

Package ‘gdalraster’

November 29, 2023

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

Version 1.7.0

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 several '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 exposed C++ class encapsulating a 'GDALDataset' and its associated 'GDALRasterBand' objects, plus 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 'combine()' to identify and count unique pixel combinations across multiple input layers; raster 'calc()' to evaluate an R expression on a stack of layers with pixel coordinates available as variables in the expression; 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, xml2

LinkingTo Rcpp

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

NeedsCompilation yes

SystemRequirements GDAL (>= 2.4.0, built against GEOS), PROJ (>= 4.8.0), sqlite3

Encoding UTF-8

RoxygenNote 7.2.3

VignetteBuilder knitr

Config/testthat/edition 3

Author Chris Toney [aut, cre] (R interface/additional functionality),
 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 in
 src/cmb_table.h),
 Peter Dimov [ctb, cph] (Boost combine hashes method in src/cmb_table.h)

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

Repository CRAN

Date/Publication 2023-11-29 09:00:02 UTC

R topics documented:

gdalraster-package	3
bandCopyWholeRaster	5
bbox_from_wkt	6
bbox_intersect	7
bbox_to_wkt	8
buildRAT	9
buildVRT	12
calc	14
CmbTable-class	18
combine	20
copyDatasetFiles	22
create	23
createColorRamp	24
createCopy	25
DEFAULT_DEM_PROC	26
DEFAULT_NODATA	27
deleteDataset	28
dem_proc	29
displayRAT	30
epsg_to_wkt	30
fillNodata	31
GDALRaster-class	32
gdal_formats	43
gdal_version	43

getCreationOptions	44
get_cache_used	45
get_config_option	45
get_pixel_line	46
g_buffer	47
has_geos	47
inv_geotransform	48
inv_project	49
plot_raster	50
polygonize	53
proj_networking	56
proj_search_paths	57
proj_version	57
rasterFromRaster	58
rasterize	60
rasterToVRT	62
read_ds	67
renameDataset	69
RunningStats-class	70
set_config_option	72
sieveFilter	73
srs_is_geographic	75
srs_is_projected	75
srs_is_same	76
srs_to_wkt	77
transform_xy	78
translate	79
warp	80
Index	84

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 class GDALRaster and several related stand-alone functions:

- **GDALRaster-class** is an exposed C++ class that allows opening a raster dataset and calling methods on the GDALDataset, GDALDriver and GDALRasterBand objects in the underlying API (e.g., get/set parameters, read/write pixel data).

- raster creation: `create()`, `createCopy()`, `rasterFromRaster()`, `translate()`, `getCreationOptions()`
- data management: `copyDatasetFiles()`, `deleteDataset()`, `renameDataset()`, `bandCopyWholeRaster()`
- virtual raster: `buildVRT()`, `rasterToVRT()`
- reproject/resample/crop/mosaic: `warp()`
- algorithms: `dem_proc()`, `fillNodata()`, `polygonize()`, `rasterize()`, `sieveFilter()`, `GDALRaster$getChecksum()`
- raster attribute tables: `buildRAT()`, `displayRAT()`, `GDALRaster$getDefaultRAT()`, `GDALRaster$setDefaultRAT()`
- 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()`, `g_buffer()`, `has_geos()`
- GDAL configuration: `gdal_version()`, `gdal_formats()`, `get_cache_used()`, `get_config_option()`, `set_config_option()`
- PROJ configuration: `proj_version()`, `proj_search_paths()`, `proj_networking()`

Additional functionality includes:

- `RunningStats-class` calculates mean and variance in one pass. The min, max, sum, and count are also tracked (efficient summary statistics on large data streams).
- `CmbTable-class` 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 `GDALRaster-class` and several wrapper functions borrows from the GDAL API documentation, (c) 1998-2023, Frank Warmerdam, Even Rouault, and others, [MIT license](#).

Sample datasets included with the package are used in examples throughout the documentation. The sample data include **LANDFIRE** raster layers describing terrain, vegetation and wildland fuels (LF 2020 version), Landsat C2 Analysis Ready Data from **USGS Earth Explorer**, and Monitoring Trends in Burn Severity (**MTBS**) fire perimeters from 1984-2022. Metadata for the sample datasets are in `inst/extdata/metadata.zip`.

`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 is by: Frank Warmerdam, Even Rouault and others
(see <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

bandCopyWholeRaster *Copy a whole raster band efficiently*

Description

bandCopyWholeRaster() copies the complete raster contents of one band to another similarly configured band. The source and destination bands must have the same xsize and ysize. The bands do not have to have the same data type. It implements efficient copying, in particular "chunking" the copy in substantial blocks. This is a wrapper for GDALRasterBandCopyWholeRaster() in the GDAL API.

Usage

```
bandCopyWholeRaster(  
  src_filename,  
  src_band,  
  dst_filename,  
  dst_band,  
  options = NULL  
)
```

Arguments

src_filename	Filename of the source raster.
src_band	Band number in the source raster to be copied.
dst_filename	Filename of the destination raster.
dst_band	Band number in the destination raster to copy into.
options	Optional list of transfer hints in a vector of "NAME=VALUE" pairs. The currently supported options are:

- "COMPRESSED=YES" to force alignment on target dataset block sizes to achieve best compression.
- "SKIP_HOLES=YES" to skip chunks that contain only empty blocks. Empty blocks are blocks that are generally not physically present in the file, and when read through GDAL, contain only pixels whose value is the nodata value when it is set, or whose value is 0 when the nodata value is not set. The query is done in an efficient way without reading the actual pixel values (if implemented by the raster format driver, otherwise will not be skipped).

