Package 'gdalraster'

August 29, 2023

Title Bindings to the 'Geospatial Data Abstraction Library' Raster API

Version 1.4.0

Description Interface to the raster API of the 'Geospatial Data Abstraction Library' ('GDAL') supporting manual creation of uninitialized datasets, creation from existing raster as template, low level I/O, configuration of virtual raster (VRT), coordinate transformation, and access to 'gdalwarp' for reprojection. Includes selected 'GDAL' algorithms and functions for working with spatial reference systems. Calling signatures resemble the native C, C++ and Python APIs provided by the 'GDAL' project (<https://gdal.org>). Bindings are implemented via 'Rcpp' exposed class along with several stand-alone functions. Additional functionality includes: class 'RunningStats' for efficient summary statistics on large data streams; class 'CmbTable' for counting unique combinations of integers with a hash table; raster overlay to identify and count unique pixel combinations across multiple input layers; raster calculation by evaluating any R expression on a stack of layers with pixel coordinates available as variables; and raster display using base 'graphics'. 'gdalraster' may be suitable for applications that primarily need low-level I/O or prefer a direct 'GDAL' API. The additional functionality is somewhat aimed at thematic data analysis but may have other utility.

License MIT + file LICENSE

Copyright See file inst/COPYRIGHTS for details.

URL https://usdaforestservice.github.io/gdalraster/, https://github.com/USDAForestService/gdalraster

BugReports https://github.com/USDAForestService/gdalraster/issues

Depends R (>= 4.2.0)

Imports graphics, grDevices, methods, Rcpp (>= 1.0.7), stats, tools, utils

LinkingTo Rcpp

Suggests knitr, rmarkdown, scales, testthat (>= 3.0.0), xml2

NeedsCompilation yes

SystemRequirements GDAL (>= 2.3.0, built against GEOS), PROJ (>= 4.8.0)

Encoding UTF-8

RoxygenNote 7.2.3

VignetteBuilder knitr

Config/testthat/edition 3

Author Chris Toney [aut, cre],
Frank Warmerdam [ctb, cph] (GDAL API/documentation; src/progress_r.cpp from /gdal/port/cpl_progress.cpp),
Even Rouault [ctb, cph] (GDAL API/documentation),
Marius Appel [ctb, cph] (configure.ac based on https://github.com/appelmar/gdalcubes),
Daniel James [ctb, cph] (Boost combine hashes method),
Peter Dimov [ctb, cph] (Boost combine hashes method)

Maintainer Chris Toney <chris.toney@usda.gov>

Repository CRAN

Date/Publication 2023-08-29 18:40:02 UTC

R topics documented:

`gdalraster-package` *Bindings to the GDAL Raster API*

Description

`gdalraster` is an interface to the Geospatial Data Abstraction Library (GDAL) for low level raster I/O. Calling signatures resemble those of the native C, C++ and Python APIs provided by the GDAL project. See `https://gdal.org/api/` for details of the GDAL Raster API.

Details

Core functionality is contained in the `GDALRaster` class and several GDAL related stand-alone functions:

- `GDALRaster` exposes a C++ class that allows opening a raster dataset and calling methods on the Dataset, Driver and RasterBand objects in the underlying API (get/set parameters, read/write pixel data).
- raster creation: `create`, `createCopy`, `rasterFromRaster`
- virtual raster: `rasterToVRT`
- reproject: `warp`
- algorithms: `dem_proc`, `fillNodata`, `GDALRaster$getChecksum`
- geotransform conversion: `inv_geotransform`, `get_pixel_line`
- coordinate transformation: `transform_xy`, `inv_project`
- spatial reference convenience functions: `epsg_to_wkt`, `srs_to_wkt`, `srs_is_geographic`, `srs_is_projected`, `srs_is_same`
- geometry convenience functions: `bbox_from_wkt`, `bbox_to_wkt`, `bbox_intersect`, `bbox_union`, `has_geos`
- GDAL configuration: `gdal_version`, `get_cache_used`, `get_config_option`, `set_config_option`

Additional functionality includes:

- class `RunningStats` calculates mean and variance in one pass. The min, max, sum, and count are also tracked (i.e., summary statistics on a data stream).

- class CmbTable implements a hash table for counting unique combinations of integer values.
- combine overlays multiple rasters so that a unique ID is assigned to each unique combination of input values. Pixel counts for each unique combination are obtained, and combination IDs are optionally written to an output raster.
- calc evaluates an R expression for each pixel in a raster layer or stack of layers. Individual pixel coordinates are available as variables in the R expression, as either x/y in the raster projected coordinate system or inverse projected longitude/latitude.
- plot_raster displays raster data using base R graphics. Supports single-band grayscale, RGB, color tables and color map functions (e.g., color ramp).

Note

Documentation for several wrapper functions borrows from the GDAL API documentation, (c) 1998-2023, Frank Warmerdam, Even Rouault, and others, MIT license.

Sample datasets are included with the package and used in examples throughout the documentation. The sample data include LANDFIRE raster layers describing terrain, vegetation and wildland fuels (LF 2020 version), and Landsat C2 Analysis Ready Data downloaded from USGS Earth Explorer.

system.file() is used in the examples to access the sample datasets. This enables the code to run regardless of where R is installed. Users will normally give file names as a regular full path or relative to the current working directory.

Author(s)

GDAL: https://github.com/OSGeo/gdal/graphs/contributors

R interface/additional functionality: Chris Toney

Maintainer: Chris Toney <chris.toney at usda.gov>

See Also

GDAL Raster Data Model:
https://gdal.org/user/raster_data_model.html

Raster format descriptions:
https://gdal.org/drivers/raster/index.html

Geotransform tutorial:
https://gdal.org/tutorials/geotransforms_tut.html

---

bbox_from_wkt *Get the bounding box of a geometry specified in OGC WKT format.*

Description

bbox_from_wkt() returns the bounding box of a WKT 2D geometry (e.g., LINE, POLYGON, MULTIPOLYGON).

Usage

```
bbox_from_wkt(wkt)
```

Arguments

wkt	Character. OGC WKT string for a simple feature 2D geometry.

Value

Numeric vector of length four containing the xmin, ymin, xmax, ymax of the geometry specified by wkt.

See Also

bbox_to_wkt()

Examples

```
bnd <- "POLYGON ((324467.3 5104814.2, 323909.4 5104365.4, 323794.2
5103455.8, 324970.7 5102885.8, 326420.0 5103595.3, 326389.6 5104747.5,
325298.1 5104929.4, 325298.1 5104929.4, 324467.3 5104814.2))"
bbox_from_wkt(bnd)
```

---

bbox_intersect *Bounding box intersection / union*

Description

bbox_intersect() returns the bounding box intersection, and bbox_union() returns the bounding box union, for input of either raster file names or list of bounding boxes. All of the inputs must be in the same projected coordinate system. These functions require GDAL built with the GEOS library.

Usage

```
bbox_intersect(x, as_wkt = FALSE)

bbox_union(x, as_wkt = FALSE)
```

Arguments

x	Either a character vector of raster file names, or a list with each element a bounding box numeric vector (xmin, ymin, xmax, ymax).
as_wkt	Logical. TRUE to return the bounding box as a polygon in OGC WKT format, or FALSE to return as a numeric vector.

Value

The intersection (`bbox_intersect()`) or union (`bbox_union()`) of inputs. If `as_wkt = FALSE`, a numeric vector of length four containing xmin, ymin, xmax, ymax. If `as_wkt = TRUE`, a character string containing OGC WKT for the bbox as POLYGON. `NA` is returned if GDAL was built without the GEOS library.

See Also

`bbox_from_wkt()`, `bbox_to_wkt()`

Examples

```
bbox_list <-list()

elev_file <- system.file("extdata/storml_elev.tif", package="gdalraster")
ds <- new(GDALRaster, elev_file, read_only=TRUE)
bbox_list[[1]] <- ds$bbox()
ds$close()

b5_file <- system.file("extdata/sr_b5_20200829.tif", package="gdalraster")
ds <- new(GDALRaster, b5_file, read_only=TRUE)
bbox_list[[2]] <- ds$bbox()
ds$close()

bnd <- "POLYGON ((324467.3 5104814.2, 323909.4 5104365.4, 323794.2
5103455.8, 324970.7 5102885.8, 326420.0 5103595.3, 326389.6 5104747.5,
325298.1 5104929.4, 325298.1 5104929.4, 324467.3 5104814.2))"
bbox_list[[3]] <- bbox_from_wkt(bnd)

print(bbox_list)
bbox_intersect(bbox_list)
bbox_union(bbox_list)
```

---

bbox_to_wkt *Convert a bounding box to POLYGON in OGC WKT format.*

Description

`bbox_to_wkt()` returns a WKT POLYGON string for the given bounding box. This function requires GDAL built with the GEOS library.

Usage

```
bbox_to_wkt(bbox)
```

Arguments

bbox	Numeric vector of length four containing xmin, ymin, xmax, ymax.

Value

Character string for an OGC WKT polygon. An empty string is returned if GDAL was built without the GEOS library.

See Also

bbox_from_wkt()

Examples

```
elev_file <- system.file("extdata/storml_elev.tif", package="gdalraster")
ds <- new(GDALRaster, elev_file, read_only=TRUE)
bbox_to_wkt(ds$bbox())
ds$close()
```

---

calc *Raster calculation*

Description

calc() evaluates an R expression for each pixel in a raster layer or stack of layers. Each layer is defined by a raster filename, band number, and a variable name to use in the R expression. If not specified, band defaults to 1 for each input raster. Variable names default to LETTERS if not specified (A (layer 1), B (layer 2), ...). All of the input layers must have the same extent and cell size. The projection will be read from the first raster in the list of inputs. Individual pixel coordinates are also available as variables in the R expression, as either x/y in the raster projected coordinate system or inverse projected longitude/latitude.

Usage

```
calc(
  expr,
  rasterfiles,
  bands = NULL,
  var.names = NULL,
  dstfile = tempfile("rastcalc", fileext = ".tif"),
  fmt = NULL,
  dtName = "Int16",
  out_band = NULL,
  options = NULL,
  nodata_value = NULL,
  setRasterNodataValue = FALSE,
  usePixelLonLat = FALSE,
  write_mode = "safe"
)
```

Arguments

expr	An R expression as a character string (e.g., "A + B").
rasterfiles	Character vector of source raster filenames.
bands	Integer vector of band numbers to use for each raster layer.
var.names	Character vector of variable names to use for each raster layer.
dstfile	Character filename of output raster.
fmt	Output raster format name (e.g., "GTiff" or "HFA"). Will attempt to guess from the output filename if not specified.
dtName	Character name of output data type (e.g., Byte, Int16, UInt16, Int32, UInt32, Float32).
out_band	Integer band number in dstfile for writing output.
options	Optional list of format-specific creation options in a vector of "NAME=VALUE" pairs (e.g., options = c("COMPRESS=LZW") to set LZW compression during creation of a GTiff file).
nodata_value	Numeric value to assign if expr returns NA.
setRasterNodataValue	Logical. TRUE will attempt to set the raster format nodata value to nodata_value, or FALSE not to set a raster nodata value.
usePixelLonLat	Logical. If TRUE, pixelX and pixelY will be inverse projected to geographic coordinates and available as pixelLon and pixelLat in expr (adds computation time).
write_mode	Character. Name of the file write mode for output. One of: • safe - execution stops if dstfile already exists (no output written) • overwrite - if dstfile exists if will be overwritten with a new file • update - if dstfile exists, will attempt to open in update mode and write output to out_band

Details

The variables in expr are vectors of length raster Xsize (rows of the raster layers). The expression should return a vector also of length raster Xsize (an output row). Two special variable names are available in expr by default: pixelX and pixelY provide the pixel center coordinate in projection units. If usePixelLonLat = TRUE, the pixel x/y coordinates will also be inverse projected to longitude/latitude and available in expr as pixelLon and pixelLat (in the same geographic coordinate system used by the input projection, which is read from the first input raster).

