Driver Options

Description

Look up the documented options for the core supported vector drivers (see GDAL_VECTOR_DRIVERS), drawn from the merged option table assembled at package load (driver-metadata XML for open/creation options plus the curated configuration options).

- gdal_vector_driver_opts(): the full option table, optionally filtered by type, sub_type, and scope.
- gdal_vector_driver_opt_defaults(): a named vector mapping option name to its declared default.
- gdal_vector_driver_opt_values(): a named list mapping option name to its allowed values (booleans expanded to c("YES", "NO")); only options that declare a constrained set appear.
- gdal_vector_driver_opt_types(): a named vector mapping option name to its data_type.

Usage

```
gdal_vector_driver_opts(
  driver = NULL,
  type = NULL,
  sub_type = NULL,
  scope = NULL
)
```

```
gdal_vector_driver_opt_defaults(
  driver,
  type = NULL,
  sub_type = NULL,
  scope = NULL
)

gdal_vector_driver_opt_values(
  driver,
  type = NULL,
  sub_type = NULL,
  scope = NULL
)

gdal_vector_driver_opt_types(
  driver,
  type = NULL,
  sub_type = NULL,
  scope = NULL
)
```

Arguments

driver	Character scalar GDAL driver short name (e.g. "GPKG"). When NULL (only for gdal_vector_driver_opts()), the options for all core vector drivers are returned.
type	Optional option channel to filter to: one of "config", "open", or "creation".
sub_type	Optional creation sub-type to filter to: "dataset" or "layer".
scope	Optional data-type scope to filter to (e.g. "vector", "all").

Value

- gdal_vector_driver_opts(): a tibble::tibble() of options.
- gdal_vector_driver_opt_defaults() / gdal_vector_driver_opt_types(): a named character vector.
- gdal_vector_driver_opt_values(): a named list of character vectors.

See Also

gdal_vector_driver_open_opts(), gdal_vector_driver_creation_opts(), gdal_vector_driver_config_opts()

Examples

```
gdal_vector_driver_opts("GPKG", type = "open")
gdal_vector_driver_opt_defaults("GPKG", type = "open")
```

gdal_vsi_opts *GDAL VSI Options*

Description

Construct a gdal_vsi_opts() object from NAME = value pairs, optionally scoped to a vsi_path. These are config-like options for GDAL virtual file systems (e.g. cloud storage credentials and HTTP tuning) and render as --config NAME=VALUE.

Usage

```
gdal_vsi_opts(..., vsi_path = NULL)
```

Arguments

...	Named VSI options (NAME = value).
vsi_path	Optional VSI path prefix (e.g. "/vsis3/bucket").

Value

A gdal_vsi_opts() object.

Examples

```
gdal_vsi_opts(AWS_REGION = "us-east-1", vsi_path = "/vsis3/my-bucket")
```

gdb_config_opts *OpenFileGDB Configuration Options*

Description

Construct a gdal_config_opts() object for the OpenFileGDB driver. Only options you supply are emitted; boolean values are validated against the driver metadata.

Usage

```
gdb_config_opts(
  default_string_width = NULL,
  in_memory_spi = NULL,
  ...,
  .set_defaults = FALSE
)
```

Arguments

default_string_width	Value for OPENFILEGDB_DEFAULT_STRING_WIDTH (integer). Width for string fields created when the requested width is the unspecified value 0. GDAL default 65536.
in_memory_spi	Value for OPENFILEGDB_IN_MEMORY_SPI. Logical TRUE/FALSE (coerced to "YES"/"NO"); build an in-memory spatial index instead of using the native one.
...	Additional NAME = value options passed through verbatim alongside the typed arguments. They are coerced and validated against the driver metadata in the same way, and take precedence over a typed argument that sets the same option.
.set_defaults	Logical. If TRUE, options left unset (NULL) are filled with the driver's documented GDAL metadata defaults (via the relevant gdal_vector_driver_*_opts_defaults()); user-supplied values always take precedence. Defaults to FALSE.

Value

A gdal_config_opts() object for the OpenFileGDB driver.

See Also

gdb_open_opts(), gdb_creation_opts(), gdal_config_opts()

- OpenFileGDB GDAL Driver
  - OpenFileGDB GDAL Open Options
  - OpenFileGDB GDAL Layer Creation Options
  - OpenFileGDB GDAL Configuration Options
- ESRI File Geodatabase (.gdb) Format

Examples

```
gdb_config_opts(default_string_width = 1024L, in_memory_spi = TRUE)
```

---

gdb_creation_opts *OpenFileGDB Creation Options*

Description

Construct a layer-level gdal_creation_opts() object for the OpenFileGDB driver. Typed arguments cover the common layer options; advanced coordinate-precision grid options (XORIGIN, XYSCALE, XYTOLERANCE, Z*/M*) may be supplied through ....

Usage

```
gdb_creation_opts(
  fid = NULL,
  geometry_name = NULL,
  geometry_nullable = NULL,
  configuration_keyword = NULL,
  target_arcgis_version = NULL,
  create_multipatch = NULL,
  create_shape_area_and_length_fields = NULL,
  time_in_utc = NULL,
  column_types = NULL,
  feature_dataset = NULL,
  layer_alias = NULL,
  documentation = NULL,
  ...,
  .set_defaults = FALSE
)
```

Arguments

`fid`	Value for `FID` (name of the OID column). GDAL default `"OBJECTID"`.
`geometry_name`	Value for `GEOMETRY_NAME`. GDAL default `"SHAPE"`.
`geometry_nullable`	Value for `GEOMETRY_NULLABLE` (logical -> `"YES"`/`"NO"`). GDAL default `"YES"`.
`configuration_keyword`	Value for `CONFIGURATION_KEYWORD`. One of `DEFAULTS`/`MAX_FILE_SIZE_4GB`/`MAX_FILE_SIZE_256TB`. GDAL default `"DEFAULTS"` (UTF-8 text, up to 1 TB).
`target_arcgis_version`	Value for `TARGET_ARCGIS_VERSION` (GDAL >= 3.9). One of `ALL`/`ARCGIS_PRO_3_2_OR_LATER` (the latter required to create `Integer64`/`Date`/`Time` fields). GDAL default `"ALL"`.
`create_multipatch`	Value for `CREATE_MULTIPATCH` (logical -> `"YES"`/`"NO"`); write MultiPolygon layers as MultiPatch.
`create_shape_area_and_length_fields`	Value for `CREATE_SHAPE_AREA_AND_LENGTH_FIELDS` (logical -> `"YES"`/`"NO"`); auto-populated `Shape_Area`/`Shape_Length` fields. GDAL default `"NO"`.
`time_in_utc`	Value for `TIME_IN_UTC` (logical -> `"YES"`/`"NO"`).
`column_types`	Value for `COLUMN_TYPES` (`"field_name=fgdb_field_type,..."`) forcing FileGDB field types.
`feature_dataset`	Value for `FEATURE_DATASET` (FeatureDataset folder for the new layer; created if it does not exist).
`layer_alias`	Value for `LAYER_ALIAS` (layer-name alias).
`documentation`	Value for `DOCUMENTATION` (XML documentation string).

`...` Additional `NAME = value` options passed through verbatim alongside the typed arguments. They are coerced and validated against the driver metadata in the same way, and take precedence over a typed argument that sets the same option.

`.set_defaults` Logical. If `TRUE`, options left unset (`NULL`) are filled with the driver's documented GDAL metadata defaults (via the relevant `gdal_vector_driver_*_opts_defaults()`); user-supplied values always take precedence. Defaults to `FALSE`.

Value

A layer-level `gdal_creation_opts()` object for the `OpenFileGDB` driver.

See Also

`gdb_open_opts()`, `gdal_creation_opts()`

- OpenFileGDB GDAL Driver
  - OpenFileGDB GDAL Open Options
  - OpenFileGDB GDAL Layer Creation Options
  - OpenFileGDB GDAL Configuration Options
- ESRI File Geodatabase (.gdb) Format

Examples

```
gdb_creation_opts(geometry_name = "SHAPE", target_arcgis_version = "ALL")
```

---

gdb_open_opts *OpenFileGDB Open Options*

Description

Construct a `gdal_open_opts()` object for the `OpenFileGDB` driver.

Usage

```
gdb_open_opts(list_all_tables = NULL, ..., .set_defaults = FALSE)
```

Arguments

`list_all_tables` Value for `LIST_ALL_TABLES` (`"YES"`/`"NO"`; logical coerced). Whether to list all tables, including system/internal `GDB_*` tables. GDAL default `"NO"`.