Value

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

See Also

[GDALRaster-class](#), [create\(\)](#), [createCopy\(\)](#), [rasterFromRaster\(\)](#)

Examples

```
## copy Landsat data from a single-band file to a new multi-band image
b5_file <- system.file("extdata/sr_b5_20200829.tif", package="gdalraster")
dst_file <- paste0(tempdir(), "/", "sr_multi.tif")
rasterFromRaster(b5_file, dst_file, nbands=7, init=0)
opt <- c("COMPRESSED=YES", "SKIP_HOLES=YES")
bandCopyWholeRaster(b5_file, 1, dst_file, 5, options=opt)
ds <- new(GDALRaster, dst_file, read_only=TRUE)
ds$getStatistics(band=5, approx_ok=FALSE, force=TRUE)
ds$close()
```

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, extend_x = 0, extend_y = 0)
```

Arguments

wkt	Character. OGC WKT string for a simple feature 2D geometry.
extend_x	Numeric scalar. Distance to extend the output bounding box in both directions along the x-axis (results in xmin = bbox[1] - extend_x, xmax = bbox[3] + extend_x).
extend_y	Numeric scalar. Distance to extend the output bounding box in both directions along the y-axis (results in ymin = bbox[2] - extend_y, ymax = bbox[4] + extend_y).

Value

Numeric vector of length four containing the xmin, ymin, xmax, ymax of the geometry specified by wkt (possibly extended by values in extend_x, extend_y).

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, 100, 100)
```

bbox_intersect	<i>Bounding box intersection / union</i>
----------------	--

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/storm1_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. Requires GDAL built with the GEOS library.

Usage

```
bbox_to_wkt(bbox, extend_x = 0, extend_y = 0)
```

Arguments

<code>bbox</code>	Numeric vector of length four containing <code>xmin</code> , <code>ymin</code> , <code>xmax</code> , <code>ymax</code> .
<code>extend_x</code>	Numeric scalar. Distance in units of <code>bbox</code> to extend the rectangle in both directions along the x-axis (results in <code>xmin = bbox[1] - extend_x</code> , <code>xmax = bbox[3] + extend_x</code>).
<code>extend_y</code>	Numeric scalar. Distance in units of <code>bbox</code> to extend the rectangle in both directions along the y-axis (results in <code>ymin = bbox[2] - extend_y</code> , <code>ymax = bbox[4] + extend_y</code>).

Value

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

See Also

[bbox_from_wkt\(\)](#), [g_buffer\(\)](#)

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()
```

 buildRAT

Build a GDAL Raster Attribute Table as data frame

Description

`buildRAT()` reads all pixels of an input raster to obtain the set of unique values and their counts. The result is returned as a data frame suitable for use with the class method `GDALRaster$setDefaultRAT()`. The returned data frame might be further modified before setting as a Raster Attribute Table in a dataset, for example, by adding columns containing class names, color values, or other information (see Details). An optional input data frame containing such attributes may be given, in which case `buildRAT()` will attempt to join the additional columns and automatically assign the appropriate metadata on the output data frame (i.e., assign R attributes on the data frame and its columns that define usage in a GDAL Raster Attribute Table).

Usage

