Package 'vapour'

April 4, 2025

Title Lightweight Access to the 'Geospatial Data Abstraction Library' ('GDAL')

Version 0.11.0

Description Provides low-level access to 'GDAL' functionality.
'GDAL' is the 'Geospatial Data Abstraction Library' a translator for raster and vector geospatial data formats
that presents a single raster abstract data model and single vector abstract data model to the calling application
for all supported formats <https://gdal.org/>. This package is focussed on providing exactly and only what GDAL does, to enable developing further tools.

Depends R (>= 3.3.0)

License GPL-3

Encoding UTF-8

LazyData true

LinkingTo Rcpp

Imports jsonlite, nanoarrow, Rcpp, utils, wk

RoxygenNote 7.3.2

Suggests testthat, knitr, markdown, rmarkdown, spelling

SystemRequirements libgdal-dev, GDAL (>= 2.2.3), PROJ (>= 4.8.0)

VignetteBuilder knitr

URL https://github.com/hypertidy/vapour,
https://hypertidy.github.io/vapour/

BugReports https://github.com/hypertidy/vapour/issues

Language en-US

NeedsCompilation yes

Author Michael Sumner [aut, cre] (<https://orcid.org/0000-0002-2471-7511>),
Simon Wotherspoon [ctb] (RasterIO configuration for resampling options),
Mark Padgham [ctb] (helped get started :)),
Edzer Pebesma [ctb] (wrote the field-read handling, adapted here from

sf),
Roger Bivand [ctb] (wrote configure.ac, adapted here from rgdal),
Jim Hester [ctb, cph] (wrote CollectorList.h, copied here from fs package),
Timothy Keitt [ctb] (wrote GetPointsInternal copied here from rgdal2 package),
Jeroen Ooms [ctb] (tweaked build process, provided Windows build tools),
Dale Maschette [ctb] (created the hex logo),
Joseph Stachelek [ctb],
Even Rouault [ctb] (primary author of the COG format and its use of the GDALwarp app-library, example code used by the warper function here),
Robert Hijmans [ctb] (code in terra package used as example/inspiration),
Dewey Dunnington [ctb] (wrote the columnar-access mode streaming Arrow support),
Tomas Kalibera [ctb]

Maintainer Michael Sumner <mdsumner@gmail.com>

Repository CRAN

Date/Publication 2025-04-04 04:40:02 UTC

Contents

---

vapour-package *vapour*

Description

A lightweight GDAL API package for R.

Details

Provides low-level access to 'GDAL' functionality for R packages. The aim is to minimize the level of interpretation put on the 'GDAL' facilities, to enable direct use of it for a variety of purposes. 'GDAL' is the 'Geospatial Data Abstraction Library' a translator for raster and vector geospatial data formats that presents a single raster abstract data model and single vector abstract data model to the calling application for all supported formats https://gdal.org/.

Lightweight means we access parts of the GDAL API as near as possible to their native usage. GDAL is not a lightweight library, but provide a very nice abstraction over format details for a very large number of different formats.

Functions for raster and vector sources are included.

vapour_all_drivers	list of all available drivers, with type and features
vapour_driver	report short name of driver that will be used for a data source
vapour_gdal_version	report version of GDAL in use
vapour_srs_wkt	produce WKT projection string from various projection string inputs
vapour_vsi_list	report contents of VSI sources

vapour_raster_gcp	return internal ground control points, if present
vapour_raster_info	structural metadata of a source
vapour_read_raster	read data direct from a window of a raster band source
vapour_sds_names	list individual raster sources in a source containing subdatasets
vapour_warp_raster	read data direct from a raster source into a specific window

vapour_driver	report name of the driver used for a given source

vapour_geom_name	report attribute name of geometry
vapour_geom_summary	report simple properties of each feature geometry
vapour_layer_names	list names of vector layers in a data source
vapour_layer_info	list of data source, driver, layer name/s, fields, feature count, projection
vapour_read_extent	read the extent, or bounding box, of geometries in a layer
vapour_read_fields	read attributes of features in a layer, the columnar data associated with each geometry
vapour_read_geometry	read geometry in binary (blob, WKB) form
vapour_read_geometry_ia	read geometry by index, arbitrary
vapour_read_geometry_ij	read geometry by sequential index, i to j
vapour_read_geometry_text	read geometry in text form, various formats
vapour_read_names	read the 'names' of features in a layer, the 'FID'
vapour_read_type	read the GDAL types of attributes
vapour_report_fields	report internal type of each attribute by name

As far as possible vapour aims to minimize the level of interpretation provided for the functions, so that developers can choose how things are implemented. Functions return raw lists or vectors rather than data frames or classed types.

options

The following options can be set to control global behaviour.

`Sys.getenv("vapour.sql.dialect")`	the current SQL dialect in use

SQL dialect

The SQL dialect can be set to "" (empty string), "OGRSQL", or "SQLITE".

The empty string indicates that the native dialect will be used, see [OGRSQL and SQLITE for GDAL(https://gdal.org/en/stable/user/ogr_sql_sqlite_dialect.html) and the GDAL_DMD_SUPPORTED_SQL_DIALECTS development documentation.

Setting "NATIVE" as an alias for "" is quite recent and has not been tested with vapour, similarly no testing has been done with non OGRSQL-native or SQLITE-native drivers yet.

Author(s)

Maintainer: Michael Sumner <mdsumner@gmail.com> (ORCID)

Other contributors:

- Simon Wotherspoon (RasterIO configuration for resampling options) [contributor]
- Mark Padgham (helped get started :)) [contributor]
- Edzer Pebesma (wrote the field-read handling, adapted here from sf) [contributor]
- Roger Bivand (wrote configure.ac, adapted here from rgdal) [contributor]
- Jim Hester (wrote CollectorList.h, copied here from fs package) [contributor, copyright holder]
- Timothy Keitt (wrote GetPointsInternal copied here from rgdal2 package) [contributor]

- Jeroen Ooms (tweaked build process, provided Windows build tools) [contributor]
- Dale Maschette (created the hex logo) [contributor]
- Joseph Stachelek [contributor]
- Even Rouault (primary author of the COG format and its use of the GDALwarp app-library, example code used by the warper function here) [contributor]
- Robert Hijmans (code in terra package used as example/inspiration) [contributor]
- Dewey Dunnington (wrote the columnar-access mode streaming Arrow support) [contributor]
- Tomas Kalibera [contributor]

See Also

Useful links:

- https://github.com/hypertidy/vapour
- https://hypertidy.github.io/vapour/
- Report bugs at https://github.com/hypertidy/vapour/issues

`buildvrt` *Build vrt, special case "-separate"*

Description

Build vrt, special case "-separate"

Usage

```
buildvrt(dsn)
```

Arguments

`dsn`	one or more raster sources

Value

a character string of the built vrt, multiple sources treated as bands

Examples

```
f <- system.file("extdata/sst.tif", package = "vapour", mustWork = TRUE)
vrt <- buildvrt(c(f, vapour_vrt(f)))
writeLines(vrt)
```

gdal_raster_data *General raster read and convert*

Description

The warper is used to convert source/s to an output file or to data in memory.

Usage

```
gdal_raster_data(
  dsn,
  target_crs = NULL,
  target_dim = NULL,
  target_ext = NULL,
  target_res = NULL,
  resample = "near",
  bands = 1L,
  band_output_type = NULL,
  options = character(),
  include_meta = TRUE
)

gdal_raster_dsn(
  dsn,
  target_crs = NULL,
  target_dim = NULL,
  target_ext = NULL,
  target_res = NULL,
  resample = "near",
  bands = NULL,
  band_output_type = NULL,
  options = character(),
  out_dsn = tempfile(fileext = ".tif"),
  include_meta = TRUE
)

gdal_raster_image(
  dsn,
  target_crs = NULL,
  target_dim = NULL,
  target_ext = NULL,
  target_res = NULL,
  resample = "near",
  bands = NULL,
  band_output_type = NULL,
  options = character(),
  include_meta = TRUE
```

```
)

gdal_raster_nara(
  dsn,
  target_crs = NULL,
  target_dim = NULL,
  target_ext = NULL,
  target_res = NULL,
  resample = "near",
  bands = NULL,
  band_output_type = NULL,
  options = character(),
  include_meta = TRUE
)
```

Arguments

dsn	data sources, files, urls, db strings, vrt, etc
target_crs	projection of the target grid
target_dim	dimension of the target grid
target_ext	extent of the target grid
target_res	resolution of the target grid
resample	resampling algorithm used
bands	band or bands to include, default is first band only (use NULL or a value less that one to obtain all bands)
band_output_type	specify the band type, see vapour_read_raster
options	general options passed to gdal warper
include_meta	metadata is attached, turn off by setting this to FALSE
out_dsn	file name for output "_dsn"

Details

Two functions 'gdal_raster_data' and 'gdal_raster_dsn' act like the gdalwarp command line tool, a convenience third function 'gdal_raster_image()' works especially for image data.