`...` Additional `NAME = value` options passed through verbatim alongside the typed arguments. They are coerced and validated against the driver metadata in the same way, and take precedence over a typed argument that sets the same option.

`.set_defaults` Logical. If `TRUE`, options left unset (`NULL`) are filled with the driver's documented GDAL metadata defaults (via the relevant `gdal_vector_driver_*_opts_defaults()`); user-supplied values always take precedence. Defaults to `FALSE`.

Value

A gdal_open_opts() object for the OpenFileGDB driver.

See Also

gdb_creation_opts(), gdal_open_opts()

- OpenFileGDB GDAL Driver
  - OpenFileGDB GDAL Open Options
  - OpenFileGDB GDAL Layer Creation Options
  - OpenFileGDB GDAL Configuration Options
- ESRI File Geodatabase (.gdb) Format

Examples

```
gdb_open_opts(list_all_tables = TRUE)
```

---

gpkg_config_opts *GeoPackage Configuration Options*

Description

Construct a gdal_config_opts() object for the GPKG driver. These are global configuration options applied to the GDAL process.

Usage

```
gpkg_config_opts(
  sqlite_cache = NULL,
  sqlite_journal = NULL,
  sqlite_synchronous = NULL,
  sqlite_pragma = NULL,
  use_ogr_vfs = NULL,
  num_threads = NULL,
  ...,
  .set_defaults = FALSE
)
```

Arguments

sqlite_cache	Value for OGR_SQLITE_CACHE (SQLite page cache, in MB).
sqlite_journal	Value for OGR_SQLITE_JOURNAL (journal mode).
sqlite_synchronous	Value for OGR_SQLITE_SYNCHRONOUS (e.g. "OFF").
sqlite_pragma	Value for OGR_SQLITE_PRAGMA (e.g. "pragma_name=value,...").

use_ogr_vfs	Value for SQLITE_USE_OGR_VFS (logical -> "YES"/"NO").
num_threads	Value for OGR_GPKG_NUM_THREADS (GDAL >= 3.8.3); an integer or "ALL_CPUS". Threads used when reading tables through the ArrowArray interface. GDAL default is min(4, nCPU).
...	Additional NAME = value options passed through verbatim alongside the typed arguments. They are coerced and validated against the driver metadata in the same way, and take precedence over a typed argument that sets the same option.
.set_defaults	Logical. If TRUE, options left unset (NULL) are filled with the driver's documented GDAL metadata defaults (via the relevant gdal_vector_driver_*_opts_defaults()); user-supplied values always take precedence. Defaults to FALSE.

Value

A gdal_config_opts() object for the GPKG driver.

See Also

gpkg_open_opts(), gpkg_creation_opts(), gdal_config_opts()

- GDAL GPKG (GeoPackage) vector driver
- GPKG open options
- GDAL configuration options
- SQLite PRAGMA statements
- GeoPackage specification

Examples

```
gpkg_config_opts(sqlite_synchronous = "OFF", use_ogr_vfs = TRUE, num_threads = "ALL_CPUS")
```

---

gpkg_creation_opts *GeoPackage Creation Options*

Description

Construct a gdal_creation_opts() object for the GPKG driver. The typed arguments are layer-creation options (level = "layer", the default, --lco); dataset-creation options (VERSION, METADATA_TABLES, ADD_GPKG_OGR_CONTENTS, ...) are supplied through ... together with level = "dataset".

Usage

```
gpkg_creation_opts(
  fid = NULL,
  geometry_name = NULL,
  geometry_nullable = NULL,
  spatial_index = NULL,
  identifier = NULL,
```

```
  description = NULL,
  launder = NULL,
  overwrite = NULL,
  ...,
  level = c("layer", "dataset"),
  .set_defaults = FALSE
)
```

Arguments

fid	Name of the FID column (FID). GDAL default "fid".
geometry_name	Name of the geometry column (GEOMETRY_NAME). GDAL default "geom".
geometry_nullable	Value for GEOMETRY_NULLABLE (logical -> "YES"/"NO").
spatial_index	Value for SPATIAL_INDEX (logical -> "YES"/"NO"). GDAL default "YES".
identifier	Value for IDENTIFIER (contents-table identifier).
description	Value for DESCRIPTION (contents-table description).
launder	Value for LAUNDER (logical -> "YES"/"NO"). GDAL default "NO".
overwrite	Value for OVERWRITE (logical -> "YES"/"NO"). GDAL default "NO".
...	Additional NAME = value options passed through verbatim alongside the typed arguments. They are coerced and validated against the driver metadata in the same way, and take precedence over a typed argument that sets the same option.
level	Creation-option level, "layer" (default, --lco) or "dataset" (--co). Dataset-level options (e.g. VERSION, METADATA_TABLES, ADD_GPKG_OGR_CONTENTS) are supplied through ... with level = "dataset".
.set_defaults	Logical. If TRUE, options left unset (NULL) are filled with the driver's documented GDAL metadata defaults (via the relevant gdal_vector_driver_*_opts_defaults()); user-supplied values always take precedence. Defaults to FALSE.

Value

A gdal_creation_opts() object for the GPKG driver.

See Also

gpkg_open_opts(), gdal_creation_opts()

- GDAL GPKG (GeoPackage) vector driver
- GPKG open options
- GDAL configuration options
- SQLite PRAGMA statements
- GeoPackage specification

Examples

```
gpkg_creation_opts(geometry_name = "geom", spatial_index = TRUE)
gpkg_creation_opts(VERSION = "1.4", level = "dataset")
```

gpkg_open_opts *GeoPackage GDAL Open Options*

Description

Construct a gdal_open_opts() object for the GPKG (GeoPackage) driver.

Usage

```
gpkg_open_opts(
  list_all_tables = NULL,
  prelude_statements = NULL,
  nolock = NULL,
  immutable = NULL,
  ...,
  .set_defaults = FALSE
)
```

Arguments

list_all_tables	Value for LIST_ALL_TABLES ("AUTO"/"YES"/"NO"; logical coerced). Whether to list tables not registered in gpkg_contents. GDAL default "AUTO".
prelude_statements	SQL/PRAGMA statements for PRELUDE_STATEMENTS (a single string; see gpkg_prelude_pragmas()).
nolock	Value for NOLOCK (logical -> "YES"/"NO"); open in nolock mode (skip SQLite locking; only safe for read-only access to media nothing else can write).
immutable	Value for IMMUTABLE (logical -> "YES"/"NO"); declare the database immutable. Only when the file genuinely cannot change.
...	Additional NAME = value options passed through verbatim alongside the typed arguments. They are coerced and validated against the driver metadata in the same way, and take precedence over a typed argument that sets the same option.
.set_defaults	Logical. If TRUE, options left unset (NULL) are filled with the driver's documented GDAL metadata defaults (via the relevant gdal_vector_driver_*_opts_defaults()); user-supplied values always take precedence. Defaults to FALSE.

Details

Because a GeoPackage is a SQLite database, several open options carry performance and safety implications. Of note is the PRELUDE_STATEMENTS open option, which allows you to specify arbitrary SQL statements that will run before any other queries once connected to the SQLite3 connection is established. This is commonly used to attach another database and issue cross-database requests, but we use it more commonly here to set PRAGMA statements to optimize performance and avoid the global configuration OGR_SQLITE_* options.

Each open option is enumerated and described below:

- LIST_ALL_TABLES=[AUTO/YES/NO]: Defaults to AUTO. Whether all tables, including those not listed in gpkg_contents, should be listed. If AUTO, all tables including those not listed in gpkg_contents will be listed, except if the aspatial extension is found or a table is registered as 'attributes' in gpkg_contents. If YES, all tables including those not listed in gpkg_contents will be listed, in all cases. If NO, only tables registered as 'features', 'attributes' or 'aspatial' will be listed.
- PRELUDE_STATEMENTS=[SQL]: (GDAL >= 3.2) SQL statement(s) to send on the SQLite3 connection before any other ones. In case of several statements, they must be separated with the semicolon (;) sign. This option may be useful to attach another database to the current one and issue cross-database requests.
- NOLOCK=[YES/NO]: (GDAL >= 3.4.2) Defaults to NO. Whether the database should be used without doing any file locking. Setting it to YES will only be honored when opening in read-only mode and if the journal mode is not WAL. This corresponds to the nolock=1 query parameter described at https://www.sqlite.org/uri.html.
- IMMUTABLE=[YES/NO]: (GDAL >= 3.5.3) Whether the database should be opened by assuming that the file cannot be modified by another process. This will skip any checks for change detection. This can be useful for WAL enabled files on read-only storage. GDAL will automatically try to turn it on when not being able to open in read-only mode a WAL enabled file. This corresponds to the immutable=1 query parameter described at https://www.sqlite.org/uri.html.

