Package 'vapour'

August 25, 2021

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

Version 0.8.0

Description Provides low-level access to 'GDAL' functionality for R packages.
'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/>.

Depends R (>= 3.3.0)

License GPL-3

Encoding UTF-8

LazyData true

LinkingTo Rcpp

Imports Rcpp, utils

RoxygenNote 7.1.1

Suggests testthat, knitr, markdown, rmarkdown, spelling

SystemRequirements GDAL (>= 2.0.1), PROJ.4 (>= 4.8.0), C++11

VignetteBuilder knitr

URL https://github.com/hypertidy/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] (figured out the mechanism for the resampling algorithm),
Mark Padgham [ctb] (helped get started :)),
Edzer Pebesma [ctb] (wrote allocate_attribute, copied here from sf),
Roger Bivand [ctb] (wrote configure.ac, adapted here from rgdal),
Jim Hester [ctb] (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] (wrote some source in the warper, terra package used as example/inspiration)

Maintainer Michael Sumner <mdsumner@gmail.com>

Repository CRAN

Date/Publication 2021-08-25 12:20:02 UTC

R topics documented:

`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.

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 GeoPackage format in `system.file("extdata/sst_c.gpkg",package = "vapour")`

Examples

```
## library(sf)
## plot(sst_c)
f <- system.file("extdata/sst_c.gpkg", package = "vapour")

## create an equivalent but class-less form of sst_c  with GeoJSON rather than sf sfc format
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_gdal_version *GDAL version and drivers.*

Description

Return information about the GDAL library in use.

Usage

```
vapour_gdal_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.gpkg", package = "vapour")
vapour_driver(f)

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

---

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_info *Read GDAL layer info*

Description

Read GDAL layer information for a vector data source.

Usage

```
vapour_layer_info(
  dsource,
  layer = 0L,
  sql = "",
  ...,
  extent = TRUE,
  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)
...	unused, reserved for future use
extent	logical to control if extent calculated and returned, TRUE by default (set to FALSE to avoid the extra calculation and missing value is the result)
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)

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)
```

`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

dimXY dimensions x-y, columns*rows

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

tilesXY 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)

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/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_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.

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_cpp will read a feature for each of the ways listed above and is used by those functions. It's recommended to use the more specialist functions rather than this more general one.

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).

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.

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_names *Read feature names*

Description

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

Usage

```
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.

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_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
)
```

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

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).

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_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/doxygen/ogr__core_8h.html#a787194bea637faf12d61643124a7c9fc

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, Thredds or other URL

Details

Returns a list of `datasource` and `subdataset`. In the case of a normal data source, with no subdatasets the value of both entries is the `datasource`.

Value

list of character vectors, see Details

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_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.

You can get some funky outputs from random strings, so don't do that. Common sensible 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' in PROJ.

Value

WKT2 projection string

Examples

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

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", mustWork = TRUE)
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. Only a single band may be read. The warp specification is provided by 'extent', 'dimension', and 'wkt' properties of the transformed output.

Usage

```
vapour_warp_raster(
  x,
  bands = 1L,
  extent = NULL,
  dimension = NULL,
  wkt = "",
  set_na = TRUE,
  source_wkt = NULL,
  source_extent = 0,
  resample = "near",
  silent = TRUE,
  ...,
  source_geotransform = 0,
  geotransform = NULL
)
```

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)
extent	extent of the target warped raster 'c(xmin, xmax, ymin, ymax)'
dimension	dimensions in pixels of the warped raster (x, y)
wkt	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_wkt	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
source_geotransform	DEPRECATED use 'source_extent' (override the native geotransform of the source)
geotransform	DEPRECATED use 'extent' the affine geotransform of the warped raster

Details

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 no control over the output type (always double floating point). #' 'wkt' refers to the full Well-Known-Text specification of a coordinate reference system. See vapour_srs_wkt() for conversion from PROJ.4 string to WKT. Any string accepted by GDAL may be used for 'wkt' or 'source_wkt', including EPSG strings, PROJ4 strings, and file names.

'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).

Values 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_wkt' may be used to assign the projection of the source, 'source_extent' to assign the extent of the source. Sometimes both are required.

If multiple sources are specified via 'x' and either 'source_wkt' 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 (see the gdalio package for some in-dev ideas).

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

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),
                           wkt = vapour_srs_wkt(prj))
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)
```

Index