To refer to specific bands in a multi-band file, repeat the filename in rasterfiles and specify corresponding band numbers in bands, along with optional variable names in var.names, for example,

```
rasterfiles = c("multiband.tif", "multiband.tif")
bands = c(4, 5)
var.names = c("B4", "B5")
```

Output will be written to dstfile. To update a file that already exists, set write_mode = "update" and set out_band to an existing band number in dstfile (new bands cannot be created in dstfile).

Value

Returns the output filename invisibly.

See Also

GDALRaster-class, combine(), rasterToVRT()

Examples

```
### Using pixel longitude/latitude

## Hopkins bioclimatic index (HI) as described in:
## Bechtold, 2004, West. J. Appl. For. 19(4):245-251.
## Integrates elevation, latitude and longitude into an index of the
## phenological occurrence of springtime. Here it is relativized to
## mean values for an eight-state region in the western US.
## Positive HI means spring is delayed by that number of days relative
## to the reference position, while negative values indicate spring is
## advanced. The original equation had elevation units as feet, so
## converting m to ft in `expr`.

elev_file <- system.file("extdata/storml_elev.tif", package="gdalraster")

## expression to calculate HI
expr <- "round( ((ELEV_M * 3.281 - 5449) / 100) +
                ((pixelLat - 42.16) * 4) +
                ((-116.39 - pixelLon) * 1.25) )"

## calc() writes to a tempfile by default
hi_file <- calc(expr = expr,
                rasterfiles = elev_file,
                var.names = "ELEV_M",
                dtName = "Int16",
                nodata_value = -32767,
                setRasterNodataValue = TRUE,
                usePixelLonLat = TRUE)

ds <- new(GDALRaster, hi_file, read_only=TRUE)
## min, max, mean, sd
ds$getStatistics(band=1, approx_ok=FALSE, force=TRUE)
ds$close()


### Calculate normalized difference vegetation index (NDVI)

## Landast band 4 (red) and band 5 (near infrared):
b4_file <- system.file("extdata/sr_b4_20200829.tif", package="gdalraster")
b5_file <- system.file("extdata/sr_b5_20200829.tif", package="gdalraster")

## is nodata value set
ds <- new(GDALRaster, b4_file, read_only=TRUE)
ds$getNoDataValue(band=1)   # 0
```

```
ds$close()
ds <- new(GDALRaster, b5_file, read_only=TRUE)
ds$getNoDataValue(band=1)   # 0
ds$close()

## 0 will be read as NA so don't need to handle zeros in expr
expr <- "(B5-B4)/(B5+B4)"
ndvi_file <- calc(expr = expr,
                  rasterfiles = c(b4_file, b5_file),
                  var.names = c("B4", "B5"),
                  dtName = "Float32",
                  nodata_value = -32767,
                  setRasterNodataValue = TRUE)

ds <- new(GDALRaster, ndvi_file, read_only=TRUE)
ds$getStatistics(band=1, approx_ok=FALSE, force=TRUE)
ds$close()


### Reclassify a variable by rule set

## Combine two raster layers and look for specific combinations. Then
## recode to a new value by rule set.
##
## Based on example in:
##   Stratton, R.D. 2009. Guidebook on LANDFIRE fuels data acquisition,
##   critique, modification, maintenance, and model calibration.
##   Gen. Tech. Rep. RMRS-GTR-220. U.S. Department of Agriculture,
##   Forest Service, Rocky Mountain Research Station. 54 p.
## Context: Refine national-scale fuels data to improve fire simulation
##   results in localized applications.
## Issue: Areas with steep slopes (40+ degrees) were mapped as
##   GR1 (101; short, sparse dry climate grass) and
##   GR2 (102; low load, dry climate grass) but were not carrying fire.
## Resolution: After viewing these areas in Google Earth,
##   NB9 (99; bare ground) was selected as the replacement fuel model.

## look for combinations of slope >= 40 and FBFM 101 or 102
lcp_file <- system.file("extdata/storm_lake.lcp", package="gdalraster")
rasterfiles <- c(lcp_file, lcp_file)
var.names <- c("SLP", "FBFM")
bands <- c(2, 4)
df <- combine(rasterfiles, var.names, bands)
nrow(df)
df_subset <- subset(df, SLP >= 40 & FBFM %in% c(101,102))
print(df_subset)       # twelve combinations meet the criteria
sum(df_subset$count)   # 85 total pixels

## recode these pixels to 99 (bare ground)
## the LCP driver does not support in-place write so make a copy as GTiff
tif_file <- paste0(tempdir(), "/", "storml_lndscp.tif")
createCopy("GTiff", tif_file, lcp_file)
```

```
expr <- "ifelse( SLP >= 40 & FBFM %in% c(101,102), 99, FBFM)"
calc(expr = expr,
     rasterfiles = c(lcp_file, lcp_file),
     bands = c(2, 4),
     var.names = c("SLP", "FBFM"),
     dstfile = tif_file,
     out_band = 4,
     write_mode = "update")

## verify the ouput
rasterfiles <- c(tif_file, tif_file)
df <- combine(rasterfiles, var.names, bands)
df_subset <- subset(df, SLP >= 40 & FBFM %in% c(101,102))
print(df_subset)
sum(df_subset$count)

## if LCP file format is needed: createCopy(tif_file, <new_lcp_file>)
```

CmbTable-class *Class for counting unique combinations of integers*

Description

CmbTable implements a hash table having a vector of integers as the key, and the count of occurrences of each unique integer combination as the value. A unique ID is assigned to each unique combination of input values.

Arguments

keyLen	The number of integer values comprising each combination.
varNames	Character vector of names for the variables in the combination.

Value

An object of class CmbTable. Contains a hash table having a vector of keyLen integers as the key and the count of occurrences of each unique integer combination as the value, along with methods that operate on the table as described in Details.

Usage

```
cmb <- new(CmbTable, keyLen, varNames)

## Methods (see Details)
cmb$update(int_cmb, incr)
cmb$updateFromMatrix(int_cmbs, incr)
cmb$asDataFrame()
```

Details

`new(CmbTable, keyLen, varNames)` Constructor. Returns an object of class CmbTable.

`$update(int_cmb, incr)` Updates the hash table for the integer combination in the numeric vector `int_cmb` (coerced to integer by truncation). If this combination exists in the table, its count will be incremented by `incr`. If the combination is not found in the table, it will be inserted with count set to `incr`. Returns the unique ID assigned to this combination. Combination IDs are sequential integers starting at 1.

`$updateFromMatrix(int_cmbs, incr)` This method is the same as `$update()` but for a numeric matrix of integer combinations `int_cmbs` (coerced to integer by truncation). The matrix is arranged with each column vector forming an integer combination. For example, the rows of the matrix could be one row each from a set of `keyLen` rasters all read at the same extent and pixel resolution (i.e., row-by-row raster overlay). The method calls `$update()` on each combination (each column of `int_cmbs`), incrementing count by `incr` for existing combinations, or inserting new combinations with count set to `incr`. Returns a numeric vector of length `ncol(int_cmbs)` containing the IDs assigned to the combinations.

`$asDataFrame()` Returns the CmbTable as a data frame with column `cmbid` containing the unique combination IDs, column `count` containing the counts of occurrences, and `keyLen` columns named `varNames` containing the integer values comprising each unique combination.

Examples

```
m <- matrix(c(1,2,3,1,2,3,4,5,6,1,3,2,4,5,6,1,1,1), 3, 6, byrow=FALSE)
row.names(m) <- c("v1","v2","v3")
print(m)
cmb <- new(CmbTable, 3, row.names(m))
cmb$updateFromMatrix(m, 1)
cmb$asDataFrame()
cmb$update(c(4,5,6), 1)
cmb$update(c(1,3,5), 1)
cmb$asDataFrame()
```

`combine` *Raster overlay for unique combinations*

Description

`combine()` overlays multiple rasters so that a unique ID is assigned to each unique combination of input values. The input raster layers typically have integer data types (floating point will be coerced to integer by truncation), and must have the same projection, extent and cell size. Pixel counts for each unique combination are obtained, and combination IDs are optionally written to an output raster.

Usage

```
combine(
  rasterfiles,
```

```
  var.names = NULL,
  bands = NULL,
  dstfile = NULL,
  fmt = NULL,
  dtName = "UInt32",
  options = NULL
)
```

Arguments

rasterfiles	Character vector of raster filenames to combine.
var.names	Character vector of length(rasterfiles) containing variable names for each raster layer. Defaults will be assigned if var.names are omitted.
bands	Numeric vector of length(rasterfiles) containing the band number to use for each raster in rasterfiles. Band 1 will be used for each input raster if bands are not specified.
dstfile	Character. Optional output raster filename for writing the per-pixel combination IDs. The output raster will be created (and overwritten if it already exists).
fmt	Character. Output raster format name (e.g., "GTiff" or "HFA").
dtName	Character. Output raster data type name. Combination IDs are sequential integers starting at 1. The data type for the output raster should be large enough to accommodate the potential number of unique combinations of the input values (e.g., "UInt16" or the default "UInt32").
options	Optional list of format-specific creation options in a vector of "NAME=VALUE" pairs. (e.g., options = c("COMPRESS=LZW") to set LZW compression during creation of a GTiff file).

Details

To specify input raster layers that are bands of a multi-band raster file, repeat the filename in rasterfiles and provide the corresponding band numbers in bands. For example:

```
rasterfiles <- c("multi-band.tif", "multi-band.tif", "other.tif")
bands <- c(4, 5, 1)
var.names <- c("multi_b4", "multi_b5", "other")
```

rasterToVRT() provides options for virtual clipping, resampling and pixel alignment, which may be helpful here if the input rasters are not already aligned on a common extent and cell size.

If an output raster of combination IDs is written, the user should verify that the number of combinations obtained did not exceed the range of the output data type. Combination IDs are sequential integers starting at 1. Typical output data types are the unsigned types: Byte (0 to 255), UInt16 (0 to 65,535) and UInt32 (the default, 0 to 4,294,967,295).

combine() can also run on a single raster layer to obtain a table of pixel values and their counts.

Value

A data frame with column cmbid containing the combination IDs, column count containing the pixel counts for each combination, and length(rasterfiles) columns named var.names containing the integer values comprising each unique combination.