Value

pixel values in a list vector per band, or a list of file paths

Examples

```
dsn <- system.file("extdata/sst.tif", package = "vapour")
## do nothing, get native
X <- gdal_raster_data(dsn)

## set resolution (or dimension, extent, crs, or combination thereof - GDAL
```

```
## will report/resolve incompatible opts)
X1 <- gdal_raster_data(dsn,  target_res = 1)

## add a cutline, and cut to it using gdal warp args

if (interactive()) {
 cutline <- tempfile(fileext = ".csv")
 wkt <- "POLYGON ((142 -41, 149 -41, 146 -58, 142 -41))"
 write.csv(data.frame(id = 1, WKT = wkt), cutline, row.names = FALSE)
 X1c <- gdal_raster_data(dsn,  target_res = .5,
     options = c("-cutline",cutline, "-crop_to_cutline"))
 file.remove(cutline)
}

## warp whole grid to given res
X2 <- gdal_raster_data(dsn,  target_res = 25000, target_crs = "EPSG:32755")

## specify exactly (as per vapour originally)
X3 <- gdal_raster_data(dsn,  target_ext = c(-1, 1, -1, 1) * 8e6,
 target_dim = c(512, 678), target_crs = "+proj=stere +lon_0=147 +lat_0=-90")

X4 <- gdal_raster_dsn(dsn, out_dsn = tempfile(fileext = ".tif"))
```

`sst_c` *SST contours*

Description

Southern Ocean GHRSST contours in sf data frame from 2017-07-28, read from

Details

podaac-ftp.jpl.nasa.gov/allData/ghrsst/data/GDS2/L4 GLOB/JPL/MUR/v4.1/2017/209/ 20170728090000-JPL-L4_GHRSST-SSTfnd-MUR-GLOB-v02.0-fv04.1.nc

See data-raw/sst_c.R for the derivation column `sst_c` in Celsius.

Also stored in FlatGeoBuf format in `system.file("extdata/sst_c.fgb", package = "vapour")`

Examples

```
f <- system.file("extdata/sst_c.fgb", package = "vapour")

## create a class-less form of the data in the 'sst_c.fgb' file  with GeoJSON geometry
atts <- vapour_read_fields(f)
dat <- as.data.frame(atts, stringsAsFactors = FALSE)
dat[["json"]] <- vapour_read_geometry_text(f)
names(dat)
names(sst_c)
```

tas_wkt *Example WKT coordinate reference system*

Description

A Lambert Azimuthal Equal Area Well-Known-Text string for a region centred on Tasmania.

Details

Created from '+proj=laea +lon_0=147 +lat_0=-42 +datum=WGS84'. For use in a future warping example.

vapour_create *Create raster file*

Description

This is in an incomplete interface to raster writing, for exploring.

Usage

```
vapour_create_options(driver = "GTiff")

vapour_create(
  filename,
  driver = "GTiff",
  extent = c(-180, 180, -90, 90),
  dimension = c(2048, 1024),
  projection = "EPSG:4326",
  n_bands = 1L,
  overwrite = FALSE,
  datatype = "Float32",
  options = vapour_create_options(driver)
)
```

Arguments

driver	GDAL driver to use (GTiff is default, and recommended)
filename	filename/path to create
extent	xmin,xmax,ymin,ymax 4-element vector
dimension	dimension of the output, X * Y
projection	projection of the output, best to use a full WKT but any string accepted
n_bands	number of bands in the output, default is 1

overwrite	not TRUE by default
datatype	the name of a GDAL datatype ('Float32', 'Int64', etc)
options	character vector of creation of options for the driver in use `c('COMPRESS=DEFLATE')` note how these are constructed (no '-co' element)

Details

If GeoTIFF is used (`driver = "GTiff"`, recommended) then the output is tiled 512x512, and has DEFLATE compression, and is sparse when created (no values are initiated, so the file is tiny).

Note that there is no restriction on where you can read or write from, the responsibility is yours. There is no auto driver detection done for the file format, it's up to you to set the file extension *and* the driver.

File is created using CreateCopy from a VRT in memory. This is so that we can instantiate COG layer with 'driver = "COG"'. Please note that performance is best for GTiff itself, with 'SPARSE_OK=YES'. We don't yet know how to instantiate a large COG with overviews.

There are default creation options set for COG and GTiff drivers, see 'vapour_create_options(driver "GTiff")' for what those are.

Value

the file path that was created

Examples

```
tfile <- tempfile(fileext = ".tif")
if (!file.exists(tfile)) {
 vapour_create(tfile, extent = c(-1, 1, -1, 1) * 1e6,
                     dimension = c(128, 128),
                     projection = "+proj=laea")
 file.remove(tfile)
}
```

---

vapour_crs_is_lonlat *Is the CRS string representative of angular coordinates*

Description

Returns `TRUE` if this is longitude latitude data. Missing, malformed, zero-length values are disallowed.

Usage

```
vapour_crs_is_lonlat(crs)
```

Arguments

crs	character string of length 1

Value

logical value TRUE for lonlat, FALSE otherwise

Examples

```
vapour_gdal_version() ## versions to catch problems with string input
vapour_proj_version()
vapour_crs_is_lonlat("+proj=aeqd +lon_0=147 +lat_0=-42")
vapour_crs_is_lonlat("EPSG:4326")
vapour_srs_wkt("+proj=laea")
vapour_crs_is_lonlat("+proj=laea +type=crs")
vapour_crs_is_lonlat("OGC:CRS84")
vapour_crs_is_lonlat("WGS84")
vapour_crs_is_lonlat("NAD27")
vapour_crs_is_lonlat("EPSG:3031")
```

---

vapour_gdal_version *GDAL version and drivers.*

Description

Return information about the GDAL library in use.

Usage

```
vapour_gdal_version()

vapour_proj_version()

vapour_all_drivers()

vapour_driver(dsource)
```

Arguments

dsource	data source string (i.e. file name or URL or database connection string)

Details

vapour_gdal_version returns the version of GDAL as a string. This corresponds to the "–version" as described for "GDALVersionInfo". GDAL documentation.

vapour_all_drivers returns the names and capabilities of all available drivers, in a list. This contains:

- driver the driver (short) name
- name the (long) description name
- vector logical vector indicating a vector driver

- raster logical vector indicating a raster driver
- create driver can create (note vapour provides no write capacity)
- copy driver can copy (note vapour provides no write capacity)
- virtual driver has virtual capabilities ('vsi')

vapour_driver() returns the short name of the driver, e.g. 'GPKG' or 'GTiff', to get the long name and other properties use vapour_all_drivers() and match on 'driver'.

Value

please see Details, character vectors or lists of character vectors

Examples

```
vapour_gdal_version()

drv <- vapour_all_drivers()

f <- system.file("extdata/sst_c.fgb", package = "vapour")
vapour_driver(f)

as.data.frame(drv)[match(vapour_driver(f), drv$driver), ]
```

---

vapour_geolocation *Retrieve geolocation information for a dataset*

Description

Value is a named vector in a list.

Usage

```
vapour_geolocation(x, sds = NULL)
```

Arguments

x	data source string (i.e. file name or URL or database connection string)
sds	a subdataset number, if necessary

Details

If no geolocation exist the return value is an empty list.

Value

list with a single character vector

Examples

```
drivers <- vapour_all_drivers()
ok <- drivers$raster[ drivers$driver == "netCDF"]
if (isTRUE(ok) && interactive()) {
 vapour_geolocation(system.file("extdata/gdal/geos_rad.nc", package = "vapour"), 0L)
}
```

---

vapour_geom_name *Read geometry column name*

Description

There might be one or more geometry column names, or it might be an empty string.