```
buildRAT(
  raster,
  band = 1L,
  col_names = c("VALUE", "COUNT"),
  table_type = "athematic",
  na_value = NULL,
  join_df = NULL
)
```

Arguments

<code>raster</code>	Either a <code>GDALRaster</code> object, or a character string containing the file name of a raster dataset to open.
<code>band</code>	Integer scalar, band number to read (default 1L).
<code>col_names</code>	Character vector of length two containing names to use for column 1 (pixel values) and column 2 (pixel counts) in the output data frame (defaults are <code>c("VALUE", "COUNT")</code>).
<code>table_type</code>	Character string describing the type of the attribute table. One of either <code>"thematic"</code> , or <code>"athematic"</code> for continuous data (the default).

na_value	Numeric scalar. If the set of unique pixel values has an NA, it will be recoded to na_value in the returned data frame. If NULL (the default), NA will not be recoded.
join_df	Optional data frame for joining additional attributes. Must have a column of unique values with the same name as col_names[1] ("VALUE" by default).

Details

A GDAL Raster Attribute Table (or RAT) provides attribute information about pixel values. Raster attribute tables can be used to represent histograms, color tables, and classification information. Each row in the table applies to either a single pixel value or a range of values, and might have attributes such as the histogram count for that value (or range), the color that pixels of that value (or range) should be displayed, names of classes, or various other information.

Each column in a raster attribute table has a name, a type (integer, double, or string), and a GDALRATfieldUsage. The usage distinguishes columns with particular understood purposes (such as color, histogram count, class name), and columns that have other purposes not understood by the library (long labels, ancillary attributes, etc).

In the general case, each row has a field indicating the minimum pixel value falling into that category, and a field indicating the maximum pixel value. In the GDAL API, these are indicated with usage values of GFU_Min and GFU_Max. In the common case where each row is a discrete pixel value, a single column with usage GFU_MinMax would be used instead. In R, the table is represented as a data frame with column attribute "GFU" containing the field usage as a string, e.g., "Max", "Min" or "MinMax" (case-sensitive). The full set of possible field usage descriptors is:

GFU attr	GDAL enum	Description
"Generic"	GFU_Generic	General purpose field
"PixelCount"	GFU_PixelCount	Histogram pixel count
"Name"	GFU_Name	Class name
"Min"	GFU_Min	Class range minimum
"Max"	GFU_Max	Class range maximum
"MinMax"	GFU_MinMax	Class value (min=max)
"Red"	GFU_Red	Red class color (0-255)
"Green"	GFU_Green	Green class color (0-255)
"Blue"	GFU_Blue	Blue class color (0-255)
"Alpha"	GFU_Alpha	Alpha transparency (0-255)
"RedMin"	GFU_RedMin	Color range red minimum
"GreenMin"	GFU_GreenMin	Color range green minimum
"BlueMin"	GFU_BlueMin	Color range blue minimum
"AlphaMin"	GFU_AlphaMin	Color range alpha minimum
"RedMax"	GFU_RedMax	Color range red maximum
"GreenMax"	GFU_GreenMax	Color range green maximum
"BlueMax"	GFU_BlueMax	Color range blue maximum
"AlphaMax"	GFU_AlphaMax	Color range alpha maximum

buildRAT() assigns GFU "MinMax" on the column of pixel values (named "VALUE" by default) and GFU "PixelCount" on the column of counts (named "COUNT" by default). If join_df is given, the additional columns that result from joining will have GFU assigned automatically based on the column names (*ignoring case*). First, the additional column names are checked for containing the

string "name" (e.g., "classname", "TypeName", "EVT_NAME", etc). The first matching column (if any) will be assigned a GFU of "Name" (=GFU_Name, the field usage descriptor for class names). Next, columns named "R" or "Red" will be assigned GFU "Red", columns named "G" or "Green" will be assigned GFU "Green", columns named "B" or "Blue" will be assigned GFU "Blue", and columns named "A" or "Alpha" will be assigned GFU "Alpha". Finally, any remaining columns that have not been assigned a GFU will be assigned "Generic".

In a variation of RAT, all the categories are of equal size and regularly spaced, and the categorization can be determined by knowing the value at which the categories start and the size of a category. This is called "Linear Binning" and the information is kept specially on the raster attribute table as a whole. In R, a RAT that uses linear binning would have the following attributes set on the data frame: attribute "Row0Min" = the numeric lower bound (pixel value) of the first category, and attribute "BinSize" = the numeric width of each category (in pixel value units). buildRAT() does not create tables with linear binning, but one could be created manually based on the specifications above, and applied to a raster with the class method GDALRaster\$setDefaultRAT().

A raster attribute table is thematic or athematic (continuous). In R, this is defined by an attribute on the data frame named "GDALRATTableType" with value of either "thematic" or "athematic".

Value

A data frame with at least two columns containing the set of unique pixel values and their counts. These columns have attribute "GFU" set to "MinMax" for the values, and "PixelCount" for the counts. If join_df is given, the returned data frame will have additional columns that result from merge(). The "GFU" attribute of the additional columns will be assigned automatically based on the column names (*case-insensitive* matching, see Details). The returned data frame has attribute "GDALRATTableType" set to table_type.

Note

The full raster will be scanned.

If na_value is not specified, then an NA pixel value (if present) will not be recoded in the output data frame. This may have implications if joining to other data (NA will not match), or when using the returned data frame to set a default RAT on a dataset (NA will be interpreted as the value that R uses internally to represent it for the type, e.g., -2147483648 for NA_integer_). In some cases, removing the row in the output data frame with value NA, rather than recoding, may be desirable (i.e., by removing manually or by side effect of joining via merge(), for example). Users should consider what is appropriate for a particular case.

See Also

[GDALRaster\\$getDefaultRAT\(\)](#), [GDALRaster\\$setDefaultRAT\(\)](#), [displayRAT\(\)](#)
[vignette\("raster-attribute-tables"\)](#)

Examples

```
evt_file <- system.file("extdata/storml_evt.tif", package="gdalraster")
# make a copy to modify
f <- paste0(tempdir(), "/", "storml_evt_tmp.tif")
file.copy(evt_file, f)
```

```

ds <- new(GDALRaster, f, read_only=FALSE)
ds$getDefaultRAT(band=1) # NULL

# get the full attribute table for LANDFIRE EVT from the CSV file
evt_csv <- system.file("extdata/LF20_EVT_220.csv", package="gdalraster")
evt_df <- read.csv(evt_csv)
nrow(evt_df)
head(evt_df)
evt_df <- evt_df[,1:7]

tbl <- buildRAT(ds,
               table_type = "thematic",
               na_value = -9999,
               join_df = evt_df)

nrow(tbl)
head(tbl)

# attributes on the data frame and its columns define usage in a GDAL RAT
attributes(tbl)
attributes(tbl$VALUE)
attributes(tbl$COUNT)
attributes(tbl$EVT_NAME)
attributes(tbl$EVT_LF)
attributes(tbl$EVT_PHYS)
attributes(tbl$R)
attributes(tbl$G)
attributes(tbl$B)

ds$setDefaultRAT(band=1, tbl)
ds$flushCache()

tbl2 <- ds$getDefaultRAT(band=1)
nrow(tbl2)
head(tbl2)

ds$close()