Value

A gdal_open_opts() object for the GPKG driver.

See Also

gpkg_prelude_pragmas(), gpkg_creation_opts(), gdal_open_opts()

- GDAL GPKG (GeoPackage) vector driver
- GPKG open options
- GDAL configuration options
- SQLite PRAGMA statements
- GeoPackage specification

Examples

```
gpkg_open_opts(list_all_tables = FALSE, nolock = TRUE)

prelude <- gpkg_prelude_pragmas(cache_size = -4000000, temp_store = "MEMORY")
gpkg_open_opts(list_all_tables = FALSE, prelude_statements = prelude)
```

gpkg_prelude_pragmas *GeoPackage Prelude* PRAGMA *Statements*

Description

Build a PRELUDE_STATEMENTS string of SQLite PRAGMA directives for use as a GPKG/SQLite open option. The result is a single semicolon-separated string. Because it embeds ; (and possibly ,), it is carried as single --open-option value and rendered as such.

Usage

```
gpkg_prelude_pragmas(
  cache_size = NULL,
  temp_store = NULL,
  mmap_size = NULL,
  journal_mode = NULL,
  ...
)
```

Arguments

cache_size	Integer page cache size. Negative values are in kibibytes (e.g. -4000000 is roughly 4 GB).
temp_store	Where temporary tables live: "DEFAULT", "FILE", or "MEMORY" (also accepts integer 0L/1L/2L).
mmap_size	Maximum bytes for memory-mapped I/O.
journal_mode	SQLite journal mode: "DELETE", "WAL", "TRUNCATE", "PERSIST", "MEMORY", or "OFF".
...	Additional raw PRAGMA ...; statement strings appended verbatim.

Value

A length-1 character string of semicolon-separated PRAGMA statements (or "").

See Also

gpkg_open_opts()

Examples

```
gpkg_prelude_pragmas(cache_size = -4000000, temp_store = "MEMORY", journal_mode = "WAL")
```

gpq_creation_opts *GeoParquet Creation Options*

Description

Construct a layer-level gdal_creation_opts() object for the Parquet (GeoParquet) driver. Only options you supply are emitted; enumerated values are validated against the driver metadata.

Usage

```
gpq_creation_opts(
  compression = NULL,
  compression_level = NULL,
  geometry_encoding = NULL,
  row_group_size = NULL,
  geometry_name = NULL,
  fid = NULL,
  polygon_orientation = NULL,
  edges = NULL,
  creator = NULL,
  write_covering_bbox = NULL,
  covering_bbox_name = NULL,
  use_parquet_geo_types = NULL,
  sort_by_bbox = NULL,
  timestamp_with_offset = NULL,
  coordinate_precision = NULL,
  ...,
  .set_defaults = FALSE
)
```

Arguments

compression	Value for COMPRESSION. One of NONE/SNAPPY/GZIP/BROTLI/ZSTD/ LZ4_RAW/LZ4_HADOOP (available values depend on how the Parquet library was built). GDAL default "SNAPPY" when available, otherwise NONE.
compression_level	Value for COMPRESSION_LEVEL (GDAL >= 3.12). Codec-dependent integer.
geometry_encoding	Value for GEOMETRY_ENCODING. One of WKB/WKT/GEOARROW/ GEOARROW_INTERLEAVED. GDAL default "WKB" (recommended for interoperability).
row_group_size	Value for ROW_GROUP_SIZE (maximum rows per group). GDAL default 65536.
geometry_name	Value for GEOMETRY_NAME. GDAL default "geometry".
fid	Value for FID (name of the FID column to create; if unset, no FID column is created).

polygon_orientation
: Value for POLYGON_ORIENTATION. One of COUNTERCLOCKWISE/ UNMODIFIED. GDAL default "COUNTERCLOCKWISE".

edges
: Value for EDGES. One of PLANAR/SPHERICAL. GDAL default "PLANAR".

creator
: Value for CREATOR (name of the creating application).

write_covering_bbox
: Value for WRITE_COVERING_BBOX (GDAL >= 3.9). One of AUTO/YES/NO (logical coerced); write per-row xmin/ymin/xmax/ymax bounding-box columns for faster spatial filtering. GDAL default "AUTO".

covering_bbox_name
: Value for COVERING_BBOX_NAME (GDAL >= 3.13). Defaults to the geometry column name suffixed with _bbox.

use_parquet_geo_types
: Value for USE_PARQUET_GEO_TYPES (GDAL >= 3.12; requires libarrow >= 21). One of YES/NO/ONLY. GDAL default "NO".

sort_by_bbox
: Value for SORT_BY_BBOX (GDAL >= 3.9; logical -> "YES"/"NO"). Spatially order features (via a temporary GeoPackage) for faster spatial filtering. GDAL default "NO".

timestamp_with_offset
: Value for TIMESTAMP_WITH_OFFSET (GDAL >= 3.13). One of AUTO/YES/NO. GDAL default "AUTO".

coordinate_precision
: Value for COORDINATE_PRECISION (number of decimals for coordinates; only for GEOMETRY_ENCODING=WKT).

...
: Additional NAME = value options passed through verbatim alongside the typed arguments. They are coerced and validated against the driver metadata in the same way, and take precedence over a typed argument that sets the same option.

.set_defaults
: Logical. If TRUE, options left unset (NULL) are filled with the driver's documented GDAL metadata defaults (via the relevant gdal_vector_driver_*_opts_defaults()); user-supplied values always take precedence. Defaults to FALSE.

Value

A layer-level gdal_creation_opts() object for the Parquet driver.

Distributing GeoParquet

Following the OGC *Best Practices for Distributing GeoParquet*, good defaults for distribution are ZSTD compression at a moderate level, a row-group size around 50000-150000, the per-row bounding-box covering columns enabled, and spatially ordered features:

```
gpq_creation_opts(
  compression = "ZSTD",
  compression_level = 15,
  row_group_size = 100000,
  write_covering_bbox = TRUE,
  sort_by_bbox = TRUE
)
```

See Also

gpq_open_opts(), gdal_creation_opts()

- GeoParquet Home Page
- (Geo)Parquet GDAL Driver
  - (Geo)Parquet GDAL Layer Creation Options
  - (Geo)Parquet GDAL Open Options
- Best Practices for Distributing GeoParquet

Examples

```
gpq_creation_opts(compression = "ZSTD", geometry_encoding = "WKB")
```

---

gpq_file_info *GeoParquet File Info*

Description

Summarize file-level information about a (Geo)Parquet file from its footer.

Usage

```
gpq_file_info(gpq_path)
```

Arguments

gpq_path	Path to a (Geo)Parquet (*.parquet) file.

Value

A gpq_file_info object (a list) with:

- source, path: the file name and full path.
- file_size: the file size in bytes.
- parquet_version, geoparquet_version: the format versions (the latter is NA for plain Parquet).
- num_rows, num_cols, num_row_groups: the row, column, and row group counts.
- row_group_size, row_group_size_last: the number of rows in the first and last row groups.
- created_by: the writing software.
- kv_keys: the key-value metadata keys present in the file.

Examples

```
f <- system.file("extdata/atlanta/atlanta.parquet", package = "gdalvector")
gpq_file_info(f)
```

gpq_geo_metadata *GeoParquet and GDAL Metadata*

Description

Read and parse the GeoParquet (geo) and GDAL (gdal:creation-options, gdal:schema) key-value metadata, plus any pandas metadata. Geometry column metadata is accessed via the declared primary_column, and the crs field is normalized to a type/name/authority summary across its three forms: absent (NULL), a PROJJSON object, or a WKT string.

Usage

```
gpq_geo_metadata(gpq_path)
```

Arguments

gpq_path	Path to a (Geo)Parquet (*.parquet) file.

Value

A gpq_geo_metadata object (a list). When no geo metadata is present, is_geoparquet is FALSE and the geospatial fields are omitted.