See Also

CmbTable-class, GDALRaster-class, calc(), rasterToVRT()

Examples

```
evt_file <- system.file("extdata/storml_evt.tif", package="gdalraster")
evc_file <- system.file("extdata/storml_evc.tif", package="gdalraster")
evh_file <- system.file("extdata/storml_evh.tif", package="gdalraster")
rasterfiles <- c(evt_file, evc_file, evh_file)
var.names <- c("veg_type", "veg_cov", "veg_ht")
df <- combine(rasterfiles, var.names)
nrow(df)
df <- df[order(-df$count),]
head(df, n = 20)

## combine two bands from a multi-band file and write the combination IDs
## to an output raster
lcp_file <- system.file("extdata/storm_lake.lcp", package="gdalraster")
rasterfiles <- c(lcp_file, lcp_file)
bands <- c(4, 5)
var.names <- c("fbfm", "tree_cov")
cmb_file <- paste0(tempdir(), "/", "fbfm_cov_cmbid.tif")
options <- c("COMPRESS=LZW")
df <- combine(rasterfiles, var.names, bands, cmb_file, options = options)
head(df)
ds <- new(GDALRaster, cmb_file, TRUE)
ds$info()
ds$close()
```

---

create *Create a new uninitialized raster*

Description

`create()` makes an empty raster in the specified format.

Usage

```
create(format, dst_filename, xsize, ysize, nbands, dataType, options = NULL)
```

Arguments

`format`	Raster format short name (e.g., "GTiff" or "HFA").
`dst_filename`	Filename to create.
`xsize`	Integer width of raster in pixels.
`ysize`	Integer height of raster in pixels.
`nbands`	Integer number of bands.

dataType	Character data type name. (e.g., common data types include Byte, Int16, UInt16, Int32, Float32).
options	Optional list of format-specific creation options in a vector of "NAME=VALUE" pairs (e.g., `options = c("COMPRESS=LZW")` to set LZW compression during creation of a GTiff file). The APPEND_SUBDATASET=YES option can be specified to avoid prior destruction of existing dataset.

Value

Logical indicating success (invisible `TRUE`). An error is raised if the operation fails.

See Also

`GDALRaster-class`, `createCopy()`, `rasterFromRaster()`

Examples

```
new_file <- paste0(tempdir(), "/", "newdata.tif")
create(format="GTiff", dst_filename=new_file, xsize=143, ysize=107,
       nbands=1, dataType="Int16")
ds <- new(GDALRaster, new_file, read_only=FALSE)
## EPSG:26912 - NAD83 / UTM zone 12N
ds$setProjection(epsg_to_wkt(26912))
gt <- c(323476.1, 30, 0, 5105082.0, 0, -30)
ds$setGeoTransform(gt)
ds$setNoDataValue(band = 1, -9999)
ds$fillRaster(band = 1, -9999, 0)
## ...
## close the dataset when done
ds$close()
```

---

createCopy *Create a copy of a raster*

Description

`createCopy()` copies a raster dataset, optionally changing the format. The extent, cell size, number of bands, data type, projection, and geotransform are all copied from the source raster.

Usage

```
createCopy(format, dst_filename, src_filename, strict = FALSE, options = NULL)
```

Arguments

format	Format short name for the output raster (e.g., "GTiff" or "HFA").
dst_filename	Filename to create.
src_filename	Filename of source raster.
strict	Logical. TRUE if the copy must be strictly equivalent, or more normally FALSE indicating that the copy may adapt as needed for the output format.
options	Optional list of format-specific creation options in a vector of "NAME=VALUE" pairs (e.g., options = c("COMPRESS=LZW") to set LZW compression during creation of a GTiff file). The APPEND_SUBDATASET=YES option can be specified to avoid prior destruction of existing dataset.

Value

Logical indicating success (invisible TRUE). An error is raised if the operation fails.

See Also

GDALRaster-class, create(), rasterFromRaster()

Examples

```
lcp_file <- system.file("extdata/storm_lake.lcp", package="gdalraster")
tif_file <- paste0(tempdir(), "/", "storml_lndscp.tif")
options <- c("COMPRESS=LZW")
createCopy(format="GTiff", dst_filename=tif_file, src_filename=lcp_file,
           options=options)
file.size(lcp_file)
file.size(tif_file)
ds <- new(GDALRaster, tif_file, read_only=FALSE)
ds$getMetadata(band=0, domain="IMAGE_STRUCTURE")
for (band in 1:ds$getRasterCount())
    ds$setNoDataValue(band, -9999)
ds$getStatistics(band=1, approx_ok=FALSE, force=TRUE)
ds$close()
```

DEFAULT_DEM_PROC *List of default DEM processing options*

Description

These values are used in dem_proc() as the default processing options:

```
list(hillshade = c("-z", "1", "-s", "1", "-az", "315",
                   "-alt", "45", "-alg", "Horn",
                   "-combined", "-compute_edges"),
     slope = c("-s", "1", "-alg", "Horn", "-compute_edges"),
     aspect = c("-alg", "Horn", "-compute_edges"),
```

```
            color_relief = character(),
            TRI = c("-alg", "Riley", "-compute_edges"),
            TPI = c("-compute_edges"),
            roughness = c("-compute_edges"))
```

Usage

```
DEFAULT_DEM_PROC
```

Format

An object of class `list` of length 7.

See Also

`dem_proc()`

https://gdal.org/programs/gdaldem.html for a description of all available command-line options for each processing mode

---

DEFAULT_NODATA *List of default nodata values by raster data type*

Description

These values are currently used in `gdalraster` when a nodata value is needed but has not been specified:

```
list("Byte" = 255, "Int8" = -128,
     "UInt16" = 65535, "Int16" = -32767,
     "UInt32" = 4294967293, "Int32" = -2147483647,
     "Float32" = -99999.0, "Float64" = -99999.0)
```

Usage

```
DEFAULT_NODATA
```

Format

An object of class `list` of length 8.

dem_proc *GDAL DEM processing*

Description

dem_proc() generates DEM derivatives from an input elevation raster. This function is a wrapper for the gdaldem command-line utility. See https://gdal.org/programs/gdaldem.html for details.

Usage

```
dem_proc(
  mode,
  srcfile,
  dstfile,
  mode_options = DEFAULT_DEM_PROC[[mode]],
  color_file = NULL
)
```

Arguments

mode	Character. Name of the DEM processing mode. One of hillshade, slope, aspect, color-relief, TRI, TPI or roughness.
srcfile	Filename of the the source elevation raster.
dstfile	Filename of the output raster.
mode_options	An optional character vector of command-line options (see DEFAULT_DEM_PROC for default values).
color_file	Filename of a text file containing lines formatted as: "elevation_value red green blue". Only used when mode = "color-relief".

Value

Logical indicating success (invisible TRUE). An error is raised if the operation fails.

Note

Band 1 of the source elevation raster is read by default, but this can be changed by including a -b command-line argument in mode_options. See the documentation for gdaldem for a description of all available options for each processing mode.

Examples

```
elev_file <- system.file("extdata/storml_elev.tif", package="gdalraster")
slp_file <- paste0(tempdir(), "/", "storml_slp.tif")
dem_proc("slope", elev_file, slp_file)
```

epsg_to_wkt *Convert spatial reference from EPSG code to OGC Well Known Text*

Description

`epsg_to_wkt()` exports the spatial reference for an EPSG code to WKT format.

Usage

```
epsg_to_wkt(epsg, pretty = FALSE)
```

Arguments

`epsg`	Integer EPSG code.
`pretty`	Logical. `TRUE` to return a nicely formatted WKT string for display to a person. `FALSE` for a regular WKT string (the default).

Details

As of GDAL 3.0, the default format for WKT export is OGC WKT 1. The WKT version can be overridden by using the `OSR_WKT_FORMAT` configuration option (see `set_config_option()`). Valid values are one of: `SFSQL`, `WKT1_SIMPLE`, `WKT1`, `WKT1_GDAL`, `WKT1_ESRI`, `WKT2_2015`, `WKT2_2018`, `WKT2`, `DEFAULT`. If `SFSQL`, a WKT1 string without AXIS, TOWGS84, AUTHORITY or EXTENSION node is returned. If `WKT1_SIMPLE`, a WKT1 string without AXIS, AUTHORITY or EXTENSION node is returned. `WKT1` is an alias of `WKT1_GDAL`. `WKT2` will default to the latest revision implemented (currently `WKT2_2018`). `WKT2_2019` can be used as an alias of `WKT2_2018` since GDAL 3.2

Value

Character string containing OGC WKT.

See Also

`srs_to_wkt()`

Examples

```
epsg_to_wkt(5070)
writeLines(epsg_to_wkt(5070, pretty=TRUE))
set_config_option("OSR_WKT_FORMAT", "WKT2")
writeLines(epsg_to_wkt(5070, pretty=TRUE))
set_config_option("OSR_WKT_FORMAT", "")
```

`fillNodata` *Fill selected pixels by interpolation from surrounding areas*

Description

`fillNodata()` is a wrapper for `GDALFillNodata()` in the GDAL Algorithms API. This algorithm will interpolate values for all designated nodata pixels (pixels having an intrinsic nodata value, or marked by zero-valued pixels in the optional raster specified in `mask_file`). For each nodata pixel, a four direction conic search is done to find values to interpolate from (using inverse distance weighting). Once all values are interpolated, zero or more smoothing iterations (3x3 average filters on interpolated pixels) are applied to smooth out artifacts.

Usage

```
fillNodata(
  filename,
  band,
  mask_file = "",
  max_dist = 100,
  smooth_iterations = 0L
)
```

Arguments

`filename`	Filename of input raster in which to fill nodata pixels.
`band`	Integer band number to modify in place.
`mask_file`	Optional filename of raster to use as a validity mask (band 1 is used, zero marks nodata pixels, non-zero marks valid pixels).
`max_dist`	Maximum distance (in pixels) that the algorithm will search out for values to interpolate (100 pixels by default).
`smooth_iterations`	The number of 3x3 average filter smoothing iterations to run after the interpolation to dampen artifacts (0 by default).

Value

Logical indicating success (invisible `TRUE`). An error is raised if the operation fails.

Note

The input raster will be modified in place. It should not be open in a `GDALRaster` object while processing with `fillNodata()`.

Examples

```
## fill nodata edge pixels in the elevation raster
elev_file <- system.file("extdata/storml_elev.tif", package="gdalraster")

## get count of nodata
df = combine(elev_file)
head(df)
df[is.na(df$storml_elev),]

## make a copy that will be modified
mod_file <- paste0(tempdir(), "/", "storml_elev_fill.tif")
file.copy(elev_file,  mod_file)

fillNodata(mod_file, band=1)

df_mod = combine(mod_file)
head(df_mod)
df_mod[is.na(df_mod$storml_elev_fill),]
```

GDALRaster-class *Class encapsulating a subset of the GDAL Raster C API*

Description

GDALRaster provides an interface for accessing a raster dataset via GDAL and calling methods on the underlying GDALDataset, GDALDriver and GDALRasterBand objects. See https://gdal.org/api/index.html for details of the GDAL API.

Arguments

filename	Character string containing the file name of a raster dataset to open, as full path or relative to the current working directory. In some cases, filename may not refer to a physical file, but instead contain format-specific information on how to access a dataset (GDAL raster format descriptions: https://gdal.org/drivers/raster/index.html).
read_only	Logical. TRUE to open the dataset read-only, or FALSE to open with write access.

Value

An object of class GDALRaster which contains a pointer to the opened dataset, and methods that operate on the dataset as described in Details.