# Display
evt_gt <- displayRAT(tbl2, title = "Raster Attribute Table for Storm Lake EVT")
class(evt_gt) # an object of class "gt_tbl" from package gt
# To show the table:
# evt_gt
# or simply call `displayRAT()` as above but without assignment
# `vignette("raster-attribute-tables")` has example output

```

Description

buildVRT() is a wrapper of the gdalbuildvrt command-line utility for building a VRT (Virtual Dataset) that is a mosaic of the list of input GDAL datasets (see <https://gdal.org/programs/gdalbuildvrt.html>).

Usage

```
buildVRT(vrt_filename, input_rasters, cl_arg = NULL)
```

Arguments

vrt_filename Character string. Filename of the output VRT.
input_rasters Character vector of input raster filenames.
cl_arg Optional character vector of command-line arguments to gdalbuildvrt.

Details

Several command-line options are described in the GDAL documentation at the URL above. By default, the input files are considered as tiles of a larger mosaic and the VRT file has as many bands as one of the input files. Alternatively, the `-separate` argument can be used to put each input raster into a separate band in the VRT dataset.

Some amount of checks are done to assure that all files that will be put in the resulting VRT have similar characteristics: number of bands, projection, color interpretation.... If not, files that do not match the common characteristics will be skipped. (This is true in the default mode for virtual mosaicing, and not when using the `-separate` option).

In a virtual mosaic, if there is spatial overlap between input rasters then the order of files appearing in the list of sources matter: files that are listed at the end are the ones from which the data will be fetched. Note that nodata will be taken into account to potentially fetch data from less priority datasets.

Value

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

See Also

[rasterToVRT\(\)](#)

Examples

```
# build a virtual 3-band RGB raster from individual Landsat band files
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)
vrt_file <- paste0(tempdir(), "/", "stornl_b6_b5_b4.vrt")
buildVRT(vrt_file, band_files, cl_arg = "-separate")
ds <- new(GDALRaster, vrt_file, read_only=TRUE)
ds$getRasterCount()
```

```
plot_raster(ds, nbands=3, main="Landsat 6-5-4 (vegetative analysis)")
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).

<code>out_band</code>	Integer band number in <code>dstfile</code> for writing output.
<code>options</code>	Optional list of format-specific creation options in a vector of "NAME=VALUE" pairs (e.g., <code>options = c("COMPRESS=LZW")</code> to set LZW compression during creation of a GTiff file).
<code>nodata_value</code>	Numeric value to assign if <code>expr</code> returns NA.
<code>setRasterNodataValue</code>	Logical. TRUE will attempt to set the raster format nodata value to <code>nodata_value</code> , or FALSE not to set a raster nodata value.
<code>usePixelLonLat</code>	Logical. If TRUE, <code>pixelX</code> and <code>pixelY</code> will be inverse projected to geographic coordinates and available as <code>pixelLon</code> and <code>pixelLat</code> in <code>expr</code> (adds computation time).
<code>write_mode</code>	Character. Name of the file write mode for output. One of: <ul style="list-style-type: none"> • <code>safe</code> - execution stops if <code>dstfile</code> already exists (no output written) • <code>overwrite</code> - if <code>dstfile</code> exists it will be overwritten with a new file • <code>update</code> - if <code>dstfile</code> exists, will attempt to open in update mode and write output to <code>out_band</code>

Details

The variables in `expr` are vectors of length `raster xsize` (row vectors of the input raster layer(s)). 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)
tbl <- combine(rasterfiles, var.names, bands)
nrow(tbl)
tbl_subset <- subset(tbl, SLP >= 40 & FBFM %in% c(101,102))
print(tbl_subset)      # twelve combinations meet the criteria
sum(tbl_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 output
rasterfiles <- c(tif_file, tif_file)
tbl <- combine(rasterfiles, var.names, bands)
tbl_subset <- subset(tbl, SLP >= 40 & FBFM %in% c(101,102))
print(tbl_subset)
sum(tbl_subset$count)

# if LCP file format is needed:
# createCopy("LCP", "storml_edited.lcp", tif_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. CmbTable is a C++ class exposed directly to R (via RCPP_EXPOSED_CLASS). Methods of the class are accessed in R using the \$ operator.

Usage

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

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

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.

`$updateFromMatrix(int_cmb, incr)` This method is the same as `$update()` but for a numeric matrix of integer combinations `int_cmb` (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_cmb`), incrementing count by `incr` for existing combinations, or inserting new combinations with count set to `incr`. Returns a numeric vector of length `ncol(int_cmb)` containing the IDs assigned to the combinations.

`$updateFromMatrixByRow(int_cmb, incr)` This method is the same as `$updateFromMatrix()` above except the integer combinations are in rows of the matrix `int_cmb` (columns are the variables). The method calls `$update()` on each combination (each row of `int_cmb`), incrementing count by `incr` for existing combinations, or inserting new combinations with count set to `incr`. Returns a numeric vector of length `nrow(int_cmb)` 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.