Usage

```
vapour_geom_name(dsource, layer = 0L, sql = "")
```

Arguments

dsource	data source name (path to file, connection string, URL)
layer	integer of layer to work with, defaults to the first (0) or the name of the layer
sql	if not empty this is executed against the data source (layer will be ignored)

Details

It might be "", or "geom", or "*ogr_geometry*" - the last is a default name given when SQL is executed by GDAL but there was no geometry name, and 'SELECT * ' or equivalent was used.

This feature is required by the DBI backend work in RGDALSQL, so that when `SELECT *` is used we can give a reasonable name to the geometry column which is obtained separately.

Value

character vector of geometry column name/s

Examples

```
file <- system.file("extdata/tab/list_locality_postcode_meander_valley.tab", package = "vapour")
vapour_geom_name(file)  ## empty string
```

vapour_geom_summary *Summary of available geometry*

Description

Read properties of geometry from a source, optionally after SQL execution.

Usage

```
vapour_geom_summary(
  dsource,
  layer = 0L,
  sql = "",
  limit_n = NULL,
  skip_n = 0,
  extent = NA
)
```

Arguments

dsource	data source name (path to file, connection string, URL)
layer	integer of layer to work with, defaults to the first (0) or the name of the layer
sql	if not empty this is executed against the data source (layer will be ignored)
limit_n	an arbitrary limit to the number of features scanned
skip_n	an arbitrary number of features to skip
extent	apply an arbitrary extent, only when 'sql' used (must be 'ex = c(xmin, xmax, ymin, ymax)' but sp bbox, sf bbox, and raster extent also accepted)

Details

Use limit_n to arbitrarily limit the number of features queried.

Value

list containing the following

- FID the feature id value (an integer, usually sequential)
- valid_geometry logical value if a non-empty geometry is available
- type integer value of geometry type from GDAL enumeration
- xmin, xmax, ymin, ymax numeric values of the extent (bounding box) of each geometry

Examples

```
file <- "list_locality_postcode_meander_valley.tab"
mvfile <- system.file(file.path("extdata/tab", file), package="vapour")
vapour_geom_summary(mvfile, limit_n = 3L)

gsum <- vapour_geom_summary(mvfile)
plot(NA, xlim = range(c(gsum$xmin, gsum$xmax), na.rm = TRUE),
         ylim = range(c(gsum$ymin, gsum$ymax), na.rm = TRUE))
rect(gsum$xmin, gsum$ymin, gsum$xmax, gsum$ymax)
text(gsum$xmin, gsum$ymin, labels = gsum$FID)
```

---

vapour_layer_extent *Read layer extent*

Description

Extent of all features in entire layer, possibly after execution of sql query and input extent filter.

Usage

```
vapour_layer_extent(dsource, layer = 0L, sql = "", extent = 0, ...)
```

Arguments

dsource	data source name (path to file, connection string, URL)
layer	integer of layer to work with, defaults to the first (0) or the name of the layer
sql	if not empty this is executed against the data source (layer will be ignored)
extent	optional extent (xmin,xmax,ymin,ymax)
...	unused

Value

vector of numeric values xmin,xmax,ymin,ymax

See Also

vapour_read_extent vapour_layer_info

Examples

```
file <- "list_locality_postcode_meander_valley.tab"
## A MapInfo TAB file with polygons
mvfile <- system.file(file.path("extdata/tab", file), package="vapour")
vapour_layer_extent(mvfile)
```

vapour_layer_info *Read GDAL layer info*

Description

Read GDAL layer information for a vector data source.

Usage

```
vapour_layer_info(
  dsource,
  layer = 0L,
  sql = "",
  extent = NA,
  count = TRUE,
  ...
)
```

Arguments

dsource	data source name (path to file, connection string, URL)
layer	integer of layer to work with, defaults to the first (0) or the name of the layer
sql	if not empty this is executed against the data source (layer will be ignored)
extent	apply an arbitrary extent, only when 'sql' used (must be 'ex = c(xmin, xmax, ymin, ymax)' but sp bbox, sf bbox, and raster extent also accepted)
count	logical to control if count calculated and returned, TRUE by default (set to FALSE to avoid the extra calculation and missing value is the result)
...	unused, reserved for future use

Details

Set extent and/or count to FALSE to avoid calculating them if not needed, it might take some time.

The layer information elements are

dsn the data source name

driver the short name of the driver used

layer the name of the layer queried

layer_names the name/s of all available layers (see vapour_layer_names)

fields a named vector of field types (see vapour_report_fields)

count the number of features in this data source (can be turned off to avoid the extra work count)

extent the extent of all features xmin, xmax, ymin, ymax (can be turned off to avoid the extra work extent)

projection a list of character strings, see next

`$projection` is a list of various formats of the projection metadata. Use `$projection$Wkt` as most authoritative, but we don't enter into the discussion or limit what might be done with this (that's up to you). Currently we see `c("Proj4", "MICoordSys", "PrettyWkt", "Wkt", "EPSG", "XML")` as names of this `$projection` element.

To get the geometry type/s of a layer see `vapour_read_type()`.

Value

list with a list of character vectors of projection metadata, see details

See Also

vapour_geom_name vapour_layer_names vapour_report_fields vapour_read_fields vapour_driver vapour_read_names

Examples

```
file <- "list_locality_postcode_meander_valley.tab"
## A MapInfo TAB file with polygons
mvfile <- system.file(file.path("extdata/tab", file), package="vapour")
info <- vapour_layer_info(mvfile)
names(info$projection)

## info depends on the query/spatial-filter
vapour_layer_info(mvfile, extent = c(412000,  420000, 5352612.8, 5425154.3),
  sql = "SELECT * FROM list_locality_postcode_meander_valley")$count
```

---

vapour_layer_names *Read GDAL layer names*

Description

Obtain the names of available layers from a GDAL vector source.

Usage

```
vapour_layer_names(dsource, ...)
```

Arguments

`dsource`	data source name (path to file, connection string, URL)
`...`	arguments ignore for deprecated compatibility (no 'sql' argument any longer)

Details

Some vector sources have multiple layers while many have only one. Shapefiles for example have only one, and the single layer gets the file name with no path and no extension. GDAL provides a quirk for shapefiles in that a directory may act as a data source, and any shapefile in that directory acts like a layer of that data source. This is a little like the one-or-many sleight that exists for raster data sources with subdatasets (there's no way to virtualize single rasters into a data source with multiple subdatasets, oh except by using VRT....)

See vapour_sds_names for more on the multiple topic.

Value

character vector of layer names

Examples

```
file <- "list_locality_postcode_meander_valley.tab"
mvfile <- system.file(file.path("extdata/tab", file), package="vapour")
vapour_layer_names(mvfile)
```

---

vapour_raster_gcp *Raster ground control points*

Description

Return any ground control points for a raster data set, if they exist.

Usage

```
vapour_raster_gcp(x, ...)
```

Arguments

x	data source string (i.e. file name or URL or database connection string)
...	ignored currently

Details

Pixel and Line coordinates do not correspond to cells in the underlying raster grid, they refer to the index space of that array in 0, ncols and 0, nrows. They are usually a subsample of the grid and may not align with the grid spacing itself (though they often do in satellite remote sensing products).

The coordinate system of the GCPs is currently not read.

Value

list with

- `Pixel` the pixel coordinate
- `Line` the line coordinate
- `X` the X coordinate of the GCP
- `Y` the Y coordinate of the GCP
- `Z` the Z coordinate of the GCP (usually zero)

Examples

```
## this file has no ground control points
## they are rare, and tend to be in large files
f <- system.file("extdata", "sst.tif", package = "vapour")
vapour_raster_gcp(f)

## a very made-up example with no real use
f1 <- system.file("extdata/gcps", "volcano_gcp.tif", package = "vapour")
vapour_raster_gcp(f1)
```

---

vapour_raster_info *Raster information*

Description

Return the basic structural metadata of a raster source understood by GDAL. Subdatasets may be specified by number, starting at 1. See vapour_sds_names for more.