Usage

```
ds <- new(GDALRaster, filename, read_only)

## Methods (see Details)
ds$getFilename()
```

```
ds$open(read_only)
ds$isOpen()

ds$info()

ds$getDriverShortName()
ds$getDriverLongName()

ds$getRasterXSize()
ds$getRasterYSize()
ds$getGeoTransform()
ds$setGeoTransform(transform)
ds$getProjectionRef()
ds$setProjection(projection)
ds$bbox()
ds$res()
ds$dim()

ds$getRasterCount()
ds$getBlockSize(band)
ds$getOverviewCount(band)
ds$buildOverviews(resampling, levels, bands)
ds$getDataTypeName(band)
ds$getStatistics(band, approx_ok, force)
ds$getNoDataValue(band)
ds$setNoDataValue(band, nodata_value)
ds$deleteNoDataValue(band)
ds$getUnitType(band)
ds$setUnitType(band, unit_type)
ds$getScale(band)
ds$setScale(band, scale)
ds$getOffset(band)
ds$setOffset(band, offset)

ds$getMetadata(band, domain)
ds$getMetadataItem(band, mdi_name, domain)
ds$setMetadataItem(band, mdi_name, mdi_value, domain)

ds$read(band, xoff, yoff, xsize, ysize, out_xsize, out_ysize)
ds$write(band, xoff, yoff, xsize, ysize, rasterData)
ds$fillRaster(value, ivalue)

ds$getChecksum(band, xoff, yoff, xsize, ysize)

ds$close()
```

Details

`new(GDALRaster, filename, read_only)` Constructor. Returns an object of class `GDALRaster`.

`$getFilename()` Returns a character string containing the filename associated with this `GDALRaster` object.

`$open(read_only)` (Re-)opens the raster dataset on the existing filename. Use this method to open a dataset that has been closed using `$close()`. May be used to re-open a dataset with a different read/write access (`read_only` set to `TRUE` or `FALSE`). The method will first close an open dataset, so it is not required to call `$close()` explicitly in this case. No return value, called for side effects.

`$isOpen()` Returns logical indicating whether the associated raster dataset is open.

`$info()` Prints various information about the raster dataset to the console (no return value, called for that side effect only). Equivalent to the output of the `gdalinfo` command-line utility (`gdalinfo -norat -noct filename`). Intended here as an informational convenience function.

`$getDriverShortName()` Returns the short name of the raster format driver (e.g., "HFA").

`$getDriverLongName()` Returns the long name of the raster format driver (e.g., "Erdas Imagine Images (.img)").

`$getRasterXSize()` Returns the number of pixels along the x dimension.

`$getRasterYSize()` Returns the number of pixels along the y dimension.

`$getGeoTransform()` Returns the affine transformation coefficients for transforming between pixel/line raster space (column/row) and projection coordinate space (geospatial x/y). The return value is a numeric vector of length six. See https://gdal.org/tutorials/geotransforms_tut.html for details of the affine transformation. *With 1-based indexing in R*, the geotransform vector contains (in map units of the raster spatial reference system):

GT[1]	x-coordinate of upper-left corner of the upper-left pixel
GT[2]	x-component of pixel width
GT[3]	row rotation (zero for north-up raster)
GT[4]	y-coordinate of upper-left corner of the upper-left pixel
GT[5]	column rotation (zero for north-up raster)
GT[6]	y-component of pixel height (negative for north-up raster)

`$setGeoTransform(transform)` Sets the affine transformation coefficients on this dataset. `transform` is a numeric vector of length six. Returns logical TRUE on success or FALSE if the geotransform could not be set.

`$getProjectionRef()` Returns the coordinate reference system of the raster as an OpenGIS WKT format string. An empty string is returned when a projection definition is not available.

`$setProjection(projection)` Sets the projection reference for this dataset. `projection` is a string in OGC WKT format. Returns logical TRUE on success or FALSE if the projection could not be set.

`$bbox()` Returns a numeric vector of length four containing the bounding box (xmin, ymin, xmax, ymax) assuming this is a north-up raster.

`$res()` Returns a numeric vector of length two containing the resolution (pixel width, pixel height as positive values) assuming this is a north-up raster.

`$dim()` Returns an integer vector of length three containing the raster dimensions. Equivalent to: `c(ds$getRasterXSize(), ds$getRasterYSize(), ds$getRasterCount())`

`$getRasterCount()` Returns the number of raster bands on this dataset. For the methods described below that operate on individual bands, the `band` argument is the integer band number (1-based).

`$getBlockSize(band)` Returns an integer vector of length two (Xsize, Ysize) containing the "natural" block size of band. GDAL has a concept of the natural block size of rasters so that applications can organize data access efficiently for some file formats. The natural block size is the block size that is most efficient for accessing the format. For many formats this is simply a whole row in which case block Xsize is the same as `$getRasterXSize()` and block Ysize is 1. However, for tiled images block size will typically be the tile size. Note that the X and Y block sizes don't have to divide the image size evenly, meaning that right and bottom edge blocks may be incomplete.

`$getOverviewCount(band)` Returns the number of overview layers available for band.

`$buildOverviews(resampling, levels, bands)` Build one or more raster overview images using the specified downsampling algorithm. `resampling` is one of "AVERAGE", "AVERAGE_MAGPHASE", "RMS", "BILINEAR", "CUBIC", "CUBICSPLINE", "GAUSS", "LANCZOS", "MODE", "NEAREST", or "NONE". `levels` is an integer vector giving the list of overview decimation factors to build (e.g., `c(2, 4, 8)`), or `0` to delete all overviews (at least for external overviews (.ovr) and GTiff internal overviews). `bands` is an integer vector giving a list of band numbers to build overviews for, or `0` to build for all bands. Note that for GTiff, overviews will be created internally if the dataset is open in update mode, while external overviews (.ovr) will be created if the dataset is open read-only. Starting with GDAL 3.2, the GDAL_NUM_THREADS configuration option can be set to "ALL_CPUS" or an integer value to specify the number of threads to use for overview computation (see `set_config_option()`). No return value, called for side effects.

`$getDataTypeName(band)` Returns the name of the pixel data type for band. The possible data types are:

`Unknown`	Unknown or unspecified type
`Byte`	8-bit unsigned integer
`Int8`	8-bit signed integer (GDAL >= 3.7)
`UInt16`	16-bit unsigned integer
`Int16`	16-bit signed integer
`UInt32`	32-bit unsigned integer
`Int32`	32-bit signed integer
`UInt64`	64-bit unsigned integer (GDAL >= 3.5)
`Int64`	64-bit signed integer (GDAL >= 3.5)
`Float32`	32-bit floating point
`Float64`	64-bit floating point
`CInt16`	Complex Int16
`CInt32`	Complex Int32
`CFloat32`	Complex Float32
`CFloat64`	Complex Float64

Some raster formats including GeoTIFF (`GTiff`) and Erdas Imagine .img (`HFA`) support sub-byte data types. Rasters can be created with these data types by specifying the `NBITS=n` creation option where n=1...7 for GTiff or n=1/2/4 for HFA. In these cases, `$getDataTypeName()` reports the apparent type `Byte`. GTiff also supports n=9...15 (`UInt16` type) and n=17...31 (`UInt32` type), and n=16 is accepted for `Float32` to generate half-precision floating point values.

`$getStatistics(band, approx_ok, force)` Returns a numeric vector of length four containing the minimum, maximum, mean and standard deviation of pixel values in band (excluding nodata pixels). Some raster formats will cache statistics allowing fast retrieval after the first request.

`approx_ok`:

- TRUE: Approximate statistics are sufficient, in which case overviews or a subset of raster tiles may be used in computing the statistics.
- FALSE: All pixels will be read and used to compute statistics (if computation is forced).

force:

- TRUE: The raster will be scanned to compute statistics. Once computed, statistics will generally be "set" back on the raster band if the format supports caching statistics. (Note: ComputeStatistics() in the GDAL API is called automatically here. This is a change in the behavior of GetStatistics() in the API, to a definitive force.)
- FALSE: Results will only be returned if it can be done quickly (i.e., without scanning the raster, typically by using pre-existing STATISTICS_xxx metadata items). NAs will be returned if statistics cannot be obtained quickly.

$getNoDataValue(band) Returns the nodata value for band if one exists. This is generally a special value defined to mark pixels that are not valid data. NA is returned if a nodata value is not defined for band. Not all raster formats support a designated nodata value.

$setNoDataValue(band, nodata_value) Sets the nodata value for band. nodata_value is a numeric value to be defined as the nodata marker. Depending on the format, changing the nodata value may or may not have an effect on the pixel values of a raster that has just been created (often not). It is thus advised to call $fillRaster() explicitly if the intent is to initialize the raster to the nodata value. In any case, changing an existing nodata value, when one already exists on an initialized dataset, has no effect on the pixels whose values matched the previous nodata value. Returns logical TRUE on success or FALSE if the nodata value could not be set.

$deleteNoDataValue(band) Removes the nodata value for band. This affects only the definition of the nodata value for raster formats that support one (does not modify pixel values). No return value, called for side effects. An error is raised if the nodata value cannot be removed.

$getUnitType(band) Returns the name of the unit type of the pixel values for band (e.g., "m" or "ft"). An empty string "" is returned if no units are available.

$setUnitType(band, unit_type) Sets the name of the unit type of the pixel values for band. unit_type should be one of "" (the default indicating it is unknown), "m" indicating meters, or "ft" indicating feet, though other nonstandard values are allowed. Returns logical TRUE on success or FALSE if the unit type could not be set.

$getScale(band) Returns the pixel value scale (units value = (raw value * scale) + offset) for band. This value (in combination with the getOffset() value) can be used to transform raw pixel values into the units returned by getUnitType(). Returns NA if a scale value is not defined for this band.

$setScale(band, scale) Sets the pixel value scale (units value = (raw value * scale) + offset) for band. Many raster formats do not implement this method. Returns logical TRUE on success or FALSE if the scale could not be set.

$getOffset(band) Returns the pixel value offset (units value = (raw value * scale) + offset) for band. This value (in combination with the getScale() value) can be used to transform raw pixel values into the units returned by getUnitType(). Returns NA if an offset value is not defined for this band.

$setOffset(band, offset) Sets the pixel value offset (units value = (raw value * scale) + offset) for band. Many raster formats do not implement this method. Returns logical TRUE on success or FALSE if the offset could not be set.

`$getMetadata(band, domain)` Returns a character vector of all metadata `name=value` pairs that exist in the specified `domain`, or `""` (empty string) if there are no metadata items in `domain` (metadata in the context of the GDAL Raster Data Model: https://gdal.org/user/raster_data_model.html). Set `band = 0` to retrieve dataset-level metadata, or to an integer band number to retrieve band-level metadata. Set `domain = ""` (empty string) to retrieve metadata in the default domain.

`$getMetadataItem(band, mdi_name, domain)` Returns the value of a specific metadata item named `mdi_name` in the specified `domain`, or `""` (empty string) if no matching item is found. Set `band = 0` to retrieve dataset-level metadata, or to an integer band number to retrieve band-level metadata. Set `domain = ""` (empty string) to retrieve an item in the default domain.

`$setMetadataItem(band, mdi_name, mdi_value, domain)` Sets the value (`mdi_value`) of a specific metadata item named `mdi_name` in the specified `domain`. Set `band = 0` to set dataset-level metadata, or to an integer band number to set band-level metadata. Set `domain = ""` (empty string) to set an item in the default domain.