`$asMatrix()` Returns the `CmbTable` as a matrix with column 1 ("cmbid") containing the unique combination IDs, column 2 ("count") containing the counts of occurrences, and columns 3:`keyLen+2` 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)
rownames(m) <- c("layer1", "layer2", "layer3")
print(m)
cmb <- new(CmbTable, 3, rownames(m))
cmb$updateFromMatrix(m, 1)
cmb$asDataFrame()
cmb$update(c(4,5,6), 1)
cmb$update(c(1,3,5), 1)
cmb$asDataFrame()
```

```
# same as above but matrix arranged with integer combinations in the rows
m <- matrix(c(1,2,3,1,2,3,4,5,6,1,3,2,4,5,6,1,1,1), 6, 3, byrow=TRUE)
colnames(m) <- c("v1", "v2", "v3")
print(m)
cmb <- new(CmbTable, 3, colnames(m))
cmb$updateFromMatrixByRow(m, 1)
cmb$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

<code>rasterfiles</code>	Character vector of raster filenames to combine.
<code>var.names</code>	Character vector of <code>length(rasterfiles)</code> containing variable names for each raster layer. Defaults will be assigned if <code>var.names</code> are omitted.
<code>bands</code>	Numeric vector of <code>length(rasterfiles)</code> containing the band number to use for each raster in <code>rasterfiles</code> . Band 1 will be used for each input raster if bands are not specified.
<code>dstfile</code>	Character. Optional output raster filename for writing the per-pixel combination IDs. The output raster will be created (and overwritten if it already exists).
<code>fmt</code>	Character. Output raster format name (e.g., "GTiff" or "HFA").
<code>dtName</code>	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").
<code>options</code>	Optional list of format-specific creation options in a vector of "NAME=VALUE" pairs (e.g., <code>options = c("COMPRESS=LZW")</code> 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).

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\(\)](#)

[buildRAT\(\)](#) to compute a table of the unique pixel values and their counts for a single raster layer

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")
tbl <- combine(rasterfiles, var.names)
nrow(tbl)
tbl <- tbl[order(-tbl$count),]
head(tbl, 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")
opt <- c("COMPRESS=LZW")
tbl <- combine(rasterfiles, var.names, bands, cmb_file, options = opt)
head(tbl)
ds <- new(GDALRaster, cmb_file, TRUE)
ds$info()
ds$close()
```

copyDatasetFiles	<i>Copy the files of a dataset</i>
------------------	------------------------------------

Description

copyDatasetFiles() copies all the files associated with a dataset. Wrapper for GDALCopyDatasetFiles() in the GDAL API.

Usage

```
copyDatasetFiles(new_filename, old_filename, format = "")
```

Arguments

new_filename	New name for the dataset (copied to).
old_filename	Old name for the dataset (copied from).
format	Raster format short name (e.g., "GTiff"). If set to empty string "" (the default), will attempt to guess the raster format from old_filename.

Value

Logical TRUE if no error or FALSE on failure.

Note

If format is set to an empty string "" (the default) then the function will try to identify the driver from old_filename. This is done internally in GDAL by invoking the Identify method of each registered GDALDriver in turn. The first driver that successfully identifies the file name will be returned. An error is raised if a format cannot be determined from the passed file name.

See Also

[GDALRaster-class](#), [create\(\)](#), [createCopy\(\)](#), [deleteDataset\(\)](#), [renameDataset\(\)](#)

Examples

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

lcp_tmp <- paste0(tempdir(), "/", "storm_lake_copy.lcp")
copyDatasetFiles(lcp_tmp, lcp_file)
ds_copy <- new(GDALRaster, lcp_tmp, read_only=TRUE)
ds_copy$getFileList()
ds_copy$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").
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\(\)](#), [getCreationOptions\(\)](#)

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$setProjection(eps_g_to_wkt(26912))
gt <- c(323476.1, 30, 0, 5105082.0, 0, -30)
ds$setGeoTransform(gt)
ds$setNoDataValue(band = 1, -9999)
ds$fillRaster(band = 1, -9999, 0)
## ...
## close the dataset when done
ds$close()
```

createColorRamp	<i>Create a color ramp</i>
-----------------	----------------------------

Description

createColorRamp() is a wrapper for GDALCreateColorRamp() in the GDAL API. It automatically creates a color ramp from one color entry to another. Output is an integer matrix in color table format for use with GDALRaster\$setColorTable().

Usage

```
createColorRamp(
  start_index,
  start_color,
  end_index,
  end_color,
  palette_interp = "RGB"
)
```

Arguments

start_index	Integer start index (raster value).
start_color	Integer vector of length three or four. A color entry value to start the ramp (e.g., RGB values).
end_index	Integer end index (raster value).
end_color	Integer vector of length three or four. A color entry value to end the ramp (e.g., RGB values).
palette_interp	One of "Gray", "RGB" (the default), "CMYK" or "HLS" describing interpretation of start_color and end_color values (see GDAL Color Table).

Value

Integer matrix with five columns containing the color ramp from start_index to end_index, with raster index values in column 1 and color entries in columns 2:5).

Note

createColorRamp() could be called several times, using rbind() to combine multiple ramps into the same color table. Possible duplicate rows in the resulting table are not a problem when used in GDALRaster\$setColorTable() (i.e., when end_color of one ramp is the same as start_color of the next ramp).