Examples

```
f <- system.file("extdata/atlanta/atlanta.parquet", package = "gdalvector")
gpq_geo_metadata(f)
```

gpq_inspect *Comprehensive GeoParquet Introspection*

Description

Assemble gpq_file_info(), gpq_schema_info(), gpq_row_groups() and gpq_geo_metadata() into a single object.

Usage

```
gpq_inspect(gpq_path)
```

Arguments

gpq_path	Path to a (Geo)Parquet (*.parquet) file.

Value

A gpq_inspect object (a list) with file_info, schema, row_groups and geo_metadata elements.

Examples

```
f <- system.file("extdata/atlanta/atlanta.parquet", package = "gdalvector")
gpq_inspect(f)
```

`gpq_meta` *GeoParquet Metadata*

Description

A family of functions for reading (Geo)Parquet metadata directly from the file footer; no data pages are read. Each returns a classed list with `format()` and `print()` methods:

- `gpq_file_info()`: file-level summary (sizes, versions, row groups).
- `gpq_schema_info()`: the Parquet schema (leaf columns, struct nodes, logical types).
- `gpq_row_groups()`: row group and column chunk statistics, with decoded min/max values.
- `gpq_geo_metadata()`: the GeoParquet (`geo`) and GDAL (`gdal:*`) key-value metadata.
- `gpq_inspect()`: all of the above, assembled into one object.

The footer is read with `nanoparquet::read_parquet_metadata()` and embedded JSON metadata is parsed with `yyjsonr::read_json_str()`.

See Also

- GeoParquet Home Page
- (Geo)Parquet GDAL Driver
  - (Geo)Parquet GDAL Layer Creation Options
  - (Geo)Parquet GDAL Open Options
- Best Practices for Distributing GeoParquet

`gpq_open_opts` *GeoParquet Open Options*

Description

Construct a `gdal_open_opts()` object for the `Parquet` (GeoParquet) driver.

Usage

```
gpq_open_opts(
  geom_possible_names = NULL,
  crs = NULL,
  lists_as_string_json = NULL,
  ...,
  .set_defaults = FALSE
)
```

Arguments

geom_possible_names	Value for GEOM_POSSIBLE_NAMES (GDAL >= 3.8). Comma-separated list of candidate geometry column names, used only for files without GeoParquet metadata. GDAL default "geometry,wkb_geometry,wkt_geometry".
crs	Value for CRS (GDAL >= 3.8). Set or override the CRS of geometry columns, typically "AUTH:CODE" (e.g. "EPSG:4326"), or a PROJ/WKT CRS string.
lists_as_string_json	Value for LISTS_AS_STRING_JSON (GDAL >= 3.12.1; logical -> "YES"/"NO"). Report lists of strings/integers/reals as String(JSON) fields. GDAL default "NO".
...	Additional NAME = value options passed through verbatim alongside the typed arguments. They are coerced and validated against the driver metadata in the same way, and take precedence over a typed argument that sets the same option.
.set_defaults	Logical. If TRUE, options left unset (NULL) are filled with the driver's documented GDAL metadata defaults (via the relevant gdal_vector_driver_*_opts_defaults()); user-supplied values always take precedence. Defaults to FALSE.

Value

A gdal_open_opts() object for the Parquet driver.

See Also

gpq_creation_opts(), gdal_open_opts()

- GeoParquet Home Page
- (Geo)Parquet GDAL Driver
  - (Geo)Parquet GDAL Layer Creation Options
  - (Geo)Parquet GDAL Open Options
- Best Practices for Distributing GeoParquet

Examples

```
gpq_open_opts(crs = "EPSG:4326")
```

---

gpq_row_groups *GeoParquet Row Groups and Statistics*

Description

Read row group and column chunk statistics. Parquet stores chunk min/max statistics as raw bytes in the column's physical type; these are decoded to R scalars and aggregated into a per-column summary across all row groups.

Usage

```
gpq_row_groups(gpq_path)
```

Arguments

gpq_path	Path to a (Geo)Parquet (*.parquet) file.

Value

A gpq_row_groups object (a list) with:

- source, path: the file name and full path.
- row_groups: a per-row-group tibble::tibble().
- column_chunks: the raw per-chunk tibble::tibble() with leaf column names and decoded min/max appended.
- col_summary: a one-row-per-column tibble::tibble() aggregated across all row groups.

Examples

```
f <- system.file("extdata/atlanta/atlanta.parquet", package = "gdalvector")
gpq_row_groups(f)
```

---

gpq_schema *GeoParquet Schema*

Description

Read the tidied Parquet schema as a tibble: one row per leaf column with its Parquet physical type, converted/logical types, repetition, mapped R type, and (for geometry columns) the encoding and geometry types from the geo metadata. For a high-level summary instead of the full table, see gpq_schema_info().

Usage

```
gpq_schema(gpq_path)
```

Arguments

gpq_path	Path to a (Geo)Parquet (*.parquet) file.

Value

A tibble::tibble() with columns id, name, parquet_type, converted_type, logical_type, repetition, r_type, and info.

Examples

```
f <- system.file("extdata/atlanta/atlanta.parquet", package = "gdalvector")
gpq_schema(f)
```

gpq_schema_info *GeoParquet Schema Info*

Description

Summarize the Parquet schema of a (Geo)Parquet file: leaf column and struct node counts, the geometry column(s), and the distribution of Parquet physical types. The full per-column table is available from `gpq_schema()` or the object's `columns` element.

Usage

```
gpq_schema_info(gpq_path)
```

Arguments

`gpq_path`	Path to a (Geo)Parquet (`*.parquet`) file.

Value

A `gpq_schema_info` object (a list) with `source`, `path`, `struct_nodes`, `columns` (the `gpq_schema()` tibble), and the `n_leaf`/`n_struct` counts.

Examples

```
f <- system.file("extdata/atlanta/atlanta.parquet", package = "gdalvector")
gpq_schema_info(f)
```

is_vsi_path *Is VSI Path*

Description

Check if a Path is a GDAL Virtual File System (VSI) Path or URL.

Usage

```
is_vsi_path(x)
```

Arguments

`x`	Character string to check.

Value

Logical indicating if the path starts with a valid VSI prefix (i.e. `/vsicurl/` or `/vsizip/`).

Examples

```
is_vsi_path("/vsizip/data.zip")      # TRUE
is_vsi_path("/vsicurl/data.geojson") # TRUE
is_vsi_path("data.geojson")          # FALSE
```

`local_hash` *Get the Hash of a Local File*

Description

Retrieves the hash of a local file from its path.

Usage

```
local_hash(path, algo = "md5")
```

Arguments

`path`	Character string specifying the path to the local file.
`algo`	Character string specifying hash algorithm ("md5", "sha1", "sha256", "sha512").

Value

A character string representing the hash of the local file.

Examples

```
## Not run:
path <- "data-raw/cache/tiger/GENZ2024/shp/cb_2024_us_state_20m.zip"
local_hash(path)

## End(Not run)
```

`ping` *Ping*

Description

Pings a remote URL.

Usage

```
ping(url, timeout = 5L)
```

Arguments

`url`	The remote URL
`timeout`	Timeout in seconds. Defaults to `5L`.

Value

`TRUE` if successfully pinged, `FALSE` otherwise.

`read_renviron` *Read* `.Renviron` *File*

Description

Simple wrapper around `base::readRenviron()` to read a `.Renviron` file from a specified path. If no path is provided, it defaults to the current working directory.

Usage

```
read_renviron(path = NULL)
```

Arguments

`path`	The directory path where the `.Renviron` file is located. If `NULL`, defaults to the current working directory.

Value

Invisibly returns `NULL`. The function is called for its side effect of loading environment variables into the R session.

`remote_download` *Remote File Download with Change Detection*

Description

Downloads a remote file only if it has changed since the cached version. Uses HTTP Last-Modified header when available (fast), falls back to hash comparison for legacy servers.

Usage

```
remote_download(
  url,
  destfile,
  extract = FALSE,
  timeout = 600L,
  max_tries = 3L,
  force = FALSE,
  algo = "md5"
)
```

Arguments

url	Character string specifying the URL of the remote file.
destfile	Character string specifying the destination path.
extract	Logical; if TRUE and the file is a ZIP, extracts it after download.
timeout	Numeric value specifying HTTP request timeout in seconds. Defaults to 600L.
max_tries	Integer; maximum number of download attempts on failure. Defaults to 3L.
force	Logical; if TRUE, always download regardless of cache state.
algo	Character string specifying hash algorithm ("md5", "sha1", "sha256", "sha512"). Only used as fallback if Last-Modified header unavailable.