`$read(band, xoff, yoff, xsize, ysize, out_xsize, out_ysize)` Reads a region of raster data from `band`. The method takes care of pixel decimation / replication if the output size (`out_xsize * out_ysize`) is different than the size of the region being accessed (`xsize * ysize`). `xoff` is the pixel (column) offset to the top left corner of the region of the band to be accessed (zero to start from the left side). `yoff` is the line (row) offset to the top left corner of the region of the band to be accessed (zero to start from the top). *Note that raster row/column offsets use 0-based indexing.* `xsize` is the width in pixels of the region to be accessed. `ysize` is the height in pixels of the region to be accessed. `out_xsize` is the width of the output array into which the desired region will be read (typically the same value as xsize). `out_ysize` is the height of the output array into which the desired region will be read (typically the same value as ysize). Returns a numeric or complex vector containing the values that were read. It is organized in left to right, top to bottom pixel order. NA will be returned in place of the nodata value if the raster dataset has a nodata value defined for this band. Data are read as R `integer` type when possible for the raster data type (Byte, Int8, Int16, UInt16, Int32), otherwise as type `double` (UInt32, Float32, Float64). No rescaling of the data is performed (see `$getScale()` and `$getOffset()` above). An error is raised if the read operation fails.

`$write(band, xoff, yoff, xsize, ysize, rasterData)` Writes a region of raster data to `band`. `xoff` is the pixel (column) offset to the top left corner of the region of the band to be accessed (zero to start from the left side). `yoff` is the line (row) offset to the top left corner of the region of the band to be accessed (zero to start from the top). *Note that raster row/column offsets use 0-based indexing.* `xsize` is the width in pixels of the region to write. `ysize` is the height in pixels of the region to write. `rasterData` is a numeric or complex vector containing values to write. It is organized in left to right, top to bottom pixel order. NA in `rasterData` should be replaced with a suitable nodata value prior to writing (see `$getNoDataValue()` and `$setNoDataValue()` above). An error is raised if the operation fails (no return value).

`$fillRaster(band, value, ivalue)` Fills `band` with a constant value. Used to clear a band to a specified default value. `value` is the fill value (real component). `ivalue` is the imaginary component of fill value for a raster with complex data type. Set `ivalue = 0` for real data types. No return value, called for side effects.

`$getChecksum(band, xoff, yoff, xsize, ysize)` Returns a 16-bit integer (0-65535) checksum from a region of raster data on `band`. Floating point data are converted to 32-bit integer so decimal portions of such raster data will not affect the checksum. Real and imaginary components of complex bands influence the result. `xoff` is the pixel (column) offset of the window to read. `yoff` is the

line (row) offset of the window to read. *Raster row/column offsets use 0-based indexing.* `xsize` is the width in pixels of the window to read. `ysize` is the height in pixels of the window to read.

`$close()` Closes the GDAL dataset (no return value, called for side effects). Calling `$close()` results in proper cleanup, and flushing of any pending writes. Forgetting to close a dataset opened in update mode on some formats such as GTiff could result in being unable to open it afterwards. The `GDALRaster` object is still available after calling `$close()`. The dataset can be re-opened on the existing `filename` with `$open(read_only=TRUE)` or `$open(read_only=FALSE)`.

Note

The `$read()` method will perform automatic resampling if the specified output size (`out_xsize * out_ysize`) is different than the size of the region being read (`xsize * ysize`). In that case, the GDAL_RASTERIO_RESAMPLING configuration option could also be defined to override the default resampling to one of BILINEAR, CUBIC, CUBICSPLINE, LANCZOS, AVERAGE or MODE (see `set_config_option()`).

See Also

Package overview in `help("gdalraster-package")`

`create()`, `createCopy()`, `rasterFromRaster()`, `rasterToVRT()`

`fillNodata()`, `warp()`, `plot_raster()`

`read_ds()` is a convenience wrapper for `GDALRaster$read()`.

Examples

```
lcp_file <- system.file("extdata/storm_lake.lcp", package="gdalraster")
ds <- new(GDALRaster, lcp_file, read_only=TRUE)

## print information about the dataset to the console
ds$info()

## retrieve the raster format name
ds$getDriverShortName()
ds$getDriverLongName()

## retrieve dataset parameters
ds$getRasterXSize()
ds$getRasterYSize()
ds$getGeoTransform()
ds$getProjectionRef()
ds$getRasterCount()
ds$bbox()
ds$res()
ds$dim()

## retrieve some band-level parameters
ds$getBlockSize(band=1)
ds$getOverviewCount(band=1)
ds$getDataTypeName(band=1)
# LCP format does not support an intrinsic nodata value so this returns NA:
```

```
ds$getNoDataValue(band=1)

## LCP driver reports several dataset- and band-level metadata
## see the format description at https://gdal.org/drivers/raster/lcp.html
## set band=0 to retrieve dataset-level metadata
## set domain="" (empty string) for the default metadata domain
ds$getMetadata(band=0, domain="")

## retrieve metadata for a band as a vector of name=value pairs
ds$getMetadata(band=4, domain="")

## retrieve the value of a specific metadata item
ds$getMetadataItem(band=2, mdi_name="SLOPE_UNIT_NAME", domain="")

## read one row of pixel values from band 1 (elevation)
## raster row/column index are 0-based
## the upper left corner is the origin
## read the tenth row:
ncols <- ds$getRasterXSize()
rowdata <- ds$read(band=1, xoff=0, yoff=9,
                   xsize=ncols, ysize=1,
                   out_xsize=ncols, out_ysize=1)
head(rowdata)

ds$close()

## create a new raster using lcp_file as a template
new_file <- paste0(tempdir(), "/", "storml_newdata.tif")
rasterFromRaster(srcfile = lcp_file,
                 dstfile = new_file,
                 nbands = 1,
                 dtName = "Byte",
                 init = -9999)
ds_new <- new(GDALRaster, new_file, read_only=FALSE)

## write random values to all pixels
set.seed(42)
ncols <- ds_new$getRasterXSize()
nrows <- ds_new$getRasterYSize()
for (row in 0:(nrows-1)) {
    rowdata <- round(runif(ncols, 0, 100))
    ds_new$write(band=1, xoff=0, yoff=row, xsize=ncols, ysize=1, rowdata)
}

## re-open in read-only mode when done writing
## this will ensure flushing of any pending writes (implicit $close)
ds_new$open(read_only=TRUE)

## getStatistics returns min, max, mean, sd, and sets stats in the metadata
ds_new$getStatistics(band=1, approx_ok=FALSE, force=TRUE)
ds_new$getMetadataItem(band=1, "STATISTICS_MEAN", "")

## close the dataset for proper cleanup
```

```
ds_new$close()


## using a GDAL Virtual File System handler '/vsicurl/'
## see: https://gdal.org/user/virtual_file_systems.html
url <- "/vsicurl/https://raw.githubusercontent.com/"
url <- paste0(url, "usdaforestservice/gdalraster/main/sample-data/")
url <- paste0(url, "lf_elev_220_mt_hood_utm.tif")

ds <- new(GDALRaster, url, read_only=TRUE)
plot_raster(ds, main="Mount Hood elevation (m)")
ds$close()
```

---

gdal_version *Get GDAL version*

Description

gdal_version() returns runtime version information.

Usage

```
gdal_version()
```

Value

Character vector of length four containing:

- "–version" - one line version message, e.g., “GDAL 3.6.3, released 2023/03/12”
- "GDAL_VERSION_NUM" - formatted as a string, e.g., “30603000” for GDAL 3.6.3.0
- "GDAL_RELEASE_DATE" - formatted as a string, e.g., “20230312”
- "GDAL_RELEASE_NAME" - e.g., “3.6.3”

Examples

```
gdal_version()
```

get_cache_used *Get the size of memory in use by the GDAL block cache*

Description

`get_cache_used()` returns the amount of memory currently in use for GDAL block caching. This a wrapper for `GDALGetCacheUsed64()` with return value as MB.

Usage

```
get_cache_used()
```

Value

Integer. Amount of cache memory in use in MB.

Examples

```
get_cache_used()
```

get_config_option *Get GDAL configuration option*

Description

`get_config_option()` gets the value of GDAL runtime configuration option. Configuration options are essentially global variables the user can set. They are used to alter the default behavior of certain raster format drivers, and in some cases the GDAL core. For a full description and listing of available options see https://gdal.org/user/configoptions.html.

Usage

```
get_config_option(key)
```

Arguments

`key`	Character name of a configuration option.

Value

Character. The value of a (key, value) option previously set with `set_config_option()`. An empty string (`""`) is returned if `key` is not found.

See Also

`set_config_option()`

Examples

```
## this option is set during initialization of the gdalraster package
get_config_option("OGR_CT_FORCE_TRADITIONAL_GIS_ORDER")
```

`get_pixel_line` *Raster pixel/line from geospatial x,y coordinates*

Description

`get_pixel_line()` converts geospatial coordinates to pixel/line (raster column, row numbers). The upper left corner pixel is the raster origin (0,0) with column, row increasing left to right, top to bottom.

Usage

```
get_pixel_line(xy, gt)
```

Arguments

`xy`	Numeric array of geospatial x,y coordinates in the same spatial reference system as `gt`.
`gt`	Numeric vector of length six. The affine geotransform for the raster.

Value

Integer array of raster pixel/line.

See Also

`GDALRaster$getGeoTransform()`, `inv_geotransform()`

Examples

```
pt_file <- system.file("extdata/storml_pts.csv", package="gdalraster")
## id, x, y in NAD83 / UTM zone 12N
pts <- read.csv(pt_file)
print(pts)
raster_file <- system.file("extdata/storm_lake.lcp", package="gdalraster")
ds <- new(GDALRaster, raster_file, read_only=TRUE)
gt <- ds$getGeoTransform()
get_pixel_line(as.matrix(pts[,-1]), gt)
ds$close()
```

has_geos *Is GEOS available?*

Description

has_geos() returns a logical value indicating whether GDAL was built against the GEOS library.

Usage

```
has_geos()
```

Value

Logical. TRUE if GEOS is available, otherwise FALSE.

Examples

```
has_geos()
```

inv_geotransform *Invert geotransform*

Description

inv_geotransform() inverts a vector of geotransform coefficients. This converts the equation from being:
raster pixel/line (column/row) -> geospatial x/y coordinate
to:
geospatial x/y coordinate -> raster pixel/line (column/row)

Usage

```
inv_geotransform(gt)
```

Arguments

gt	Numeric vector of length six containing the geotransform to invert.

Value

Numeric vector of length six containing the inverted geotransform. The output vector will contain NAs if the input geotransform is uninvertable.

See Also

GDALRaster$getGeoTransform(), get_pixel_line()

Examples

```
elev_file <- system.file("extdata/storml_elev.tif", package="gdalraster")
ds <- new(GDALRaster, elev_file, read_only=TRUE)
gt <- ds$getGeoTransform()
ds$close()
invgt <- inv_geotransform(gt)

ptX = 324181.7
ptY = 5103901.4

## for a point x, y in the spatial reference system of elev_file
## raster pixel (column number):
pixel <- floor(invgt[1] +
               invgt[2] * ptX +
               invgt[3] * ptY)

## raster line (row number):
line <- floor(invgt[4] +
              invgt[5] * ptX +
              invgt[6] * ptY)

## get_pixel_line() applies this conversion
```

`inv_project` *Inverse project geospatial x/y coordinates to longitude/latitude*

Description

`inv_project()` transforms geospatial x/y coordinates to longitude/latitude in the same geographic coordinate system used by the given projected spatial reference system. The output long/lat can optionally be set to a specific geographic coordinate system by specifying a well known name (see Details).