See Also

[GDALRaster\\$getColorTable\(\)](#), [GDALRaster\\$getPaletteInterp\(\)](#)

Examples

```

# create a color ramp for tree canopy cover percent
# band 5 of an LCP file contains canopy cover
lcp_file <- system.file("extdata/storm_lake.lcp", package="gdalraster")
ds <- new(GDALRaster, lcp_file, read_only=TRUE)
ds$getDescription(band=5)
ds$getMetadata(band=5, domain="")
ds$close()

# create a GTiff file with Byte data type for the canopy cover band
# recode nodata -9999 to 255
tcc_file <- calc(expr = "ifelse(CANCOV == -9999, 255, CANCOV)",
  rasterfiles = lcp_file,
  bands = 5,
  var.names = "CANCOV",
  fmt = "GTiff",
  dtName = "Byte",
  nodata_value = 255,
  setRasterNodataValue = TRUE)

ds_tcc <- new(GDALRaster, tcc_file, read_only=FALSE)

# create a color ramp from 0 to 100 and set as the color table
colors <- createColorRamp(start_index = 0,
  start_color = c(211, 211, 211),
  end_index = 100,
  end_color = c(0, 100, 0))

print(colors)
ds_tcc$setColorTable(band=1, col_tbl=colors, palette_interp="RGB")
ds_tcc$setRasterColorInterp(band=1, col_interp="Palette")

# close and re-open the dataset in read_only mode
ds_tcc$open(read_only=TRUE)

plot_raster(ds_tcc, interpolate=FALSE, legend=TRUE,
  main="Storm Lake Tree Canopy Cover (%)")
ds_tcc$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

<code>format</code>	Format short name for the output raster (e.g., "GTiff" or "HFA").
<code>dst_filename</code>	Filename to create.
<code>src_filename</code>	Filename of source raster.
<code>strict</code>	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.
<code>options</code>	Optional list of format-specific creation options in a vector of "NAME=VALUE" pairs (e.g., <code>options = c("COMPRESS=LZW")</code> to set LZW compression during creation of a GTiff file). The <code>APPEND_SUBDATASET=YES</code> 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\(\)](#), [getCreationOptions\(\)](#), [translate\(\)](#)

Examples

```
lcp_file <- system.file("extdata/storm_lake.lcp", package="gdalraster")
tif_file <- paste0(tempdir(), "/", "storml_lndscp.tif")
opt <- c("COMPRESS=LZW")
createCopy(format="GTiff", dst_filename=tif_file, src_filename=lcp_file,
           options=opt)
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.

deleteDataset	<i>Delete named dataset</i>
---------------	-----------------------------

Description

deleteDataset() will attempt to delete the named dataset in a format specific fashion. Full featured drivers will delete all associated files, database objects, or whatever is appropriate. The default behavior when no format specific behavior is provided is to attempt to delete all the files that would be returned by GDALRaster\$getFileList() on the dataset. The named dataset should not be open in any existing GDALRaster objects when deleteDataset() is called. Wrapper for GDALDeleteDataset() in the GDAL API.

Usage

```
deleteDataset(filename, format = "")
```

Arguments

filename	Filename to delete (should not be open in a GDALRaster object).
format	Raster format short name (e.g., "GTiff"). If set to empty string "" (the default), will attempt to guess the raster format from filename.

Value

Logical TRUE if no error or FALSE on failure.

Note

If format is set to an empty string "" (the default) then the function will try to identify the driver from filename. This is done internally in GDAL by invoking the Identify method of each registered GDALDriver in turn. The first driver that successfully identifies the file name will be returned. An error is raised if a format cannot be determined from the passed file name.

See Also

[GDALRaster-class](#), [create\(\)](#), [createCopy\(\)](#), [copyDatasetFiles\(\)](#), [renameDataset\(\)](#)

Examples

```
b5_file <- system.file("extdata/sr_b5_20200829.tif", package="gdalraster")
b5_tmp <- paste0(tempdir(), "/", "b5_tmp.tif")
file.copy(b5_file, b5_tmp)

ds <- new(GDALRaster, b5_tmp, read_only=TRUE)
ds$buildOverviews("BILINEAR", levels = c(2, 4, 8), bands = c(1))
files <- ds$getFileList()
print(files)
ds$close()
```

```
file.exists(files)
deleteDataset(b5_tmp)
file.exists(files)
```

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 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/storm1_elev.tif", package="gdalraster")
slp_file <- paste0(tempdir(), "/", "storm1_slp.tif")
dem_proc("slope", elev_file, slp_file)
```

displayRAT	<i>Display a GDAL Raster Attribute Table</i>
------------	--

Description

displayRAT() generates a presentation table. Colors are shown if the Raster Attribute Table contains RGB columns. This function requires package gt.

Usage

```
displayRAT(tbl, title = "Raster Attribute Table")
```

Arguments

tbl	A data frame formatted as a GDAL RAT (e.g., as returned by buildRAT() or GDALRaster\$getDefaultRAT()).
title	Character string to be used in the table title.

Value

An object of class "gt_tbl" (i.e., a table created with gt::gt()).

See Also

[buildRAT\(\)](#), [GDALRaster\\$getDefaultRAT\(\)](#)
[vignette\("raster-attribute-tables"\)](#)

Examples

```
# see examples for `buildRAT()`
```

epsg_to_wkt	<i>Convert spatial reference from EPSG code to OGC Well Known Text</i>
-------------	--

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(eps_g_to_wkt(5070, pretty=TRUE))
set_config_option("OSR_WKT_FORMAT", "WKT2")
writeLines(eps_g_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
tbl <- buildRAT(elev_file)
head(tbl)
tbl[is.na(tbl$VALUE),]

## 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)

mod_tbl = buildRAT(mod_file)
head(mod_tbl)
mod_tbl[is.na(mod_tbl$VALUE),]
```

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 Raster 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 (see 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. GDALRaster is a C++ class exposed directly to R (via RCPP_EXPOSED_CLASS). Methods of the class are accessed in R using the \$ operator.