Usage

```
vapour_raster_info(x, ..., sds = NULL, min_max = FALSE)
```

Arguments

`x`	data source string (i.e. file name or URL or database connection string)
`...`	currently unused
`sds`	a subdataset number, if necessary
`min_max`	logical, control computing min and max values in source ('FALSE' by default)

Details

The structural metadata are

extent the extent of the data, xmin, xmax, ymin, ymax - these are the lower left and upper right corners of pixels

geotransform the affine transform

dimension dimensions x-y, columns*rows

minmax numeric values of the computed min and max from the first band (optional)

block dimensions x-y of internal tiling scheme

projection text version of map projection parameter string

bands number of bands in the dataset

projstring the proj string version of 'projection'

nodata_value not implemented

overviews the number and size of any available overviews

filelist the list of files involved (may be none, and so will be a single NA character value)

datatype the band type name, in GDAL form 'Byte', 'Int16', 'Float32', etc.

subdatasets any subdataset DSNs is present, otherwise `NULL`

corners corner coordinates of the data, for non-zero skew geotransforms a 2-column matrix with rows upperLeft, lowerLeft, lowerRight, upperRight, and center

Note that the geotransform is a kind of obscure combination of the extent and dimension, I don't find it useful and modern GDAL is moving away from needing it so much. Extent is more sensible and used in many places in a straightforward way.

On access vapour functions will report on the existence of subdatasets while defaulting to the first subdataset found.

Value

list with vectors 'geotransform', 'dimXY', 'minmax', 'tilesXY', 'projection', 'bands', 'proj4', 'nodata_value', 'overviews', 'filelist' see sections in Details for more on each element

Subdatasets

Some sources provide multiple data sets, where a dataset is described by a 2- (or more) dimensional grid whose structure is described by the metadata described above. Note that *subdataset* is a different concept to *band or dimension*. Sources that may have multiple data sets are HDF4/HDF5 and NetCDF, and they are loosely analogous to the concept of *layer* in GDAL vector data. Variables are usually seen as distinct data but in GDAL and related 2D-interpretations this concept is leveraged as a 3rd dimension (and higher). In a GeoTIFF a third dimension might be implicit across bands, i.e. to express time varying data and so each band is not properly a variable. Similarly in NetCDF, the data may be any dimensional but there's only an implicit link for other variables that exist in that same dimensional space. When using GDAL you are always traversing this confusing realm.

If subdatasets are present but not specified the first is queried. The choice of subdataset is analogous to the way that the `raster` package behaves, and uses the argument `varname`. Variables in NetCDF correspond to subdatasets, but a single data set might have multiple variables in different bands or in dimensions, so this guide does not hold across various systems.

The Geo Transform

From https://gdal.org/en/stable/user/raster_data_model.html.

The affine transform consists of six coefficients returned by `GDALDataset::GetGeoTransform()` which map pixel/line coordinates into georeferenced space using the following relationship:

`Xgeo = GT(0) + Xpixel*GT(1) + Yline*GT(2)`

`Ygeo = GT(3) + Xpixel*GT(4) + Yline*GT(5)`

They are

GT0, xmin the x position of the lower left corner of the lower left pixel

GT1, xres the scale of the x-axis, the width of the pixel in x-units

GT2, yskew y component of the pixel width

GT3, ymax the y position of the upper left corner of the upper left pixel

GT4, xskew x component of the pixel height

GT5, yres the scale of the y-axis, the height of the pixel in *negative* y-units

Please note that these coefficients are equivalent to the contents of a *world file* but that the order is not the same and the world file uses cell centre convention rather than edge. https://en.wikipedia.org/wiki/World_file

Usually the skew components are zero, and so only four coefficients are relevant and correspond to the offset and scale used to position the raster - in combination with the number of rows and columns of data they provide the spatial extent and the pixel size in each direction. Very rarely a an actual affine raster will be use with this *rotation* specified within the transform coefficients.

Calculation of 'minmax' can take a significant amount of time, so it's not done by default. Use 'minmax = TRUE' to do it. (It does perform well, but may be prohibitive for very large or remote sources.)

Overviews

If there are no overviews this element will simply be a single-element vector of value 0. If there are overviews, the first value will give the number of overviews and their dimensions will be listed as pairs of x,y values.

See Also

vapour_sds_info

Examples

```
f <- system.file("extdata", "sst.tif", package = "vapour")
vapour_raster_info(f)
```

vapour_read_fids *Read feature names*

Description

Obtains the internal 'Feature ID (FID)' for a data source.

Usage

```
vapour_read_fids(
  dsource,
  layer = 0L,
  sql = "",
  limit_n = NULL,
  skip_n = 0,
  extent = NA
)

vapour_read_names(
  dsource,
  layer = 0L,
  sql = "",
  limit_n = NULL,
  skip_n = 0,
  extent = NA
)
```

Arguments

dsource	data source name (path to file, connection string, URL)
layer	integer of layer to work with, defaults to the first (0) or the name of the layer
sql	if not empty this is executed against the data source (layer will be ignored)
limit_n	an arbitrary limit to the number of features scanned
skip_n	an arbitrary number of features to skip
extent	apply an arbitrary extent, only when 'sql' used (must be 'ex = c(xmin, xmax, ymin, ymax)' but sp bbox, sf bbox, and raster extent also accepted)

Details

This may be virtual (created by GDAL for the SQL interface) and may be 0- or 1- based. Some drivers have actual names, and they are persistent and arbitrary. Please use with caution, this function can return the current FIDs, but there's no guarantee of what it represents for subsequent access.

An earlier version use 'OGRSQL' to obtain these names, which was slow for some drivers and also clashed with independent use of the sql argument. vapour_read_names() is an older name, aliased to vapour_read_fids().

Value

character vector of geometry id 'names'

Examples

```
file <- "list_locality_postcode_meander_valley.tab"
mvfile <- system.file(file.path("extdata/tab", file), package="vapour")
range(fids <- vapour_read_names(mvfile))
length(fids)
```

---

vapour_read_fields *Read feature field data*

Description

Read features fields (attributes), optionally after SQL execution.

Usage

```
vapour_read_fields(
  dsource,
  layer = 0L,
  sql = "",
  limit_n = NULL,
  skip_n = 0,
  extent = NA
)

vapour_read_attributes(
  dsource,
  layer = 0L,
  sql = "",
  limit_n = NULL,
  skip_n = 0,
  extent = NA
)
```

Arguments

dsource	data source name (path to file, connection string, URL)
layer	integer of layer to work with, defaults to the first (0) or the name of the layer
sql	if not empty this is executed against the data source (layer will be ignored)
limit_n	an arbitrary limit to the number of features scanned
skip_n	an arbitrary number of features to skip
extent	apply an arbitrary extent, only when 'sql' used (must be 'ex = c(xmin, xmax, ymin, ymax)' but sp bbox, sf bbox, and raster extent also accepted)

Details

Internal types are not fully supported, there are straightforward conversions for numeric, integer (32-bit) and string types. Date, Time, DateTime are returned as character, and Integer64 is returned as numeric.

Value

list of vectors one for each field in the source, each will be the same length which will depend on the values of 'skip_n', 'limit_n', 'sql', and the available records in the source. The types will be raw, numeric, integer, character, logical depending on the available mapping to the types in the source for the data there to R's native vectors.

Examples

```
file <- "list_locality_postcode_meander_valley.tab"
mvfile <- system.file(file.path("extdata/tab", file), package="vapour")
att <- vapour_read_fields(mvfile)
str(att)
sq <- "SELECT * FROM list_locality_postcode_meander_valley WHERE FID < 5"
(att <- vapour_read_fields(mvfile, sql = sq))
pfile <- "list_locality_postcode_meander_valley.tab"
dsource <- system.file(file.path("extdata/tab", pfile), package="vapour")
SQL <- "SELECT NAME FROM list_locality_postcode_meander_valley WHERE POSTCODE < 7300"
vapour_read_fields(dsource, sql = SQL)
```

---

vapour_read_geometry *Read GDAL feature geometry*

Description

Read GDAL geometry as binary blob, text, or numeric extent.