Usage

```
inv_project(pts, srs, well_known_gcs = "")
```

Arguments

`pts`	A two-column data frame or numeric matrix containing geospatial x/y coordinates
`srs`	Character string in OGC WKT format specifying the projected spatial reference system for `pts`.
`well_known_gcs`	Optional character string containing a supported well known name of a geographic coordinate system (see Details for supported values).

Details

By default, the geographic coordinate system of the projection specified by `srs` will be used. If a specific geographic coordinate system is desired, then `well_known_gcs` can be set to one of the values below:

`EPSG:n`	where `n` is the code of a geographic CRS
`WGS84`	same as `EPSG:4326`
`WGS72`	same as `EPSG:4322`
`NAD83`	same as `EPSG:4269`
`NAD27`	same as `EPSG:4267`
`CRS84`	same as `WGS84`
`CRS72`	same as `WGS72`
`CRS27`	same as `NAD27`

The returned array will always be in longitude, latitude order (traditional GIS order) regardless of the axis order defined for the names above.

Value

Numeric array of longitude, latitude. An error is raised if the transformation cannot be performed.

See Also

`transform_xy()`

Examples

```
pt_file <- system.file("extdata/storml_pts.csv", package="gdalraster")
## id, x, y in NAD83 / UTM zone 12N
pts <- read.csv(pt_file)
print(pts)
inv_project(pts[,-1], epsg_to_wkt(26912))
inv_project(pts[,-1], epsg_to_wkt(26912), "NAD27")
```

---

plot_raster *Display raster data*

Description

`plot_raster()` displays raster data using base `graphics`.

Usage

```
plot_raster(
  data,
  xsize = NULL,
  ysize = NULL,
```

```
  nbands = 1,
  max_pixels = 2.5e+07,
  col_tbl = NULL,
  normalize = TRUE,
  minmax_def = NULL,
  minmax_pct_cut = NULL,
  col_map_fn = NULL,
  xlim = c(0, xsize),
  ylim = c(ysize, 0),
  interpolate = TRUE,
  asp = 1,
  axes = TRUE,
  main = "",
  xlab = "x",
  ylab = "y",
  xaxs = "i",
  yaxs = "i",
  legend = NULL,
  digits = 2,
  na_col = rgb(0, 0, 0, 0),
  ...
)
```

Arguments

data	Either a GDALRaster object from which data will be read, or a numeric vector of pixel values arranged in left to right, top to bottom order.
xsize	The number of pixels along the x dimension in data. If data is a GDALRaster object, specifies the size at which the raster will be read (used for argument out_xsize in GDALRaster$read()). By default, the entire raster will be read at full resolution.
ysize	The number of pixels along the y dimension in data. If data is a GDALRaster object, specifies the size at which the raster will be read (used for argument out_ysize in GDALRaster$read()). By default, the entire raster will be read at full resolution.
nbands	The number of bands in data. Must be either 1 (grayscale) or 3 (RGB). For RGB, data are interleaved by band.
max_pixels	The maximum number of pixels that the function will use for display (per band). An error is raised if (xsize * ysize) exceeds this value. Setting to NULL turns off this check.
col_tbl	A color table as a matrix or data frame with four columns. Column 1 contains the numeric raster values, columns 2:4 contain the intensities (between 0 and 1) of the red, green and blue primaries.
normalize	Logical. TRUE to rescale pixel values so that their range is [0,1], normalized to the full range of the pixel data by default (min(data), max(data), per band). Ignored if col_tbl is used. Set normalize to FALSE if a color map function is used that operates on raw pixel values (see col_map_fn below).

minmax_def	Normalize to user-defined min/max values (in terms of the pixel data, per band). For single-band grayscale, a numeric vector of length two containing min, max. For 3-band RGB, a numeric vector of length six containing b1_min, b2_min, b3_min, b1_max, b2_max, b3_max.
minmax_pct_cut	Normalize to a truncated range of the pixel data using percentile cutoffs (removes outliers). A numeric vector of length two giving the percentiles to use (e.g., c(2, 98)). Applied per band. Ignored if minmax_def is used.
col_map_fn	An optional color map function (default is grDevices::gray for single-band data or grDevices::rgb for 3-band). Ignored if col_tbl is used. Set normalize to FALSE if using a color map function that operates on raw pixel values.
xlim	Numeric vector of length two giving the x coordinate range. The default uses pixel/line coordinates (c(0, xsize)).
ylim	Numeric vector of length two giving the y coordinate range. The default uses pixel/line coordinates (c(ysize, 0)).
interpolate	Logical indicating whether to apply linear interpolation to the image when drawing (default TRUE).
asp	Numeric. The aspect ratio y/x (see ?plot.window).
axes	Logical. TRUE to draw axes (the default).
main	The main title (on top).
xlab	Title for the x axis (see ?title).
ylab	Title for the y axis (see ?title).
xaxs	The style of axis interval calculation to be used for the x axis (see ?par).
yaxs	The style of axis interval calculation to be used for the y axis (see ?par).
legend	Logical indicating whether to include a legend on the plot. By default, a legend will be included if plotting single-band data without col_tbl specified. Legend is not currently supported for color tables, or with 3-band RGB data.
digits	The number of digits to display after the decimal point in the legend labels when raster data are floating point.
na_col	Color to use for NA as a 7- or 9-character hexadecimal code. The default is transparent ("#00000000", the return value of rgb(0,0,0,0)).
...	Other parameters to be passed to plot.default().

Note

plot_raster() uses the function graphics::rasterImage() for plotting which is not supported on some devices (see ?rasterImage).

If data is an object of class GDALRaster, then plot_raster() will attempt to read the entire raster into memory by default (unless the number of pixels per band would exceed max_pixels). A reduced resolution overview can be read by setting xsize, ysize smaller than the raster size on disk. (If data is instead specified as a vector of pixel values, a reduced resolution overview would be read by setting out_xsize and out_ysize smaller than the raster region defined by xsize, ysize in a call to GDALRaster$read()). The GDAL_RASTERIO_RESAMPLING configuration option can be defined to override the default resampling (NEAREST) to one of BILINEAR, CUBIC, CUBICSPLINE, LANCZOS, AVERAGE or MODE, for example:

```
set_config_option("GDAL_RASTERIO_RESAMPLING", "BILINEAR")
```

See Also

GDALRaster$read(), read_ds(), set_config_option()

Examples

```
## Elevation
elev_file <- system.file("extdata/storml_elev.tif", package="gdalraster")
ds <- new(GDALRaster, elev_file, read_only=TRUE)

# grayscale
plot_raster(ds, main="Storm Lake elevation (m)")

# color ramp from user-defined palette
elev_pal <- c("#00A60E","#63C600","#E6E600","#E9BD3B",
              "#ECB176","#EFC2B3","#F2F2F2")
ramp <- scales::colour_ramp(elev_pal, alpha=FALSE)
plot_raster(ds, col_map_fn=ramp, main="Storm Lake elevation (m)")

ds$close()

## Landsat band combination
b4_file <- system.file("extdata/sr_b4_20200829.tif", package="gdalraster")
b5_file <- system.file("extdata/sr_b5_20200829.tif", package="gdalraster")
b6_file <- system.file("extdata/sr_b6_20200829.tif", package="gdalraster")
band_files <- c(b6_file, b5_file, b4_file)

r <- vector("integer")
for (f in band_files) {
  ds <- new(GDALRaster, f, read_only=TRUE)
  dm <- ds$dim()
  r <- c(r, read_ds(ds))
  ds$close()
}

plot_raster(r, xsize=dm[1], ysize=dm[2], nbands=3,
            main="Landsat 6-5-4 (vegetative analysis)")

## LANDFIRE Existing Vegetation Cover (EVC) with color map
evc_file <- system.file("extdata/storml_evc.tif", package="gdalraster")

# colors from the CSV attribute table distributed by LANDFIRE
evc_csv <- system.file("extdata/LF20_EVC_220.csv", package="gdalraster")
vat <- read.csv(evc_csv)
head(vat)
vat <- vat[,c(1,6:8)]

ds <- new(GDALRaster, evc_file, read_only=TRUE)
plot_raster(ds, col_tbl=vat, interpolate=FALSE,
            main="Storm Lake LANDFIRE EVC")

ds$close()
```

rasterFromRaster *Create a raster from an existing raster as template*

Description

rasterFromRaster() creates a new raster with spatial reference, extent and resolution taken from a template raster, without copying data. Optionally changes the format, number of bands, data type and nodata value, sets driver-specific dataset creation options, and initializes to a value.

Usage

```
rasterFromRaster(
  srcfile,
  dstfile,
  fmt = NULL,
  nbands = NULL,
  dtName = NULL,
  options = NULL,
  init = NULL,
  dstnodata = init
)
```

Arguments

srcfile	Source raster filename.
dstfile	Output raster filename.
fmt	Output raster format name (e.g., "GTiff" or "HFA"). Will attempt to guess from the output filename if fmt is not specified.
nbands	Number of output bands.
dtName	Output raster data type name. (e.g., "Byte", "Int16", "UInt16", "Int32" or "Float32").
options	Optional list of format-specific creation options in a vector of "NAME=VALUE" pairs. (e.g., options = c("COMPRESS=LZW") to set LZW compression during creation of a GTiff file).
init	Numeric value to initialize all pixels in the output raster.
dstnodata	Numeric nodata value for the output raster.

Value

Returns the destination filename invisibly.

See Also

GDALRaster-class, create(), createCopy(), rasterToVRT()

Examples

```
## band 2 in a FARSITE landscape file has slope degrees
## convert slope degrees to slope percent in a new raster
lcp_file <- system.file("extdata/storm_lake.lcp", package="gdalraster")
ds_lcp <- new(GDALRaster, lcp_file, read_only=TRUE)
ds_lcp$getMetadata(band=2, domain="")

slpp_file <- paste0(tempdir(), "/", "storml_slpp.tif")
options = c("COMPRESS=LZW")
rasterFromRaster(srcfile = lcp_file,
                 dstfile = slpp_file,
                 nbands = 1,
                 dtName = "Int16",
                 options = options,
                 init = -32767)
ds_slp <- new(GDALRaster, slpp_file, read_only=FALSE)

## slpp_file is initialized to -32767 and nodata value set
ds_slp$getNoDataValue(band=1)

## extent and cell size are the same as lcp_file
ds_lcp$bbox()
ds_lcp$res()
ds_slp$bbox()
ds_slp$res()

## convert slope degrees in lcp_file band 2 to slope percent in slpp_file
## bring through LCP nodata -9999 to the output nodata value
ncols <- ds_slp$getRasterXSize()
nrows <- ds_slp$getRasterYSize()
for (row in 0:(nrows-1)) {
    rowdata <- ds_lcp$read(band=2,
                           xoff=0, yoff=row,
                           xsize=ncols, ysize=1,
                           out_xsize=ncols, out_ysize=1)
    rowslpp <- tan(rowdata*pi/180) * 100
    rowslpp[rowdata==-9999] <- -32767
    dim(rowslpp) <- c(1, ncols)
    ds_slp$write(band=1, xoff=0, yoff=row, xsize=ncols, ysize=1, rowslpp)
}

## min, max, mean, sd
ds_slp$getStatistics(band=1, approx_ok=FALSE, force=TRUE)

ds_slp$close()
ds_lcp$close()
```

`rasterToVRT` *Create a GDAL virtual raster*

Description

rasterToVRT() creates a virtual raster dataset (VRT format) derived from a source raster with options for virtual subsetting, virtually resampling the source data at a different pixel resolution, or applying a virtual kernel filter.