Usage

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

## Methods (see Details)
ds$getFilename()
ds$open(read_only)
ds$isOpen()
ds$getFileList()

ds$info()
ds$infoAsJSON()

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$getRasterCount()
ds$getDescription(band)
ds$setDescription(band)
ds$getBlockSize(band)
ds$getOverviewCount(band)
ds$buildOverviews(resampling, levels, bands)
ds$getDataTypeName(band)
ds$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$getRasterColorInterp(band)
ds$setRasterColorInterp(band, col_interp)

ds$getMinMax(band, approx_ok)
ds$getStatistics(band, approx_ok, force)
ds$getHistogram(band, min, max, num_buckets, incl_out_of_range, approx_ok)
ds$getDefaultHistogram(band, force)

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

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

ds$getColorTable(band)
ds$getPaletteInterp(band)
ds$setColorTable(band, col_tbl, palette_interp)

ds$getDefaultRAT(band)
ds$setDefaultRAT(band, df)

ds$flushCache()

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 (filename originally used to open the dataset).

`$open(read_only)` (Re-)opens the raster dataset on the existing filename. Use this method to open a dataset that has been closed using `$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.

`$getFileList()` Returns a character vector of files believed to be part of this dataset. If it returns an empty string ("") it means there is believed to be no local file system files associated with the dataset (e.g., a virtual file system). The returned filenames will normally be relative or absolute paths depending on the path used to originally open the dataset.

`$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.

`$infoAsJSON()` Returns information about the raster dataset as a JSON-formatted string. Contains full output of the `gdalinfo` command-line utility (`gdalinfo -json -stats -hist filename`).

`$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 OGC 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).

`$getDescription(band)` Returns a string containing the description for band. An empty string is returned if no description is set for the band. (Setting `band = 0` will return the dataset-level description.)

`$setDescription(band, desc)` Sets a description for band. `desc` is the character string to set. No return value.

`$getBlockSize(band)` Returns an integer vector of length two (`xsize`, `ysize`) containing the "natural" block size of band. GDAL has a concept of the natural block size of rasters so that applications can organize data access efficiently for some file formats. The natural block size is the block size that is most efficient for accessing the format. For many formats this is simply a whole row in which case block `xsize` is the same as `$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 (a.k.a. pyramids) 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. External overviews created in GTiff format may be compressed using the `COMPRESS_OVERVIEW` configuration option. All compression methods supported by the GTiff driver are available (e.g., `set_config_option("COMPRESS_OVERVIEW", "LZW")`). Since GDAL 3.6, `COMPRESS_OVERVIEW` is honoured when creating internal overviews of GTiff files. The [GDAL documentation for `gdaladdo`](#) command-line utility describes additional configuration for overview building. See also `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.

`$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. 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 empty string "" (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.

`$getScale(band)` Returns the pixel value scale ($\text{units value} = (\text{raw value} * \text{scale}) + \text{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.

`$setScale(band, scale)` Sets the pixel value scale ($\text{units value} = (\text{raw value} * \text{scale}) + \text{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.

`$getOffset(band)` Returns the pixel value offset ($\text{units value} = (\text{raw value} * \text{scale}) + \text{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.

`$setOffset(band, offset)` Sets the pixel value offset ($\text{units value} = (\text{raw value} * \text{scale}) + \text{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.

`$getRasterColorInterp(band)` Returns a string describing the color interpretation for band. The color interpretation values and their meanings are:

Undefined	Undefined
Gray	Grayscale
Palette	Paletted (see associated color table)
Red	Red band of RGBA image
Green	Green band of RGBA image

Blue	Blue band of RGBA image
Alpha	Alpha (0=transparent, 255=opaque)
Hue	Hue band of HLS image
Saturation	Saturation band of HLS image
Lightness	Lightness band of HLS image
Cyan	Cyan band of CMYK image
Magenta	Magenta band of CMYK image
Yellow	Yellow band of CMYK image
Black	Black band of CMYK image
YCbCr_Y	Y Luminance
YCbCr_Cb	Cb Chroma
YCbCr_Cr	Cr Chroma

`$setRasterColorInterp(band, col_interp)` Sets the color interpretation for band. See above for the list of valid values for `col_interp` (passed as a string).

`$getMinMax(band, approx_ok)` Returns a numeric vector of length two containing the min/max values for band. If `approx_ok` is TRUE and the raster format knows these values intrinsically then those values will be returned. If that doesn't work, a subsample of blocks will be read to get an approximate min/max. If the band has a nodata value it will be excluded from the minimum and maximum. If `approx_ok` is FALSE, then all pixels will be read and used to compute an exact range.

`$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.

`$getHistogram(band, min, max, num_buckets, incl_out_of_range, approx_ok)`

Computes raster histogram for band. `min` is the lower bound of the histogram. `max` is the upper bound of the histogram. `num_buckets` is the number of buckets to use (bucket size is $(\text{max} - \text{min}) / \text{num_buckets}$). `incl_out_of_range` is a logical scalar: if TRUE values below the histogram range will be mapped into the first bucket and values above will be mapped into the last bucket, if FALSE out of range values are discarded. `approx_ok` is a logical scalar: TRUE if an approximate histogram is OK (generally faster), or FALSE for an exactly computed histogram. Returns the histogram as a numeric vector of length `num_buckets`.