Value

Invisibly returns the path to the downloaded file.

---

remote_exists *Check if a Remote File Exists at a Given URL*

Description

Determines whether a remote file exists by sending a HEAD request to the specified URL and checking the HTTP status code.

Usage

```
remote_exists(url)
```

Arguments

url	Character string specifying the URL of the remote file.

Value

A logical value: TRUE if the remote file exists (HTTP status 200), FALSE otherwise.

Examples

```
## Not run:
url <- "https://www2.census.gov/geo/tiger/GENZ2024/shp/cb_2024_us_state_20m.zip"
remote_exists(url)

## End(Not run)
```

`remote_hash` *Get the Hash of a Remote File from its URL*

Description

Retrieves the hash of a remote file from its URL.

Usage

```
remote_hash(url, algo = "md5")
```

Arguments

`url`	Character string specifying the URL of the remote file.
`algo`	Character string specifying hash algorithm ("md5", "sha1", "sha256", "sha512").

Value

A character string representing the hash of the remote file.

Examples

```
## Not run:
url <- "https://www2.census.gov/geo/tiger/GENZ2024/shp/cb_2024_us_state_20m.zip"
remote_hash(url)

## End(Not run)
```

`remote_head` *Perform a* `HEAD` *HTTP Request for a Remote URL*

Description

Sends a `HEAD` request to the specified URL and retrieves the response headers.

This function is useful for checking the existence of a resource or retrieving metadata without downloading the entire content.

Usage

```
remote_head(url)
```

Arguments

url	Character string specifying the URL to send the HEAD request to.

Value

A list containing:

- request: The httr2::request()
- response: The httr2::response()
- headers: The httr2::resp_headers() from the response

Examples

```
## Not run:
url <- "https://www2.census.gov/geo/tiger/GENZ2024/shp/cb_2024_us_state_20m.zip"
head_info <- remote_head(url)
print(head_info$headers)

## End(Not run)
```

---

remote_last_modified *Get Local & Remote Resources Last-Modified Timestamp*

Description

These functions derive timestamps for local and remote resources.

- remote_last_modified(): Parses the Last-Modified HTTP response header as the timestamp. Returns NA if the header is not available.
- local_last_modified(): Local file last modified timestamp.

Usage

```
remote_last_modified(url)

local_last_modified(path)
```

Arguments

url	Character string specifying the URL of the remote file.
path	Path to local file to get the last modified timestamp for.

Value

- `remote_last_modified()`: `POSIXct` datetime representing the `Last-Modified` header timestamp, or `NA` if not provided by the server.
- `local_last_modified()`: `POSIXct` datetime of the file's last modification.

Examples

```
## Not run:
url <- "https://www2.census.gov/geo/tiger/GENZ2024/shp/cb_2024_us_state_20m.zip"
remote_last_modified(url)

path <- "data-raw/cache/tiger/GENZ2024/shp/cb_2024_us_state_20m.zip"
local_last_modified(path)

## End(Not run)
```

`remote_list` *List files at a remote HTTP directory index*

Description

Performs a `GET` request against an Apache-style autoindex URL and returns the relative file/directory hrefs listed on the page. Works with Census Bureau TIGER and GENZ directory listings.

Usage

```
remote_list(url, pattern = NULL, full_url = FALSE)
```

Arguments

`url`	Character. The directory index URL (must return `text/html`).
`pattern`	Optional regex to filter returned hrefs (e.g. `"\\.zip$"`).
`full_url`	Logical. If `TRUE`, returns fully-qualified URLs by joining `url` with each relative href. Default `FALSE`.

Value

A character vector of relative (or absolute, if `full_url = TRUE`) file/directory hrefs, `NA`s and navigation links excluded.

remote_size *Get the Size of a Remote File from its URL*

Description

Retrieves the size of a remote file by sending a HEAD request to the specified URL and extracting the Content-Length header.

Usage

```
remote_size(url)
```

Arguments

url	Character string specifying the URL of the remote file.

Value

A numeric value representing the size of the remote file in bytes.

Examples

```
## Not run:
url <- "https://www2.census.gov/geo/tiger/GENZ2024/shp/cb_2024_us_state_20m.zip"
remote_size(url)

## End(Not run)
```

shp_config_opts *ESRI Shapefile Configuration Options*

Description

Construct a gdal_config_opts() object for the ESRI Shapefile driver. These are global GDAL configuration options applied to the process (via gdalraster::set_config_option() / --config). Only options you supply are emitted.

Usage

```
shp_config_opts(
  shape_rewind_on_write = NULL,
  shape_restore_shx = NULL,
  shape_2gb_limit = NULL,
  shape_encoding = NULL,
  ...,
  .set_defaults = FALSE
)
```

Arguments

shape_rewind_on_write	Value for SHAPE_REWIND_ON_WRITE (logical -> "YES"/"NO"); whether to correct the winding order of exterior/interior rings on write. Since GDAL 3.7 the default for Polygon/MultiPolygon is "NO".
shape_restore_shx	Value for SHAPE_RESTORE_SHX (logical -> "YES"/"NO"); restore a missing/broken .shx from the .shp on open. GDAL default "NO".
shape_2gb_limit	Value for SHAPE_2GB_LIMIT (logical -> "YES"/"NO"); strictly enforce the 2 GB .shp/.dbf size limit when updating.
shape_encoding	Value for SHAPE_ENCODING (override DBF encoding with any CPLRecode() encoding; "" disables recoding).
...	Additional NAME = value options passed through verbatim alongside the typed arguments. They are coerced and validated against the driver metadata in the same way, and take precedence over a typed argument that sets the same option.
.set_defaults	Logical. If TRUE, options left unset (NULL) are filled with the driver's documented GDAL metadata defaults (via the relevant gdal_vector_driver_*_opts_defaults()); user-supplied values always take precedence. Defaults to FALSE.

Value

A gdal_config_opts() object for the ESRI Shapefile driver.

See Also

shp_open_opts(), shp_creation_opts(), gdal_config_opts()

- Shapefile Home Page
- Shapefile GDAL Driver
  - Shapefile GDAL Capabilities
  - Shapefile GDAL Open Options
  - Shapefile GDAL Layer Creation Options
  - Shapefile GDAL Configuration Options
- Shapefile C Library
- ESRI Shapefile Technical Description (PDF)
- Shapefile .SHP File API
- Attribute .DBF File API
- Xbase File Format Description
- Shapelib Code Page

Examples

```
shp_config_opts(shape_restore_shx = TRUE)
```

shp_creation_opts *ESRI Shapefile Creation Options*

Description

Construct a layer-level gdal_creation_opts() object for the ESRI Shapefile driver.

Usage

```
shp_creation_opts(
  spatial_index = NULL,
  encoding = NULL,
  resize = NULL,
  shpt = NULL,
  two_gb_limit = NULL,
  auto_repack = NULL,
  dbf_date_last_update = NULL,
  dbf_eof_char = NULL,
  ...,
  .set_defaults = FALSE
)
```

Arguments

spatial_index	Value for SPATIAL_INDEX (logical -> "YES"/"NO"); create a .qix spatial index. GDAL default "NO".
encoding	Value for ENCODING (DBF encoding written to the .cpg/header). GDAL default "LDID/87".
resize	Value for RESIZE (logical -> "YES"/"NO"); resize fields to their optimal size. GDAL default "NO".
shpt	Value for SHPT (shape type override): one of NULL/POINT/ARC/POLYGON/ MULTIPOINT (2D), the *Z/*M/*ZM measured/3D variants, or MULTIPATCH.
two_gb_limit	Value for 2GB_LIMIT (logical -> "YES"/"NO"); enforce the 2 GB .shp/.dbf size limit. GDAL default "NO".
auto_repack	Value for AUTO_REPACK (logical -> "YES"/"NO"); auto-repack when needed. GDAL default "YES".
dbf_date_last_update	Value for DBF_DATE_LAST_UPDATE (YYYY-MM-DD); modification date written in the DBF header. Defaults to the current date.
dbf_eof_char	Value for DBF_EOF_CHAR (logical -> "YES"/"NO"); write the 0x1A end-of-file character in the .dbf. GDAL default "YES".
...	Additional NAME = value options passed through verbatim alongside the typed arguments. They are coerced and validated against the driver metadata in the same way, and take precedence over a typed argument that sets the same option.