Usage

```
rasterToVRT(
  srcfile,
  relativeToVRT = FALSE,
  vrtfile = tempfile("tmprast", fileext = ".vrt"),
  resolution = NULL,
  subwindow = NULL,
  src_align = TRUE,
  resampling = "nearest",
  krnl = NULL,
  normalized = TRUE
)
```

Arguments

srcfile	Source raster filename.
relativeToVRT	Logical. Indicates whether the source filename should be interpreted as relative to the .vrt file (TRUE) or not relative to the .vrt file (FALSE, the default). If TRUE, the .vrt file is assumed to be in the same directory as srcfile and basename(srcfile) is used in the .vrt file. Use TRUE if the .vrt file will always be stored in the same directory with srcfile.
vrtfile	Output VRT filename.
resolution	A numeric vector of length two (xres, yres). The pixel size must be expressed in georeferenced units. Both must be positive values. The source pixel size is used if resolution is not specified.
subwindow	A numeric vector of length four (xmin, ymin, xmax, ymax). Selects subwindow of the source raster with corners given in georeferenced coordinates (in the source CRS). If not given, the upper left corner of the VRT will be the same as source, and the VRT extent will be the same or larger than source depending on resolution.
src_align	Logical. • TRUE: the upper left corner of the VRT extent will be set to the upper left corner of the source pixel that contains subwindow xmin, ymax. The VRT will be pixel-aligned with source if the VRT resolution is the same as the source pixel size, otherwise VRT extent will be the minimum rectangle that contains subwindow for the given pixel size. Often, src_align=TRUE when selecting a raster minimum bounding box for a vector polygon. • FALSE: the VRT upper left corner will be exactly subwindow xmin, ymax, and the VRT extent will be the minimum rectangle that contains subwindow for the given pixel size. If subwindow is not given, the source raster extent is

used in which case src_align=FALSE has no effect. Use src_align=FALSE to pixel-align two rasters of different sizes, i.e., when the intent is target alignment.

resampling — The resampling method to use if xsize, ysize of the VRT is different than the size of the underlying source rectangle (in number of pixels). The values allowed are nearest, bilinear, cubic, cubicspline, lanczos, average and mode (as character).

krnl — A filtering kernel specified as pixel coefficients. krnl is a array with dimensions (size, size), where size must be an odd number. krnl can also be given as a vector with length size x size. For example, a 3x3 average filter is given by:

```
krnl <- c(
0.11111, 0.11111, 0.11111,
0.11111, 0.11111, 0.11111,
0.11111, 0.11111, 0.11111)
```

A kernel cannot be applied to sub-sampled or over-sampled data.

normalized — Logical. Indicates whether the kernel is normalized. Defaults to TRUE.

Details

rasterToVRT() has similarities to the command-line utility gdalbuildvrt (https://gdal.org/programs/gdalbuildvrt.html) but is not a wrapper for it and does not build mosaics. rasterToVRT() is somewhat tailored for clipping and pixel-aligning various raster data in relation to vector polygon boundaries. It also supports VRT kernel filtering.

A VRT dataset is saved as a plain-text file with extension .vrt. This file contains a description of the dataset in an XML format. The description includes the source raster filename which can be a full path (relativeToVRT = FALSE) or relative path (relativeToVRT = TRUE). For relative path, rasterToVRT() assumes that the .vrt file will be in the same directory as the source file and uses basename(srcfile). The elements of the XML schema describe how the source data will be read, along with algorithms potentially applied and so forth. Documentation of the XML format for .vrt is at: https://gdal.org/drivers/raster/vrt.html.

Since .vrt is a small plain-text file it is fast to write and requires little storage space. Read performance is not degraded for certain simple operations (e.g., virtual clip without resampling). Reading will be slower for virtual resampling to a different pixel resolution or virtual kernel filtering since the operations are performed on-the-fly (but .vrt does not require the up front writing of a resampled or kernel-filtered raster to a regular format). VRT is sometimes useful as an intermediate raster in a series of processing steps, e.g., as a tempfile (the default).

GDAL VRT format has several capabilities and uses beyond those covered by rasterToVRT(). See the URLs above for a full discussion.

Value

Returns the VRT filename invisibly.

Note

Pixel alignment is specified in terms of the source raster pixels (i.e., srcfile of the virtual raster). The use case in mind is virtually clipping a raster to the bounding box of a vector polygon and

keeping pixels aligned with `srcfile` (`src_align = TRUE`). `src_align` would be set to `FALSE` if the intent is "target alignment". For example, if `subwindow` is the bounding box of another raster with a different layout, then also setting `resolution` to the pixel resolution of the target raster and `src_align = FALSE` will result in a virtual raster pixel-aligned with the target (i.e., pixels in the virtual raster are no longer aligned with its `srcfile`). Resampling defaults to `nearest` if not specified. Examples for both cases of `src_align` are given below.

`rasterToVRT()` assumes `srcfile` is a north-up raster. Requires package `xml2`.

See Also

`GDALRaster-class`, `bbox_from_wkt()`

`warp()` can write VRT for virtual reprojection

Examples

```
### resample

evt_file <- system.file("extdata/storml_evt.tif", package="gdalraster")
ds <- new(GDALRaster, evt_file, TRUE)
ds$res()
ds$bbox()
ds$close()

## use combine() with one input to get a table of pixel counts for
## the raster values
vat <- combine(evt_file)
print(vat[-1]) # drop the cmbid in this case
sum(vat$count)

## resample at 90-m resolution
## EVT is thematic vegetation type so use a majority value
vrt_file <- rasterToVRT(evt_file,
                        resolution=c(90,90),
                        resampling="mode")

## .vrt is a small xml file pointing to the source raster
file.size(vrt_file)

vat90m <- combine(vrt_file, var.names=c("evt90m"))
print(vat90m[-1])
sum(vat90m$count)

ds <- new(GDALRaster, vrt_file, TRUE)
ds$res()
ds$bbox()
ds$close()


### clip

evt_file <- system.file("extdata/storml_evt.tif", package="gdalraster")
```

```
ds_evt <- new(GDALRaster, evt_file, TRUE)
ds_evt$bbox()

## WKT string for a boundary within the EVT extent
bnd = "POLYGON ((324467.3 5104814.2, 323909.4 5104365.4, 323794.2
5103455.8, 324970.7 5102885.8, 326420.0 5103595.3, 326389.6 5104747.5,
325298.1 5104929.4, 325298.1 5104929.4, 324467.3 5104814.2))"

## src_align = TRUE
vrt_file <- rasterToVRT(evt_file,
                        subwindow = bbox_from_wkt(bnd),
                        src_align=TRUE)
ds_vrt <- new(GDALRaster, vrt_file, TRUE)

## VRT is a virtual clip, pixel-aligned with the EVT raster
bbox_from_wkt(bnd)
ds_vrt$bbox()
ds_vrt$res()

## src_align = FALSE
vrt_file <- rasterToVRT(evt_file,
                        subwindow = bbox_from_wkt(bnd),
                        src_align=FALSE)
ds_vrt_noalign <- new(GDALRaster, vrt_file, TRUE)

## VRT upper left corner (xmin, ymax) is exactly bnd xmin, ymax
ds_vrt_noalign$bbox()
ds_vrt_noalign$res()

ds_vrt$close()
ds_vrt_noalign$close()
ds_evt$close()


### subset and pixel align two rasters

## FARSITE landscape file for the Storm Lake area
lcp_file <- system.file("extdata/storm_lake.lcp", package="gdalraster")
ds_lcp <- new(GDALRaster, lcp_file, read_only=TRUE)

## Landsat band 5 file covering the Storm Lake area
b5_file <- system.file("extdata/sr_b5_20200829.tif", package="gdalraster")
ds_b5 <- new(GDALRaster, b5_file, read_only=TRUE)

ds_lcp$bbox()  # 323476.1 5101872.0  327766.1 5105082.0
ds_lcp$res()   # 30 30

ds_b5$bbox()   # 323400.9 5101815.8  327870.9 5105175.8
ds_b5$res()    # 30 30

## src_align = FALSE because we need target alignment in this case:
vrt_file <- rasterToVRT(b5_file,
                        resolution = ds_lcp$res(),
```

```
                        subwindow = ds_lcp$bbox(),
                        src_align = FALSE)
ds_b5vrt <- new(GDALRaster, vrt_file, TRUE)

ds_b5vrt$bbox() # 323476.1 5101872.0  327766.1 5105082.0
ds_b5vrt$res()  # 30 30

## read the the Landsat file pixel-aligned with the LCP file
## summarize band 5 reflectance where FBFM = 165
## LCP band 4 contains FBFM (a classification of fuel beds):
ds_lcp$getMetadata(band=4, domain="")

## verify Landsat nodata (0):
ds_b5vrt$getNoDataValue(band=1)
## will be read as NA and omitted from stats
rs <- new(RunningStats, na_rm=TRUE)

ncols <- ds_lcp$getRasterXSize()
nrows <- ds_lcp$getRasterYSize()
for (row in 0:(nrows-1)) {
    row_fbfm <- ds_lcp$read(band=4, xoff=0, yoff=row,
                            xsize=ncols, ysize=1,
                            out_xsize=ncols, out_ysize=1)
    row_b5 <- ds_b5vrt$read(band=1, xoff=0, yoff=row,
                            xsize=ncols, ysize=1,
                            out_xsize=ncols, out_ysize=1)
   rs$update(row_b5[row_fbfm == 165])
}
rs$get_count()
rs$get_mean()
rs$get_min()
rs$get_max()
rs$get_sum()
rs$get_var()
rs$get_sd()

ds_b5vrt$close()
ds_lcp$close()
ds_b5$close()
```

---

read_ds *Convenience wrapper for* GDALRaster$read()

Description

read_ds() will read from a raster dataset already opened with a GDALRaster object. By default, it will attempt to read the full raster extent from all bands at full resolution.

Usage

```
read_ds(
  ds,
  bands = NULL,
  xoff = 0,
  yoff = 0,
  xsize = ds$getRasterXSize(),
  ysize = ds$getRasterYSize(),
  out_xsize = xsize,
  out_ysize = ysize
)
```

Arguments

ds	An object of class GDALRaster in open state.
bands	Integer vector of band numbers to read. By default all bands will be read.
xoff	Integer. The pixel (column) offset to the top left corner of the raster region to be read (zero to start from the left side).
yoff	Integer. The line (row) offset to the top left corner of the raster region to be read (zero to start from the top).
xsize	Integer. The width in pixels of the region to be read.
ysize	Integer. The height in pixels of the region to be read.
out_xsize	Integer. The width of the output buffer into which the desired region will be read.
out_ysize	Integer. The height of the output buffer into which the desired region will be read.

Value

Returns a numeric or complex vector containing the values that were read. It is organized in left to right, top to bottom pixel order, interleaved by band. NA will be returned in place of the nodata value if the raster dataset has a nodata value defined for the band. Data are read as R integer type when possible for the raster data type (Byte, Int8, Int16, UInt16, Int32), otherwise as type double (UInt32, Float32, Float64).

Note

There is small overhead in calling read_ds() compared with calling GDALRaster$read() directly. This would only matter if calling the function repeatedly to read a raster in chunks. For the case of reading a large raster in many chunks (e.g., by row), it will be more efficient performance-wise to call GDALRaster$read() directly.