Usage

```
vapour_read_geometry_ia(dsource, layer = 0L, sql = "", extent = NA, ia = NULL)

vapour_read_geometry_ij(dsource, layer = 0L, sql = "", extent = NA, ij = NULL)

vapour_read_geometry(
  dsource,
  layer = 0L,
  sql = "",
  limit_n = NULL,
  skip_n = 0,
  extent = NA
)

vapour_read_geometry_text(
```

```
  dsource,
  layer = 0L,
  sql = "",
  textformat = "json",
  limit_n = NULL,
  skip_n = 0,
  extent = NA
)

vapour_read_extent(
  dsource,
  layer = 0L,
  sql = "",
  limit_n = NULL,
  skip_n = 0,
  extent = NA
)

vapour_read_type(
  dsource,
  layer = 0L,
  sql = "",
  limit_n = NULL,
  skip_n = 0,
  extent = NA
)
```

Arguments

dsource	data source name (path to file, connection string, URL)
layer	integer of layer to work with, defaults to the first (0) or the name of the layer
sql	if not empty this is executed against the data source (layer will be ignored)
extent	apply an arbitrary extent, only when 'sql' used (must be 'ex = c(xmin, xmax, ymin, ymax)' but sp bbox, sf bbox, and raster extent also accepted)
ia	an arbitrary index, integer vector with values between 0 and one less the number of features, duplicates allowed and arbitrary order is ok
ij	an range index, integer vector of length two with values between 0 and one less the number of features, this range of geometries is returned
limit_n	an arbitrary limit to the number of features scanned
skip_n	an arbitrary number of features to skip
textformat	indicate text output format, available are "json" (default), "gml", "kml", "wkt"

Details

`vapour_read_geometry` will read features as binary WKB, `vapour_read_geometry_text` as various text formats (geo-json, wkt, kml, gml),

`vapour_read_extent` a numeric extent which is the native bounding box, the four numbers (in this order) `xmin, xmax, ymin, ymax`. For each function an optional SQL string will be evaluated against the data source before reading.

`vapour_read_geometry_ia` will read features by *arbitrary index*, so any integer between 0 and one less than the number of features. These may be duplicated. If 'ia' is greater than the highest index NULL is returned, but if less than 0 the function will error.

`vapour_read_geometry_ij` will read features by *index range*, so two numbers to read ever feature between those limits inclusively. 'i' and 'j' must be increasing.

`vapour_read_type` will read the (wkb) type of the geometry as an integer. These are `0` unknown, `1` Point, `2` LineString, `3` Polygon, `4` MultiPoint, `5` MultiLineString, `6` MultiPolygon, `7` GeometryCollection, and the other more exotic types listed in "api/vector_c_api.html" from the GDAL home page (as at October 2020). A missing value 'NA' indicates an empty geometry.

Note that `limit_n` and `skip_n` interact with the affect of `sql`, first the query is executed on the data source, then while looping through available features `skip_n` features are ignored, and then a feature-count begins and the loop is stopped if `limit_n` is reached.

Note that `extent` applies to the 'SpatialFilter' of 'ExecuteSQL': https://gdal.org/user/ogr_sql_dialect.html#executesql.

Value

for `vapour_read_geometry()`, `vapour_read_geometry_ia()` and `vapour_read_geometry_ij()` a raw vector of geometry, for `vapour_read_extent()` a list of numeric vectors each with 'xmin,xmax,ymin,ymax' respectively for each geometry, for `vapour_read_type()` a character vector. See Details for more information.

Examples

```
file <- "list_locality_postcode_meander_valley.tab"
## A MapInfo TAB file with polygons
mvfile <- system.file(file.path("extdata/tab", file), package="vapour")
## A shapefile with points
pfile <- system.file("extdata/point.shp", package = "vapour")

## raw binary WKB points in a list
ptgeom <- vapour_read_geometry(pfile)
## create a filter query to ensure data read is small
SQL <- "SELECT FID FROM list_locality_postcode_meander_valley WHERE FID < 3"
## polygons in raw binary (WKB)
plgeom <- vapour_read_geometry_text(mvfile, sql = SQL)
## polygons in raw text (GeoJSON)
txtjson <- vapour_read_geometry_text(mvfile, sql = SQL)

## polygon extents in a list xmin, xmax, ymin, ymax
exgeom <- vapour_read_extent(mvfile)

## points in raw text (GeoJSON)
txtpointjson <- vapour_read_geometry_text(pfile)
## points in raw text (WKT)
txtpointwkt <- vapour_read_geometry_text(pfile, textformat = "wkt")
```

vapour_read_raster *Raster IO (read)*

Description

Read a window of data from a GDAL raster source. The first argument is the source name and the second is a 6-element window of offset, source dimension, and output dimension.

Usage

```
vapour_read_raster(
  x,
  band = 1,
  window,
  resample = "nearestneighbour",
  ...,
  sds = NULL,
  native = FALSE,
  set_na = TRUE,
  band_output_type = "",
  unscale = TRUE,
  nara = FALSE
)
```

Arguments

x	data source
band	index of which band to read (1-based)
window	src_offset, src_dim, out_dim
resample	resampling method used (see details)
...	reserved
sds	index of subdataset to read (usually 1)
native	apply the full native window for read, FALSE by default
set_na	specify whether NA values should be set for the NODATA
band_output_type	numeric type of band to apply (else the native type if ''), is mapped to one of 'Byte', 'Int32', or 'Float64'
unscale	default is TRUE so native values will be converted by offset and scale to floating point
nara	logical whether to return a (scaled) nativeRaster

Details

The value of window may be input as only 4 elements, in which case the source dimension Will be used as the output dimension.

This is analogous to the rgdal function readGDAL with its arguments offset, region.dim and output.dim. There's no semantic wrapper for this in vapour, but see https://github.com/hypertidy/lazyraster for one approach.

Resampling options will depend on GDAL version, but currently 'NearestNeighbour' (default), 'Average', 'Bilinear', 'Cubic', 'CubicSpline', 'Gauss', 'Lanczos', 'Mode' are potentially available. These are compared internally by converting to lower-case. Detailed use of this is barely tried or tested with vapour, but is a standard facility used in GDAL. Easiest way to compare results is with gdal_translate.

There is no write support in vapour.

Currently the window argument is required. If this argument unspecified and native = TRUE then the default window specification will be used, the entire extent at native resolution. If 'window' is specified and native = TRUE then the window is used as-is, with a warning (native is ignored).

'band_output_type' can be 'raw', 'integer', 'double', or case-insensitive versions of the GDAL types 'Byte', 'UInt16', 'Int16', 'UInt32', 'Int32', 'Float32', or 'Float64'. These are mapped to one of the supported types 'Byte' ('== raw'), 'Int32' ('== integer'), or 'Float64' ('== double').

Value

list of numeric vectors (only one for 'band')

Examples

```
f <- system.file("extdata", "sst.tif", package = "vapour")
## a 5*5 window from a 10*10 region
vapour_read_raster(f, window = c(0, 0, 10, 10, 5, 5))
vapour_read_raster(f, window = c(0, 0, 10, 10, 5, 5), resample = "Lanczos")
## find the information first
ri <- vapour_raster_info(f)
str(matrix(vapour_read_raster(f, window = c(0, 0, ri$dimXY, ri$dimXY)), ri$dimXY[1]))
## the method can be used to up-sample as well
str(matrix(vapour_read_raster(f, window = c(0, 0, 10, 10, 15, 25)), 15))
```

---

vapour_read_raster_block

Read or write raster block

Description

Read a 'block' from raster.

Usage

```
vapour_read_raster_block(
  dsource,
  offset,
  dimension,
  band = 1L,
  band_output_type = "",
  unscale = TRUE,
  nara = FALSE
)
```

Arguments

dsource	file name to read from, or write to
offset	position x,y to start writing (0-based, y-top)
dimension	window size to read from, or write to
band	which band to read (1-based)
band_output_type	numeric type of band to apply (else the native type if ") can be one of 'Byte', 'Int32', or 'Float64'
unscale	default is TRUE so native values will be converted by offset and scale to floating point
nara	if 'TRUE' return in nativeRaster format

Value

a list with a vector of data from the band read

Examples

```
f <- system.file("extdata", "sst.tif", package = "vapour")
v <- vapour_read_raster_block(f, c(0L, 0L), dimension = c(2L, 3L), band = 1L)
```

---

vapour_read_raster_raw

type safe(r) raster read

Description

These wrappers around vapour_read_raster() guarantee single vector output of the nominated type.