`.set_defaults` Logical. If `TRUE`, options left unset (`NULL`) are filled with the driver's documented GDAL metadata defaults (via the relevant `gdal_vector_driver_*_opts_defaults()`); user-supplied values always take precedence. Defaults to `FALSE`.

Value

A layer-level `gdal_creation_opts()` object for the `ESRI Shapefile` driver.

See Also

`shp_open_opts()`, `gdal_creation_opts()`

- Shapefile Home Page
- Shapefile GDAL Driver
  - Shapefile GDAL Capabilities
  - Shapefile GDAL Open Options
  - Shapefile GDAL Layer Creation Options
  - Shapefile GDAL Configuration Options
- Shapefile C Library
- ESRI Shapefile Technical Description (PDF)
- Shapefile .SHP File API
- Attribute .DBF File API
- Xbase File Format Description
- Shapelib Code Page

Examples

```
shp_creation_opts(spatial_index = TRUE, encoding = "UTF-8")
```

---

shp_open_opts *ESRI Shapefile Open Options*

Description

Construct a `gdal_open_opts()` object for the `ESRI Shapefile` driver.

Usage

```
shp_open_opts(
  encoding = NULL,
  dbf_date_last_update = NULL,
  adjust_type = NULL,
  adjust_geom_type = NULL,
  auto_repack = NULL,
  dbf_eof_char = NULL,
  ...,
  .set_defaults = FALSE
)
```

Arguments

encoding	Value for ENCODING (override DBF encoding with any CPLRecode() encoding; "" avoids recoding).
dbf_date_last_update	Value for DBF_DATE_LAST_UPDATE (YYYY-MM-DD); modification date written in the DBF header. Defaults to the current date.
adjust_type	Value for ADJUST_TYPE (logical -> "YES"/"NO"); read the whole .dbf to refine ambiguous Real/Integer/Integer64 field types. GDAL default "NO".
adjust_geom_type	Value for ADJUST_GEOM_TYPE. One of NO/FIRST_SHAPE/ALL_SHAPES; how the layer geometry type (notably the M dimension) is determined. GDAL default "FIRST_SHAPE".
auto_repack	Value for AUTO_REPACK (logical -> "YES"/"NO"); auto-repack the shapefile when needed. GDAL default "YES".
dbf_eof_char	Value for DBF_EOF_CHAR (logical -> "YES"/"NO"); write the 0x1A end-of-file character in the .dbf. GDAL default "YES".
...	Additional NAME = value options passed through verbatim alongside the typed arguments. They are coerced and validated against the driver metadata in the same way, and take precedence over a typed argument that sets the same option.
.set_defaults	Logical. If TRUE, options left unset (NULL) are filled with the driver's documented GDAL metadata defaults (via the relevant gdal_vector_driver_*_opts_defaults()); user-supplied values always take precedence. Defaults to FALSE.

Value

A gdal_open_opts() object for the ESRI Shapefile driver.

See Also

shp_config_opts(), shp_creation_opts(), gdal_open_opts()

- Shapefile Home Page
- Shapefile GDAL Driver
  - Shapefile GDAL Capabilities
  - Shapefile GDAL Open Options
  - Shapefile GDAL Layer Creation Options
  - Shapefile GDAL Configuration Options
- Shapefile C Library
- ESRI Shapefile Technical Description (PDF)
- Shapefile .SHP File API
- Attribute .DBF File API
- Xbase File Format Description
- Shapelib Code Page

Examples

```
shp_open_opts(encoding = "UTF-8", auto_repack = TRUE)
```

`sql_where_valid_geom` *Build a SQL* `ST_IsValid` *WHERE Clause*

Description

Constructs a SQL `ST_IsValid()` expression for filtering geometries.

Usage

```
sql_where_valid_geom(geom_col, negate = FALSE)
```

Arguments

`geom_col`	Character. The name of the geometry column.
`negate`	Logical. If `TRUE`, filters for *invalid* geometries. Default is `FALSE`.

Value

A character string containing the SQL expression.

Examples

```
sql_where_valid_geom("geom")
sql_where_valid_geom("geom", negate = TRUE)
```

`sys_available_ram` `sys_available_ram` - *System Available RAM*

Description

Get the amount of usable physical RAM available to the R session using `gdalraster::get_usable_physical_ram()`, which calls the `CPLGetUsablePhysicalRAM()` C++ function from GDAL's Common Portable Library (CPL).

Usage

```
sys_available_ram()
```

Details

This function returns the total *physical RAM usable by a process, in bytes*.

It will be limited to **2GB** for 32-bit processes.

It takes into account resource limits (virtual memory) of POSIX systems. It additionally will take into account `RLIMIT_RSS` on Linux.

On Windows, it will return the total physical RAM minus the memory used by the system and other processes, as reported by the Windows API, in bytes.

Value

A numeric scalar representing the number of bytes as a `bit64::integer64()` type (or zero `0` in case of failure).

FlatGeobuf Spatial Index RAM Check

This memory may already be partly accounted for by other processes, but is still useful for estimating how much RAM is available for processing large vector data without causing out-of-memory errors.

It is used by the `check_available_ram()` check utility which is used in `fgb_validate_spatial_index_ram()` to ensure that there is sufficient RAM to build a spatial index for a given dataset:

"The creation of the packet Hilbert R-Tree requires an amount of RAM which is at least the number of features times 83 bytes."

Examples

```
## Not run:
sys_available_ram()

## End(Not run)
```

`sys_error_code` `sys_error_code` - *System Error Codes*

Description

Get system error codes and their descriptions. If a specific code is provided, returns the name, value, and description for that code. If no code is provided, returns a tibble of all system error codes.

Usage

```
sys_error_code(code = NULL)
```

Arguments

`code`	(Optional) Integer or character string representing the system error code to look up. If `NULL` (the default), returns all system error codes. Can be one or more codes to filter by.

Value

A tibble::tibble() with the name, value, and description of the system error code(s). If one or more codes are provided, returns only the matching code(s). If no codes are found, returns NULL invisibly.

See Also

ps::errno() for the underlying system error codes data.

Examples

```
## Not run:
# Get all system error codes
sys_error_code()

# Get specific error code information
sys_error_code(2)  # Example: ENOENT (No such file or directory)

## End(Not run)
```

---

sys_num_cpus sys_num_cpus - *System Number of CPUs*

Description

Get the number of CPU cores available on the current machine using gdalraster::get_num_cpus(), which calls the internal GDAL C++ library function GDALGetCPUs().

Usage

```
sys_num_cpus()
```

Details

This method is more robust than parallel::detectCores() because it accounts for CPU affinity, container limits, and other environmental restrictions that may cap the processing pools actually available to the R session.

However, on a standard unconstrained desktop machine, it will return the same value as parallel::detectCores(logical = TRUE) because both report the total logical processing channels.

- **Number of CPUs (GDAL):** gdalraster::get_num_cpus() queries the C++ backend to see how many logical execution slots (hardware threads) are exposed by the operating system.
- **Number of Cores (parallel):** parallel::detectCores(logical = TRUE) targets the virtual threads generated by Hyper-Threading (Intel) or SMT (AMD). Conversely, running parallel::detectCores(logical = FALSE) attempts to return only the count of independent physical cores on the processor.

Value

Integer representing the number of CPU cores available to the R session.

Examples

```
## Not run:
sys_num_cpus()

## End(Not run)
```

`sys_os` `sys_os` - *System OS Name*

Description

Get the current machine's operating system name.

Usage

```
sys_os()
```

Value

Character string resulting from `Sys.info()[["sysname"]]`, which will be one of c("windows", "linux", "darwin", etc.) depending on the system.

Examples

```
## Not run:
sys_os()

## End(Not run)
```

`sys_path` `sys_path` - *System PATH*

Description

Get the Current Machine's PATH Environment Variable as a Character Vector

Usage

```
sys_path(filter = NULL)
```

Arguments

filter	Optional character string. If provided, only paths containing this string will be returned.

Value

Character vector of paths from the system's PATH environment variable, split by the appropriate path separator for the operating system. If `filter` is provided, only paths containing the filter string are included in the returned vector. If no paths match the filter, an empty character vector is returned: `character(0)`.

If `filter` is provided, only paths containing the filter string are returned.

Examples

```
## Not run:
sys_path()

## End(Not run)
```

---

sys_pid — sys_pid - *System Process ID*

Description

Get the current process ID of the R session.