By default, this function will attempt to read the full raster into memory. It generally should not be called on large raster datasets using the default argument values. The memory size in bytes of the returned vector will be approximately (xsize * ysize * number of bands * 4) for data read as integer, and (xsize * ysize * number of bands * 8) for data read as double (plus small object overhead for the vector).

See Also

GDALRaster$read()

Examples

```
## read elevation, slope, and aspect from a multiband LCP file
lcp_file <- system.file("extdata/storm_lake.lcp", package="gdalraster")
ds <- new(GDALRaster, lcp_file, read_only=TRUE)

r <- read_ds(ds, bands=c(1:3))
typeof(r)
length(r)
object.size(r)

ds$close()
```

RunningStats-class *Class to calculate mean and variance in one pass*

Description

RunningStats computes summary statistics on a data stream efficiently. Mean and variance are calculated with Welford's online algorithm (https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance). The min, max, sum and count are also tracked. The input data values are not stored in memory, so this class can be used to compute statistics for very large data streams.

Arguments

na_rm	Logical. TRUE to remove NA from the input data or FALSE to retain NA (defaults to TRUE).

Value

An object of class RunningStats. A RunningStats object maintains the current minimum, maximum, mean, variance, sum and count of values that have been read from the stream. It can be updated repeatedly with new values (i.e., chunks of data read from the input stream), but its memory footprint is negligible. Class methods for updating with new values and retrieving current values of statistics are described in Details.

Usage

```
rs <- new(RunningStats, na_rm)

## Methods (see Details)
rs$update(newvalues)
rs$get_count()
rs$get_mean()
rs$get_min()
```

```
rs$get_max()
rs$get_sum()
rs$get_var()
rs$get_sd()
rs$reset()
```

Details

`new(RunningStats, na_rm)` Constructor. Returns an object of class `RunningStats`.

`$update(newvalues)` Updates the `RunningStats` object with a numeric vector of `newvalues` (i.e., a chunk of values from the data stream). No return value, called for side effects.

`$get_count()` Returns the count of values received from the data stream.

`$get_mean()` Returns the mean of values received from the data stream.

`$get_min()` Returns the minimum value received from the data stream.

`$get_max()` Returns the maximum value received from the data stream.

`$get_sum()` Returns the sum of values received from the data stream.

`$get_var()` Returns the variance of values from the data stream (denominator n - 1).

`$get_sd()` Returns the standard deviation of values from the data stream (denominator n - 1).

`$reset()` Clears the `RunningStats` object to its initialized state (count = 0). No return value, called for side effects.

Examples

```
set.seed(42)

rs <- new(RunningStats, na_rm=TRUE)
chunk <- runif(1000)
rs$update(chunk)
object.size(rs)

rs$get_count()
length(chunk)

rs$get_mean()
mean(chunk)

rs$get_min()
min(chunk)

rs$get_max()
max(chunk)

rs$get_var()
var(chunk)

rs$get_sd()
sd(chunk)
```

```
## 10^9 values read in 10,000 chunks
## should take under 2 minutes on typical PC hardware
for (i in 1:1e4) {
  chunk <- runif(1e5)
  rs$update(chunk)
}
rs$get_count()
rs$get_mean()
rs$get_var()

object.size(rs)
```

`set_config_option` *Set GDAL configuration option*

Description

`set_config_option()` sets a GDAL runtime configuration option. Configuration options are essentially global variables the user can set. They are used to alter the default behavior of certain raster format drivers, and in some cases the GDAL core. For a full description and listing of available options see https://gdal.org/user/configoptions.html.

Usage

```
set_config_option(key, value)
```

Arguments

`key`	Character name of a configuration option.
`value`	Character value to set for the option. `value = ""` (empty string) will unset a value previously set by `set_config_option()`.

Value

No return value, called for side effects.

See Also

`get_config_option()`

Examples

```
set_config_option("GDAL_CACHEMAX", "64")
get_config_option("GDAL_CACHEMAX")
## unset:
set_config_option("GDAL_CACHEMAX", "")
```

srs_is_geographic *Check if WKT definition is a geographic coordinate system*

Description

`srs_is_geographic()` will attempt to import the given WKT string as a spatial reference system, and returns `TRUE` if the root is a GEOGCS node. This is a wrapper for `OSRIsGeographic()` in the GDAL Spatial Reference System C API.

Usage

```
srs_is_geographic(srs)
```

Arguments

`srs`	Character OGC WKT string for a spatial reference system

Value

Logical. `TRUE` if `srs` is geographic, otherwise `FALSE`

See Also

`srs_is_projected()`, `srs_is_same()`

Examples

```
srs_is_geographic(epsg_to_wkt(5070))
srs_is_geographic(srs_to_wkt("WGS84"))
```

srs_is_projected *Check if WKT definition is a projected coordinate system*

Description

`srs_is_projected()` will attempt to import the given WKT string as a spatial reference system (SRS), and returns `TRUE` if the SRS contains a PROJCS node indicating a it is a projected coordinate system. This is a wrapper for `OSRIsProjected()` in the GDAL Spatial Reference System C API.

Usage

```
srs_is_projected(srs)
```

Arguments

`srs`	Character OGC WKT string for a spatial reference system

Value

Logical. TRUE if srs is projected, otherwise FALSE

See Also

srs_is_geographic(), srs_is_same()

Examples

```
srs_is_projected(epsg_to_wkt(5070))
srs_is_projected(srs_to_wkt("WGS84"))
```

---

srs_is_same *Do these two spatial references describe the same system?*

Description

srs_is_same() returns TRUE if these two spatial references describe the same system. This is a wrapper for OSRIsSame() in the GDAL Spatial Reference System C API.

Usage

```
srs_is_same(srs1, srs2)
```

Arguments

srs1	Character OGC WKT string for a spatial reference system
srs2	Character OGC WKT string for a spatial reference system

Value

Logical. TRUE if these two spatial references describe the same system, otherwise FALSE.

See Also

srs_is_geographic(), srs_is_projected()

Examples

```
elev_file <- system.file("extdata/storml_elev.tif", package="gdalraster")
ds <- new(GDALRaster, elev_file, TRUE)
srs_is_same(ds$getProjectionRef(), epsg_to_wkt(26912))
srs_is_same(ds$getProjectionRef(), epsg_to_wkt(5070))
ds$close()
```

srs_to_wkt *Convert spatial reference definition to OGC Well Known Text*

Description

`srs_to_wkt()` converts a spatial reference system (SRS) definition in various text formats to WKT. The function will examine the input SRS, try to deduce the format, and then export it to WKT.

Usage

```
srs_to_wkt(srs, pretty = FALSE)
```

Arguments

`srs`	Character string containing an SRS definition in various formats (see Details).
`pretty`	Logical. `TRUE` to return a nicely formatted WKT string for display to a person. `FALSE` for a regular WKT string (the default).

Details

This is a wrapper for `OSRSetFromUserInput()` in the GDAL Spatial Reference System C API with output to WKT. The input SRS may take the following forms:

- `WKT` - to convert WKT versions (see below)
- `EPSG:n` - EPSG code n
- `AUTO:proj_id,unit_id,lon0,lat0` - WMS auto projections
- `urn:ogc:def:crs:EPSG::n` - OGC URNs
- PROJ.4 definitions
- `filename` - file read for WKT, XML or PROJ.4 definition
- well known name such as `NAD27`, `NAD83`, `WGS84` or `WGS72`
- `IGNF:xxxx`, `ESRI:xxxx` - definitions from the PROJ database
- PROJJSON (PROJ >= 6.2)

This function is intended to be flexible, but by its nature it is imprecise as it must guess information about the format intended. `epsg_to_wkt()` could be used instead for EPSG codes.

As of GDAL 3.0, the default format for WKT export is OGC WKT 1. The WKT version can be overridden by using the `OSR_WKT_FORMAT` configuration option (see `set_config_option()`). Valid values are one of: `SFSQL`, `WKT1_SIMPLE`, `WKT1`, `WKT1_GDAL`, `WKT1_ESRI`, `WKT2_2015`, `WKT2_2018`, `WKT2`, `DEFAULT`. If `SFSQL`, a WKT1 string without AXIS, TOWGS84, AUTHORITY or EXTENSION node is returned. If `WKT1_SIMPLE`, a WKT1 string without AXIS, AUTHORITY or EXTENSION node is returned. `WKT1` is an alias of `WKT1_GDAL`. `WKT2` will default to the latest revision implemented (currently `WKT2_2018`). `WKT2_2019` can be used as an alias of `WKT2_2018` since GDAL 3.2

Value

Character string containing OGC WKT.

See Also

epsg_to_wkt()

Examples

```
srs_to_wkt("NAD83")
writeLines(srs_to_wkt("NAD83", pretty=TRUE))
set_config_option("OSR_WKT_FORMAT", "WKT2")
writeLines(srs_to_wkt("NAD83", pretty=TRUE))
set_config_option("OSR_WKT_FORMAT", "")
```

---

transform_xy *Transform geospatial x/y coordinates*

Description

transform_xy() transforms geospatial x/y coordinates to a new projection.

Usage

```
transform_xy(pts, srs_from, srs_to)
```

Arguments

pts	A two-column data frame or numeric matrix containing geospatial x/y coordinates
srs_from	Character string in OGC WKT format specifying the spatial reference system for pts.
srs_to	Character string in OGC WKT format specifying the output spatial reference system.

Value

Numeric array of geospatial x/y coordinates in the projection specified by srs_to.

See Also

epsg_to_wkt(), srs_to_wkt(), inv_project()

Examples

```
pt_file <- system.file("extdata/storml_pts.csv", package="gdalraster")
pts <- read.csv(pt_file)
print(pts)
## id, x, y in NAD83 / UTM zone 12N
## transform to NAD83 / CONUS Albers
transform_xy(pts = pts[,-1],
             srs_from = epsg_to_wkt(26912),
             srs_to = epsg_to_wkt(5070))
```

---

warp *Raster reprojection*

Description

warp() is a wrapper for the gdalwarp command-line utility. See https://gdal.org/programs/gdalwarp.html for details.

Usage

```
warp(src_files, dst_filename, t_srs, cl_arg = NULL)
```

Arguments

src_files	Character vector of source file(s) to be reprojected.
dst_filename	Filename of the output raster.
t_srs	Character. Target spatial reference system. Usually an EPSG code ("EPSG:#####") or a well known text (WKT) SRS definition.
cl_arg	Optional character vector of command-line arguments to gdalwarp in addition to -t_srs.

Value

Logical indicating success (invisible TRUE). An error is raised if the operation fails.

See Also

GDALRaster-class, srs_to_wkt()

Examples

```
## reproject the elevation raster to NAD83 / CONUS Albers (EPSG:5070)
elev_file <- system.file("extdata/storml_elev.tif", package="gdalraster")

## command-line arguments for gdalwarp
## resample to 90-m resolution using average and keep pixels aligned:
args = c("-tr", "90", "90", "-r", "average", "-tap")
## output to Erdas Imagine format (HFA), creation option for compression:
```

```
args = c(args, "-of", "HFA", "-co", "COMPRESSED=YES")

alb83_file <- paste0(tempdir(), "/", "storml_elev_alb83.img")
warp(elev_file, alb83_file, t_srs="EPSG:5070", cl_arg = args)

ds <- new(GDALRaster, alb83_file, read_only=TRUE)
ds$getDriverLongName()
ds$getProjectionRef()
ds$res()
ds$getStatistics(band=1, approx_ok=FALSE, force=TRUE)
ds$close()
```

Index