Usage

```
vapour_read_raster_raw(
  x,
  band = 1,
  window,
  resample = "nearestneighbour",
  ...,
  sds = NULL,
  native = FALSE,
  set_na = TRUE,
  nara = FALSE
)

vapour_read_raster_int(
  x,
  band = 1,
  window,
  resample = "nearestneighbour",
  ...,
  sds = NULL,
  native = FALSE,
  set_na = TRUE
)

vapour_read_raster_dbl(
  x,
  band = 1,
  window,
  resample = "nearestneighbour",
  ...,
  sds = NULL,
  native = FALSE,
  set_na = TRUE
)

vapour_read_raster_chr(
  x,
  band = 1,
  window,
  resample = "nearestneighbour",
  ...,
  sds = NULL,
  native = FALSE,
  set_na = TRUE
)

vapour_read_raster_hex(
  x,
```

```
  band = 1,
  window,
  resample = "nearestneighbour",
  ...,
  sds = NULL,
  native = FALSE,
  set_na = TRUE
)
```

Arguments

x	data source
band	index of which band to read (1-based)
window	src_offset, src_dim, out_dim
resample	resampling method used (see details)
...	reserved
sds	index of subdataset to read (usually 1)
native	apply the full native window for read, FALSE by default
set_na	specify whether NA values should be set for the NODATA
nara	logical whether to return a (scaled) nativeRaster

Details

*_hex and *_chr are aliases of each other.

Value

atomic vector of the nominated type raw, int, dbl, or character (hex)

Examples

```
f <- system.file("extdata", "sst.tif", package = "vapour")
vapour_read_raster_int(f, window = c(0, 0, 5, 4))
vapour_read_raster_raw(f, window = c(0, 0, 5, 4))
vapour_read_raster_chr(f, window = c(0, 0, 5, 4))
plot(vapour_read_raster_dbl(f, native = TRUE), pch = ".", ylim = c(273, 300))
```

---

vapour_report_fields *Read feature field types.*

Description

Obtains the internal type-constant name for the data attributes in a source.

Usage

```
vapour_report_fields(dsource, layer = 0L, sql = "")

vapour_report_attributes(dsource, layer = 0L, sql = "")
```

Arguments

dsource	data source name (path to file, connection string, URL)
layer	integer of layer to work with, defaults to the first (0) or the name of the layer
sql	if not empty this is executed against the data source (layer will be ignored)

Details

Use this to compare the interpreted versions converted into R types by vapour_read_fields.

This and vapour_read_fields() are aliased to older versions named 'vapour_report_attributes()' and 'vapour_read_attributes()', but "field" is a clearer and more sensible name (in our opinion).

These are defined for the enum OGRFieldType in GDAL itself. https://gdal.org/en/stable/doxygen/ogr__core_8h.html

Value

named character vector of the GDAL types for each field

Examples

```
file <- "list_locality_postcode_meander_valley.tab"
mvfile <- system.file(file.path("extdata/tab", file), package="vapour")
vapour_report_fields(mvfile)

## modified by sql argument
vapour_report_fields(mvfile,
  sql = "SELECT POSTCODE, NAME FROM list_locality_postcode_meander_valley")
```

---

vapour_sds_names *GDAL raster subdatasets (variables)*

Description

A **subdataset** is a collection abstraction for a number of **variables** within a single GDAL source. If there's only one variable the datasource and the variable have the same data source string. If there is more than one the subdatasets have the form **DRIVER:"datasourcename":varname**. Each subdataset name can stand in place of a data source name that has only one variable, so we always treat a source as a subdataset, even if there's only one.

Usage

```
vapour_sds_names(x)
```

Arguments

x	a data source string, filename, database connection string, or other URL

Details

Returns a character vector of 'subdatasets. In the case of a normal data source, with no subdatasets the value is source'.

If the raw SDS names contain spaces these are replaced by '%20' escape strings. A specific example is "WCS:https://elevation.nationalmap.gov:443" with request "arcgis/services/3DEPElevation/ImageServer/WCSServer?vers Gray". This function will return "..DEP3Elevation_Hillshade%20Gray". See wiki post for more details.

Value

character vector of subdataset names, or just the source itself if no SDS are present

Examples

```
f <- system.file("extdata/gdal", "sds.nc", package = "vapour")
## protect from error with netcdf problems
result <- try(vapour_sds_names(f), silent = TRUE)
if (!inherits(result, "try-error")) {
  print(result)
}
vapour_sds_names(system.file("extdata", "sst.tif", package = "vapour"))
```

---

vapour_set_config *Set and query GDAL configuration options*

Description

These functions can get and set configuration options for GDAL, for fine control over specific GDAL behaviours.

Usage

```
vapour_set_config(option, value)

vapour_get_config(option)
```

Arguments

option	GDAL config name (see Details), character string
value	value for config option, character string

Details

Configuration options may also be set as environment variables.

See GDAL config options for details on available options.

Value

character string for `vapour_get_config`, integer 1 for successful `vapour_set_config()`

Examples

```
## Not run:
(orig <- vapour_get_config("GDAL_CACHEMAX"))
vapour_set_config("GDAL_CACHEMAX", "64")
vapour_get_config("GDAL_CACHEMAX")
vapour_set_config("GDAL_CACHEMAX", orig)

## End(Not run)
```

`vapour_srs_wkt` *PROJ4 string to WKT*

Description

Convert a projstring to Well Known Text.

Usage

```
vapour_srs_wkt(crs)
```

Arguments

`crs`	projection string, see Details.

Details

The function is vectorized because why not, but probably only ever will be used on single element vectors of character strings.

Note that no sanitizing is done on inputs, we literally just 'OGRSpatialReference.SetFromUserInput(crs)' and give the output as WKT. If it's an error in GDAL it's an error in R.

Common inputs are WKT variants, 'AUTH:CODE's e.g. 'EPSG:3031', the 'OGC:CRS84' for long,lat WGS84, 'ESRI:code' and other authority variants, and datum names such as 'WGS84','NAD27' recognized by PROJ itself.

See help for 'SetFromUserInput' in 'OGRSpatialReference', and 'proj_create_crs_to_crs'.

c.proj_create_crs_to_crs

c.proj_create

SetFromUserInput

Value

WKT2 projection string

Examples

```
vapour_srs_wkt("+proj=laea +datum=WGS84")
```

---

vapour_vrt *Virtual raster*

Description

Simple VRT creation of a GDAL virtual raster. The data source string is **augmented** by input of other optional arguments. That means it **overrides** their values provided by the source data, or stands in place of this information if it is missing.

Usage

```
vapour_vrt(
  x,
  extent = NULL,
  projection = NULL,
  sds = 1L,
  bands = NULL,
  geolocation = NULL,
  ...,
  relative_to_vrt = FALSE,
  nomd = FALSE,
  overview = -1L,
  options = character()
)
```

Arguments

x	data source name, filepath, url, database connection string, or VRT text
extent	(optional) numeric extent, xmin,xmax,ymin,ymax
projection	(optional) character string, projection string ("auth:code", proj4, or WKT, or anything understood by PROJ, see Details)
sds	which subdataset to select from a source with more than one
bands	(optional) which band/s to include from the source
geolocation	vector of 2 dsn to longitude, latitude geolocation array sources
...	ignored
relative_to_vrt	default FALSE, if TRUE input strings that identify as files on the system are left as-is (by default they are made absolute at the R level)

nomd	if TRUE the Metadata tag is removed from the resulting VRT (it can be quite substantial)
overview	pick an integer overview from the source (0L is highest resolution, default -1L does nothing)
options	pass in options to the VRT creation, like 'c("-expand", "rgb", "-ot", "Byte"('

Details

Create a GDAL data source string (to be used like a filename) with various helpers. VRT stands for 'ViRTual'. A VRT string then acts as a representative of a data source for further use (to read or warp it).

An input string will be converted to a single subdataset, use 'sds' argument to select.

If 'extent', 'projection' is provided this is applied to override the source's extent and/or projection. (These might be invalid, or missing, so we facilitate correcting this).