Usage

```
sys_pid()
```

Value

Integer representing the current process ID.

Examples

```
## Not run:
sys_pid()

## End(Not run)
```

sys_platform sys_platform - *System Platform*

Description

Get the current machine's platform (operating system "family")

Usage

```
sys_platform()
```

Value

Character string resulting from `.Platform$OS.type`

Examples

```
## Not run:
sys_platform()

## End(Not run)
```

sys_which sys_which - *System* which

Description

Lightweight, convenience wrapper around `base::Sys.which()` and `base::normalizePath()`.

Usage

```
sys_which(x, winslash = "/", ...)
```

Arguments

x	Passed to `Sys.which()` `names` argument.
winslash	the separator to be used on Windows – ignored elsewhere. Must be one of `c("/", "\\")`.
...	Arguments passed on to `base::normalizePath` `path` character vector of file paths. `mustWork` logical: if `TRUE` then an error is given if the result cannot be determined; if `NA` then a warning.

Value

Character vector of paths, if found. If not found returns `NULL` instead of `""`.

Examples

```
## Not run:
sys_which("gdal")

## End(Not run)
```

`validate_gdal_opts` *Validate GDAL Options Against Driver Metadata*

Description

Check a `gdal_opts()` object against its driver's registered metadata: unknown option names, invalid enumerated (`string-select`) values, and invalid boolean values. Validation is advisory and non-blocking - it warns (with classed conditions) and returns a logical, leaving the decision to act at the call site.

Usage

```
validate_gdal_opts(x, driver = attr(x, "driver"), call = rlang::caller_env())
```

Arguments

`x`	A `gdal_open_opts()`, `gdal_creation_opts()`, or `gdal_config_opts()` object.
`driver`	GDAL driver short name; defaults to the object's `driver` attribute.
`call`	The execution environment of a currently running function, e.g. `caller_env()`. The function will be mentioned in error messages as the source of the error. See the `call` argument of `abort()` for more information.

Value

Invisibly, `TRUE` if valid, `FALSE` if any problems were found, or `NA` if validation could not be performed (no driver).

Examples

```
validate_gdal_opts(gdal_open_opts(LIST_ALL_TABLES = "NO", driver = "GPKG"))
```

vsi_azure *Build a* /vsiaz/ *Path*

Description

Build a /vsiaz/ Path

Usage

```
vsi_azure(container, blob)
```

Arguments

container	Azure Blob container name.
blob	Blob path within the container.

Value

Character scalar.

vsi_curl *Wrap a URL with* /vsicurl/

Description

Wrap a URL with /vsicurl/

Usage

```
vsi_curl(url)
```

Arguments

url	URL to wrap.

Value

Character vector of /vsicurl/ paths.

vsi_exists *Test Whether a VSI Path Exists*

Description

Test Whether a VSI Path Exists

Usage

```
vsi_exists(path)
```

Arguments

path	Path to test.

Value

Logical vector.

vsi_from_uri *Convert a URI to a GDAL VSI Path*

Description

Convert a URI to a GDAL VSI Path

Usage

```
vsi_from_uri(uri)
```

Arguments

uri	URI or URL.

Value

Character vector of VSI paths.

vsi_handlers *List VSI Handlers Used by a Path*

Description

List VSI Handlers Used by a Path

Usage

```
vsi_handlers(path)
```

Arguments

path	VSI path.

Value

Character vector of handler names.

vsi_meta *Get VSI File Metadata*

Description

Get VSI File Metadata

Usage

```
vsi_meta(path, domain = "HEADERS")
```

Arguments

path	Path to inspect.
domain	Metadata domain, typically "HEADERS" or "ZIP".

Value

Named character vector of metadata entries.

vsi_path *Convert a Path to a GDAL VSI Path*

Description

Convert a Path to a GDAL VSI Path

Usage

```
vsi_path(path, ...)
```

Arguments

path	A path, URL, or object with a `vsi_path()` method.
...	Passed to methods.

Value

Character vector of VSI paths.

vsi_size *Get the Size of a VSI Path*

Description

Get the Size of a VSI Path

Usage

```
vsi_size(path)
```

Arguments

path	Path to inspect.

Value

The size of the file/object in bytes, formatted using `rlang::as_bytes()`.

vsi_strip *Strip the Outermost VSI Handler*

Description

Strip the Outermost VSI Handler

Usage

```
vsi_strip(path, recurse = TRUE)
```

Arguments

path	Path to strip.
recurse	Logical; if TRUE (default), strips all nested VSI handlers, otherwise only the outermost one.

Value

Character vector with the outermost VSI handler removed.

vsi_sync *Sync Between VSI Paths*

Description

Sync Between VSI Paths

Usage

```
vsi_sync(src, dst, ...)
```

Arguments

src	Source path.
dst	Destination path.
...	Passed to gdalraster::vsi_sync().

Value

Invisibly returns the sync result from GDAL.

vsi_type *Get the Type of a VSI Path*

Description

Get the Type of a VSI Path

Usage

```
vsi_type(path)
```

Arguments

path	Path to inspect.

Value

Character vector of path types.

vsi_zip *Wrap a Path with* /vsizip/

Description

Wrap a Path with /vsizip/

Usage

```
vsi_zip(path)
```

Arguments

path	Path to a ZIP archive.

Value

Character vector of /vsizip/ paths.

vsi_zip_curl *Build a* /vsizip//vsicurl/ *Path*

Description

Build a /vsizip//vsicurl/ Path

Usage

```
vsi_zip_curl(url, inner = NULL)
```

Arguments

url	Remote ZIP URL.
inner	Optional inner file path within the ZIP archive.

Value

Character scalar.

xml_parse_gdal_driver_config_opts *Parse GDAL Driver Configuration Options from XML*

Description

Parses the configuration options for a GDAL driver from the provided XML document. This function is specifically designed to extract the configuration options listed in the "Configuration Options" section of a GDAL driver's documentation page, which is typically structured as an unordered list (<ul>) with list items (<li>) containing the option details.

The function looks for the first <ul> element within the section with id="configuration-options" and extracts the option name, description, default value, and possible values (if specified in brackets). The resulting data is returned as a tibble with columns for name, description, scope, default, and values (a list-column containing character vectors of possible values).

Usage

```
xml_parse_gdal_driver_config_opts(
  xml,
  scope = "all",
  driver = NULL,
  type = "config",
  call = rlang::caller_env()
)
```

Arguments

xml	The XML document to parse, typically obtained from a GDAL driver's documentation page.
scope	The scope of the options being parsed, such as "vector", "raster", or "all". This is used to categorize the options based on their applicability to different data types. Defaults to "all".
driver	The name of the GDAL driver for which the options are being parsed. This is used for labeling purposes in the resulting tibble. Defaults to NULL.
type	The type of options being parsed, such as "config", "open", or "creation". This is used for labeling purposes in the resulting tibble. Defaults to "config".
call	The calling function, used for error handling and messaging.

Value

A tibble with columns for name, description, scope, default, and values

---

xml_parse_gdal_options

Parse XML for GDAL Options

Description

Parses the XML from GDAL driver metadata into structured tbl_df tibbles.

The function is meant to be flexible enough to be able to parse any of the possible XML metadata structures from GDAL's registered (vector) driver's metadata: DMD_OPENOPTIONLIST, DMD_CREATIONOPTIONLIST, and DS_LAYER_CREATIONOPTIONLIST.

Usage

```
xml_parse_gdal_options(
  xml,
  driver = NA_character_,
  type = c("config", "open", "creation"),
  sub_type = NA_character_,
  scope = c("vector", "raster", "all"),
  call = rlang::caller_env()
)
```

Arguments

xml	The XML to be parsed. Must be either a valid XML character string or an xml2::xml_document object.
driver	(Optional) GDAL driver name added to the resulting tibble::tibble(). Useful for when merging options across multiple drivers. Defaults to NA and is not validated against the registered GDAL drivers, so it should be used with caution and primarily for internal use.

type	The option type being parsed, one of "config", "open", or "creation".
sub_type	For type = "creation", the creation option level, one of "dataset" or "layer". Ignored (forced to NA) for other types.
scope	(Optional) The scope of the options being parsed in terms of the supported data types, i.e. vector vs. raster or both. For example, GPKG has scopes defined for "vector", "raster", "raster,vector", and NA. Use this argument to filter out for only the options for a specific scope. Will always include the NA scoped options, as those are not explicitly defined for a specific scope and are likely applicable to all scopes. Defaults to NULL and will not filter for any specific scope, returning all options regardless of scope. Provided value must be one of "vector", "raster", or "all" if not NULL. Note that the scope values are not standardized across all GDAL drivers, so use with caution and always check the resulting tibbles for the expected scope values when working with drivers supporting both raster and vector data.
call	The calling function, used for error handling and messaging.