If 'bands' is provided this is used to select a set of bands (numbered from 1), which might be repeated, or in any order and contain repetitions.

`vapour_vrt()` is vectorized, it will return multiple VRT strings for multiple inputs in a "length > 1" character vector. These are all independent, this is different to the function `vapour_warp_raster()` where multiple inputs are merged (possibly by sequential overlapping).

If `geolocation` is set the 'GeoTransform' element is forcibly removed from the vrt output, in order to avoid https://github.com/hypertidy/vapour/issues/210 (there might be a better fix).

Value

VRT character string (for use by GDAL-capable tools, i.e. reading raster)

Rationale

For a raster, the basic essentials we can specify or modify for a source are

1. the source, 2) the extent, 3) the projection 4) what subdataset (these are variables from NetCDF and the like that contain multiple datasets) and 5) which band/s to provided. For extent and projection we are simply providing or correcting complete information about how to interpret the georeferencing, with subdatasets and bands this is more like a query of which ones we want. If we only wanted band 5, then the output data would have one band only (and we we read it we need `band = 1`).

We don't provide ability override the dimension, but that is possible as well. More features may come with a 'VRTBuilder' interface.

Projections

Common inputs for `projection` are WKT variants, "AUTH:CODE"s e.g. "EPSG:3031", the "OGC:CRS84" for long,lat WGS84, "ESRI:code" and other authority variants, and datum names such as 'WGS84','NAD27' recognized by PROJ itself.

See the following links to GDAL and PROJ documentation:

PROJ documentation: c.proj_create_crs_to_crs

PROJ documentation: c.proj_create

GDAL documentation: SetFromUserInput

Examples

```
tif <- system.file("extdata", "sst.tif", package = "vapour")
vapour_vrt(tif)

vapour_vrt(tif, bands = c(1, 1))
```

---

vapour_vsi_list *Read GDAL virtual source contents*

Description

Obtain the names of available items in a virtual file source.

Usage

```
vapour_vsi_list(dsource, ...)
```

Arguments

dsource	data source name (path to file, connection string, URL) with virtual prefix, see Details
...	ignored

Details

The dsource must begin with a valid form of the special vsiPREFIX, for details see GDAL Virtual File Systems.

Note that the listing is not recursive, and so cannot be used for automation. One would use this function interactively to determine a useable /vsiPREFIX/dsource data source string.

Value

character vector listing of items

Examples

```
pointzipfile <- system.file("extdata/vsi/point_shp.zip", package = "vapour")
vapour_vsi_list(sprintf("/vsizip/%s", pointzipfile))

## Not run:
## example from https://github.com/hypertidy/vapour/issues/55
#file <- "http/radmap_v3_2015_filtered_dose/radmap_v3_2015_filtered_dose.ers.zip"
#url <- "http://dapds00.nci.org.au/thredds/fileServer/rr2/national_geophysical_compilations"
```

```
#u <- sprintf("/vsizip//vsicurl/%s", file.path(url, file))
#vapour_vsi_list(u)
#[1] "radmap_v3_2015_filtered_dose"     "radmap_v3_2015_filtered_dose.ers"
#[3] "radmap_v3_2015_filtered_dose.isi" "radmap_v3_2015_filtered_dose.txt"
#gdalinfo /vsitar//home/ubuntu/LT05_L1GS_027026_20060116_20160911_01_T2.tar.gz
#vapour_vsi_list("/vsitar//home/ubuntu/LT05_L1GS_027026_20060116_20160911_01_T2.tar.gz")
#"LT05_L1TP_027026_20061218_20160911_01_T1_ANG.txt"
#"LT05_L1TP_027026_20061218_20160911_01_T1_B1.TIF"
#"LT05_L1TP_027026_20061218_20160911_01_T1_B2.TIF"
#"LT05_L1TP_027026_20061218_20160911_01_T1_B3.TIF"
#...

## End(Not run)
```

---

vapour_warp_raster *Raster warper (reprojection)*

Description

Read a window of data from a GDAL raster source through a warp specification. The warp specification is provided by 'extent', 'dimension', and 'projection' properties of the transformed output.

Usage

```
vapour_warp_raster(
  x,
  bands = NULL,
  extent = NULL,
  dimension = NULL,
  projection = "",
  set_na = TRUE,
  source_projection = NULL,
  source_extent = 0,
  resample = "near",
  silent = TRUE,
  ...,
  band_output_type = "",
  warp_options = "",
  transformation_options = "",
  open_options = "",
  options = "",
  nomd = FALSE,
  overview = -1L,
  nara = FALSE
)
```

Arguments

x	vector of data source names (file name or URL or database connection string)
bands	index of band/s to read (1-based), may be new order or replicated, or NULL (all bands used, the default)
extent	extent of the target warped raster 'c(xmin, xmax, ymin, ymax)'
dimension	dimensions in pixels of the warped raster (x, y)
projection	projection of warped raster (in Well-Known-Text, or any projection string accepted by GDAL)
set_na	NOT IMPLEMENTED logical, should 'NODATA' values be set to NA
source_projection	optional, override or augment the projection of the source (in Well-Known-Text, or any projection string accepted by GDAL)
source_extent	extent of the source raster, used to override/augment incorrect source metadata
resample	resampling method used (see details in vapour_read_raster)
silent	TRUE by default, set to FALSE to report messages
...	unused
band_output_type	numeric type of band to apply (else the native type if '') can be one of 'Byte', 'Int32', or 'Float64' but see details in vapour_read_raster()
warp_options	character vector of options, as in gdalwarp -wo - see Details
transformation_options	character vector of options, as in gdalwarp -to see Details
open_options	character vector of options, as in gdalwarp -oo - see Details
options	character vectors of options as per the gdalwarp command line
nomd	if TRUE the Metadata tag is removed from the resulting VRT (it can be quite substantial)
overview	pick an integer overview from the source (0L is highest resolution, default -1L does nothing)
nara	if 'TRUE' return in nativeRaster format

Details

Any bands may be read, including repeats.

This function is not memory safe, the source is left on disk but the output raster is all computed in memory so please be careful with very large values for 'dimension'. `1000 * 1000 * 8` for 1000 columns, 1000 rows and floating point double type will be 8Mb.

There's control over the output type, and is auto-detected from the source (raw/Byte, integer/Int32, numeric/Float64) or can be set with 'band_output_type'.

'projection' refers to any projection string for a CRS understood by GDAL. This includes the full Well-Known-Text specification of a coordinate reference system, PROJ strings, "AUTH:CODE" types, and others. See vapour_srs_wkt() for conversion from PROJ.4 string to WKT, and vapour_raster_info() and vapour_layer_info() for various formats available from a data source. Any string accepted

by GDAL may be used for 'projection' or 'source_projection', including EPSG strings, PROJ4 strings, and file names. Note that this argument was named 'wkt' up until version 0.8.0.

'extent' is the four-figure xmin,xmax,ymin,ymax outer corners of corner pixels

'dimension' is the pixel dimensions of the output, x (ncol) then y (nrow).

Options for missing data are not yet handled, just returned as-is. Note that there may be regions of "zero data" in a warped output, separate from propagated missing "NODATA" values in the source.

Argument 'source_projection' may be used to assign the projection of the source, 'source_extent' to assign the extent of the source. Sometimes both are required. Note, this is now better done by creating 'VRT', see `vapour_vrt()` for assigning the source projection, extent, and some other options.

If multiple sources are specified via 'x' and either 'source_projection' or 'source_extent' are provided, these are applied to every source even if they have valid values already. If this is not sensible please use VRT to wrap the multiple sources first.

Wild combinations of 'source_extent' and/or 'extent' may be used for arbitrary flip orientations, scale and offset. For expert usage only. Old versions allowed transform input for target and source but this is now disabled (maybe we'll write a new wrapper for that).

Value

list of vectors (only 1 for 'band') of numeric values, in raster order

Options

The various options are convenience arguments for 'warp options -wo', transformation options -to', 'open options -oo', and 'options' for any other arguments in gdalwarp. There are no 'creation options -co' or 'dataset output options -doo', because these are not supported by the MEM driver.

All 'warp_options' are paired with a '-wo' declaration and similarly for '-to', and '-oo', this is purely a convenience, since 'options' itself can be used for these as well but we recommend using the individual arguments. An example for warp options is `warp_options = c("SAMPLE_GRID=YES", "SAMPLE_STEPS=30")` and one for general arguments might be 'options = c("-ovr", "AUTO", "-nomd", "-cutline", "/path/to/cut.gpkg", "-crop_to_cutline")'. If they would be separated by spaces on the command line then include as separate elements in the options character vector.

See GDALWarpOptions for '-wo'.

See GDAL transformation options for '-to'.

See GDALWARP command line app for further details.

Note we already apply the following gdalwarp arguments based on input R arguments to this function.

-of MEM is hardcoded, but may be extended in future

-t_srs set via 'projection'

-s_srs set via 'source_projection'

-te set via 'extent'

-ts set via 'dimension'

-r set via 'resample'

-ot set via 'band_output_type'

-te_srs not supported

-a_ullr (not a gdalwarp argument, but we do analog) set via 'source_extent' use vapour_vrt() instead

In future all 'source_*' arguments may be deprecated in favour of augmentation by 'vapour_vrt()'.

Common inputs for `projection` are WKT variants, 'AUTH:CODE's e.g. 'EPSG:3031', the 'OGC:CRS84' for lon,lat WGS84, 'ESRI:code' and other authority variants, and datum names such as 'WGS84','NAD27' recognized by PROJ itself.

See help for 'SetFromUserInput' in 'OGRSpatialReference', and 'proj_create_crs_to_crs'.

c.proj_create_crs_to_crs

c.proj_create

SetFromUserInput

See Also

vapour_read_raster vapour_read_raster_raw vapour_read_raster_int vapour_read_raster_dbl vapour_read_raster_chr vapour_read_raster_hex

Examples

```
b <- 4e5
f <- system.file("extdata", "sst.tif", package = "vapour")
prj <- "+proj=aeqd +lon_0=147 +lat_0=-42"
vals <- vapour_warp_raster(f, extent = c(-b, b, -b, b),
                           dimension = c(186, 298),
                           bands = 1,
                           projection = vapour_srs_wkt(prj),
                           warp_options = c("SAMPLE_GRID=YES"))


image(list(x = seq(-b, b, length.out = 187), y = seq(-b, b, length.out = 298),
    z = matrix(unlist(vals, use.names = FALSE), 186)[,298:1]), asp = 1)
```

---

vapour_warp_raster_raw

type safe(r) raster warp

Description

These wrappers around vapour_warp_raster() guarantee single vector output of the nominated type.

Usage

```
vapour_warp_raster_raw(
  x,
  bands = NULL,
  extent = NULL,
  dimension = NULL,
  projection = "",
  set_na = TRUE,
  source_projection = NULL,
  source_extent = 0,
  resample = "near",
  silent = TRUE,
  ...,
  warp_options = "",
  transformation_options = "",
  open_options = "",
  options = ""
)

vapour_warp_raster_int(
  x,
  bands = NULL,
  extent = NULL,
  dimension = NULL,
  projection = "",
  set_na = TRUE,
  source_projection = NULL,
  source_extent = 0,
  resample = "near",
  silent = TRUE,
  ...,
  warp_options = "",
  transformation_options = "",
  open_options = "",
  options = ""
)

vapour_warp_raster_dbl(
  x,
  bands = NULL,
  extent = NULL,
  dimension = NULL,
  projection = "",
  set_na = TRUE,
  source_projection = NULL,
  source_extent = 0,
  resample = "near",
  silent = TRUE,
```

```
  ...,
  warp_options = "",
  transformation_options = "",
  open_options = "",
  options = ""
)

vapour_warp_raster_chr(
  x,
  bands = NULL,
  extent = NULL,
  dimension = NULL,
  projection = "",
  set_na = TRUE,
  source_projection = NULL,
  source_extent = 0,
  resample = "near",
  silent = TRUE,
  ...,
  warp_options = "",
  transformation_options = "",
  open_options = "",
  options = ""
)

vapour_warp_raster_hex(
  x,
  bands = NULL,
  extent = NULL,
  dimension = NULL,
  projection = "",
  set_na = TRUE,
  source_projection = NULL,
  source_extent = 0,
  resample = "near",
  silent = TRUE,
  ...,
  warp_options = "",
  transformation_options = "",
  open_options = "",
  options = ""
)
```

Arguments

x	vector of data source names (file name or URL or database connection string)
bands	index of band/s to read (1-based), may be new order or replicated, or NULL (all bands used, the default)

extent	extent of the target warped raster 'c(xmin, xmax, ymin, ymax)'
dimension	dimensions in pixels of the warped raster (x, y)
projection	projection of warped raster (in Well-Known-Text, or any projection string accepted by GDAL)
set_na	NOT IMPLEMENTED logical, should 'NODATA' values be set to NA
source_projection	optional, override or augment the projection of the source (in Well-Known-Text, or any projection string accepted by GDAL)
source_extent	extent of the source raster, used to override/augment incorrect source metadata
resample	resampling method used (see details in vapour_read_raster)
silent	TRUE by default, set to FALSE to report messages
...	unused
warp_options	character vector of options, as in gdalwarp -wo - see Details
transformation_options	character vector of options, as in gdalwarp -to see Details
open_options	character vector of options, as in gdalwarp -oo - see Details
options	character vectors of options as per the gdalwarp command line

Details

_hex and _chr are aliases of each other.

Value

atomic vector of the nominated type raw, int, dbl, or character (hex)

Examples

```
b <- 4e5
f <- system.file("extdata", "sst.tif", package = "vapour")
prj <- "+proj=aeqd +lon_0=147 +lat_0=-42"
bytes <- vapour_warp_raster_raw(f, extent = c(-b, b, -b, b),
                            dimension = c(18, 2),
                            bands = 1,
                            projection = prj)
# not useful given source type floating point, but works
str(bytes)
```

vapour_write_raster_block *Write data to a block* in an existing file.

Description

Be careful! The write function doesn't create a file, you have to use an existing one. Don't write to a file you don't want to update by mistake.

Usage

```
vapour_write_raster_block(
  dsource,
  data,
  offset,
  dimension,
  band = 1L,
  overwrite = FALSE
)
```

Arguments

dsource	data source name
data	data vector, length should match prod(dimension) or length 1 allowed
offset	offset to start
dimension	dimension to write
band	which band to write to (1-based)
overwrite	set to FALSE as a safety valve to not overwrite an existing file

Value

a logical value indicating success (or failure) of the write

Examples

```
f <- system.file("extdata", "sst.tif", package = "vapour")
v <- vapour_read_raster_block(f, c(0L, 0L), dimension = c(2L, 3L), band = 1L)
file.copy(f, tf <- tempfile(fileext = ".tif"))
try(vapour_write_raster_block(tf, data = v[[1]], offset = c(0L, 0L),
               dimension = c(2L, 3L), band = 1L))
if (file.exists(tf)) file.remove(tf)
```

vector_vrt *Vector VRT*

Description

Just a simple text generator to generate the VRT for a vector layer, First layer is chosen if not otherwise specified.

Usage

```
vector_vrt(x, layer = 1L, projection = NULL, sql = NULL, a_srs = NULL)
```

Arguments

x	data source name
layer	layer index (1-based) or name
projection	crs of the output
sql	SQL for ExecuteSQL to define the layer
a_srs	set the source crs

Details

Using 'sql' overrides the 'layer', and using 'projection' results in the geometries being transformed. No check is made of the layer source projection.

Use 'a_srs' to ensure the source has a source crs (that might be the only thing you use this for, even if not reprojecting).

It's expected that if you use this with a source without a source projection, you'll get "Failed to import source SRS", so use argument "a_srs" to set it if needed (or many other GDAL other facilities that do this).

Value

single element character vector

Examples

```
file <- "list_locality_postcode_meander_valley.tab"
## A MapInfo TAB file with polygons
mvfile <- system.file(file.path("extdata/tab", file), package="vapour")
vector_vrt(mvfile, sql = "SELECT * FROM list_locality_postcode_meander_valley LIMIT 5 OFFSET 4")

## read this with vapour_read_geometry() and it will be projected to VicGrid
vector_vrt(mvfile, projection = "EPSG:3111")
```

Index