Value

tibble::tibble() with fields for name, description, scope, type, default, and values (a list-column with a character vector for the possible values).

Additional fields for driver and opt_type may be included if those parameters are provided.

Examples

```
# parse DS_LAYER_CREATIONOPTIONLIST
gdalraster::gdal_get_driver_md("GPKG", mdi_name = "DS_LAYER_CREATIONOPTIONLIST") |>
  xml_parse_gdal_options()

# parse DMD_CREATIONOPTIONLIST
gdalraster::gdal_get_driver_md("GPKG", mdi_name = "DMD_CREATIONOPTIONLIST") |>
  xml_parse_gdal_options()

# parse DMD_OPENOPTIONLIST
gdalraster::gdal_get_driver_md("GPKG", mdi_name = "DMD_OPENOPTIONLIST") |>
  xml_parse_gdal_options()
```

Index

∗ **checks**
checks, 14
∗ **gpq**
gpq_meta, 42
∗ **utils**
checks, 14

abort(), *6*, *8–13*, *63*
as_config_option, 4
as_config_option(), *21, 22*
as_gdal_args, 4
as_gdal_args(), *4*, *21, 22*
as_gdal_config_opts, 6
as_gdal_creation_opts, 7
as_gdal_open_opts, 8
as_gdal_vsi_opts, 9
as_gdalg, 10

base::.class2(), *11*
base::inherits(), *11*
base::normalizePath, *62*
base::normalizePath(), *62*
base::readRenviron(), *47*
base::Sys.which(), *62*
bit64::integer64(), *58*

check_available_ram, 11
check_available_ram(), *58*
check_inherits, 11
check_inherits2 *(*check_inherits*)*, 11
check_inherits_all *(*check_inherits*)*, 11
check_inherits_any *(*check_inherits*)*, 11
check_not_empty, 13
check_string, 13
checks, 14

fgb_config_opts, 14
fgb_creation_opts, 15
fgb_creation_opts(), *14*, *16*, *22*
fgb_open_opts, 16
fgb_open_opts(), *14, 15*
fgb_validate_spatial_index_ram, 17
fgb_validate_spatial_index_ram(), *58*

gdal_config, 17
gdal_config_links, 18
gdal_config_opts, 18
gdal_config_opts(), *4*, *6*, *14*, *18*, *21*, *24*, *28*, *29*, *32, 33*, *52, 53*, *63*
gdal_creation_opts, 19
gdal_creation_opts(), *7, 8*, *15*, *19*, *21*, *25*, *29*, *31*, *33, 34*, *38–40*, *54, 55*, *63*
gdal_driver_names *(*gdal_drivers*)*, 19
gdal_drivers, 19
gdal_drivers(), *23*
gdal_open_opts, 20
gdal_open_opts(), *8, 9*, *16*, *20, 21*, *26*, *31, 32*, *35, 36*, *42, 43*, *55, 56*, *63*
gdal_opts, 21
gdal_opts(), *4, 5*, *22*, *63*
gdal_render, 22
gdal_render(), *21, 22*
gdal_vector_driver_capabilities, 22
gdal_vector_driver_capabilities(), *20*
gdal_vector_driver_config_opts, 23
gdal_vector_driver_config_opts(), *27*
gdal_vector_driver_config_opts_defaults *(*gdal_vector_driver_config_opts*)*, 23
gdal_vector_driver_config_opts_types *(*gdal_vector_driver_config_opts*)*, 23
gdal_vector_driver_config_opts_values *(*gdal_vector_driver_config_opts*)*, 23
gdal_vector_driver_creation_opts, 24
gdal_vector_driver_creation_opts(), *27*
gdal_vector_driver_creation_opts_defaults *(*gdal_vector_driver_creation_opts*)*, 24

gdal_vector_driver_creation_opts_types (gdal_vector_driver_creation_opts), 24
gdal_vector_driver_creation_opts_values (gdal_vector_driver_creation_opts), 24
gdal_vector_driver_open_opts, 25
gdal_vector_driver_open_opts(), *27*
gdal_vector_driver_open_opts_defaults (gdal_vector_driver_open_opts), 25
gdal_vector_driver_open_opts_types (gdal_vector_driver_open_opts), 25
gdal_vector_driver_open_opts_values (gdal_vector_driver_open_opts), 25
gdal_vector_driver_opt_defaults (gdal_vector_driver_opts), 26
gdal_vector_driver_opt_types (gdal_vector_driver_opts), 26
gdal_vector_driver_opt_values (gdal_vector_driver_opts), 26
gdal_vector_driver_opts, 26
gdal_vector_driver_opts(), *20*, *24–26*
GDAL_VECTOR_DRIVERS, *26*
gdal_vsi_opts, 28
gdal_vsi_opts(), *4*, *9*, *10*, *21*, *28*
gdalraster::gdal_alg(), *4*, *5*, *21*
gdalraster::gdal_formats(), *19*
gdalraster::gdal_run(), *4*, *21*
gdalraster::get_num_cpus(), *59*
gdalraster::get_usable_physical_ram(), *57*
gdalraster::set_config_option(), *4*, *18*, *21*, *52*
gdalraster::vsi_set_path_option(), *9*
gdalraster::vsi_sync(), *68*
gdb_config_opts, 28
gdb_creation_opts, 29
gdb_creation_opts(), *29*, *32*
gdb_open_opts, 31
gdb_open_opts(), *22*, *29*, *31*
gpkg_config_opts, 32
gpkg_creation_opts, 33
gpkg_creation_opts(), *33*, *36*
gpkg_open_opts, 35
gpkg_open_opts(), *22*, *33*, *34*, *37*
gpkg_prelude_pragmas, 37
gpkg_prelude_pragmas(), *35*, *36*
gpq_creation_opts, 38
gpq_creation_opts(), *22*, *43*
gpq_file_info, 40
gpq_file_info(), *41*, *42*
gpq_geo_metadata, 41
gpq_geo_metadata(), *41*, *42*
gpq_inspect, 41
gpq_inspect(), *42*
gpq_meta, 42
gpq_open_opts, 42
gpq_open_opts(), *40*
gpq_row_groups, 43
gpq_row_groups(), *41*, *42*
gpq_schema, 44
gpq_schema(), *45*
gpq_schema_info, 45
gpq_schema_info(), *41*, *42*, *44*

httr2::request(), *50*
httr2::resp_headers(), *50*
httr2::response(), *50*

is_vsi_path, 45

local_hash, 46
local_last_modified (remote_last_modified), 50

nanoparquet::read_parquet_metadata(), *42*

ping, 46
ps::errno(), *59*

read_renviron, 47
remote_download, 47
remote_exists, 48
remote_hash, 49
remote_head, 49
remote_last_modified, 50
remote_list, 51
remote_size, 52
rlang::as_bytes(), *67*
rlang::check_string(), *13*
rlang::inherits_all(), *11*
rlang::inherits_any(), *11*
rlang::is_empty(), *13*

shp_config_opts, 52
shp_config_opts(), *56*
shp_creation_opts, 54
shp_creation_opts(), *53*, *56*
shp_open_opts, 55
shp_open_opts(), *22*, *53*, *55*
sql_where_valid_geom, 57
sys_available_ram, 57
sys_available_ram(), *11*
sys_error_code, 58
sys_num_cpus, 59
sys_os, 60
sys_path, 60
sys_pid, 61
sys_platform, 62
sys_which, 62

tibble::tibble(), *19, 20*, *24–27*, *44*, *59*, *71, 72*

validate_gdal_opts, 63
vsi_azure, 64
vsi_curl, 64
vsi_exists, 65
vsi_from_uri, 65
vsi_handlers, 66
vsi_meta, 66
vsi_path, 67
vsi_size, 67
vsi_strip, 68
vsi_sync, 68
vsi_type, 69
vsi_zip, 69
vsi_zip_curl, 70

xml_parse_gdal_driver_config_opts, 70
xml_parse_gdal_options, 71

yyjsonr::read_json_str(), *42*