`$getDefaultHistogram(band, force)` Returns a default raster histogram for band. In the GDAL API, this method is overridden by derived classes (such as `GDALPamRasterBand`, `VRTDataset`, `HFADataset`...) that may be able to fetch efficiently an already stored histogram. `force` is a logical scalar: `TRUE` to force the computation of a default histogram; or if `FALSE` and no default histogram is available, a warning is emitted and the returned list has a 0-length histogram vector. Returns a list of length four containing named elements `$min` (lower bound), `$max` (upper bound), `$num_buckets` (number of buckets), and `$histogram` (a numeric vector of length `num_buckets`).

`$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.

`$getMetadataDomainList(band)` Returns a character vector of metadata domains or "" (empty string). Set `band = 0` to retrieve dataset-level domains, or to an integer band number to retrieve band-level domains.

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

`$getColorTable(band)` Returns the color table associated with band, or NULL if there is no associated color table. The color table is returned as an integer matrix with five columns. To associate a color with a raster pixel, the pixel value is used as a subscript into the color table. This means that the colors are always applied starting at zero and ascending (see [GDAL Color Table](#)). Column 1 contains the pixel values. Interpretation of columns 2:5 depends on the value of `$getPaletteInterp()` (see below). For "RGB", columns 2:5 contain red, green, blue, alpha as 0-255 integer values.

`$getPaletteInterp(band)` If band has an associated color table, this method returns a character string with the palette interpretation for columns 2:5 of the table. An empty string ("") is returned if band does not have an associated color table. The palette interpretation values and their meanings are:

Gray	column 2 contains grayscale values (columns 3:5 unused)
RGB	columns 2:5 contain red, green, blue, alpha
CMYK	columns 2:5 contain cyan, magenta, yellow, black
HLS	columns 2:4 contain hue, lightness, saturation (column 5 unused)

`$setColorTable(band, col_tbl, palette_interp)` Sets the raster color table for band (see [GDAL Color Table](#)). `col_tbl` is an integer matrix or data frame with either four or five columns (see `$getColorTable()` above). Column 1 contains the pixel values. Valid values are integers 0 and larger (note that GTiff format supports color tables only for Byte and UInt16 bands). Negative values will be skipped with a warning emitted. Interpretation of columns 2:5 depends on the value of `$getPaletteInterp()` (see above). For "RGB", columns 2:4 contain red, green, blue as 0-255 integer values, and an optional column 5 contains alpha transparency values (defaults to 255 opaque). `palette_interp` is one of "Gray", "RGB", "CMYK" or "HLS" (see `$getPaletteInterp()` above). Returns logical TRUE on success or FALSE if the color table could not be set.

`$getDefaultRAT(band)` Returns the Raster Attribute Table for band as a data frame, or NULL if there is no associated Raster Attribute Table. See the stand-alone function `buildRAT()` for details of the Raster Attribute Table format.

`$setDefaultRAT(band, df)` Sets a default Raster Attribute Table for band from data frame `df`. The input data frame will be checked for attribute "GDALRATTableType" which can have values of "thematic" or "athematic" (for continuous data). Columns of the data frame will be checked for attribute "GFU" (for "GDAL field usage"). If the "GFU" attribute is missing, a value of "Generic" will be used (corresponding to `GFU_Generic` in the GDAL API, for general purpose field). Columns with other, specific field usage values should generally be present in `df`, such as fields containing the set of unique (discrete) pixel values (`GFU_MinMax`), pixel counts (`GFU_PixelCount`), class names (`GFU_Name`), color values (`GFUs_Red`, "Green", "Blue"), etc. The data frame will also be checked for attributes "Row0Min" and "BinSize" which can have numeric values that describe linear binning. See the stand-alone function `buildRAT()` for details of the GDAL Raster Attribute Table format and its representation as data frame.

`$flushCache()` Flush all write cached data to disk. Any raster data written via GDAL calls, but buffered internally will be written to disk. Using this method does not preclude calling `$close()` to properly close the dataset and ensure that important data not addressed by `$flushCache()` is written in the file (see also `$open()` above). No return value, called for side effect.

`$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")`
 vignette("raster-api-tutorial")
[read_ds\(\)](#) 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 a list of files composing the dataset
ds$getFileList()

## 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$getDescription(band=1)
ds$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, "usdaforests/service/gdalraster/main/sample-data/")
url <- paste0(url, "lf_elev_220_mt_hood_utm.tif")

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

```

gdal_formats

Report all configured GDAL drivers for raster formats

Description

gdal_formats() prints to the console a list of the supported raster formats.

Usage

```
gdal_formats()
```

Value

No return value, called for reporting only.

Examples

```
gdal_formats()
```

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., "3060300" 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()
```

```
getCreationOptions    Return the list of creation options of a GDAL driver
```

Description

getCreationOptions() returns the list of creation options supported by a GDAL format driver as an XML string (invisibly). Wrapper for GDALGetDriverCreationOptionList() in the GDAL API. Information about the available creation options is also printed to the console by default.

Usage

```
getCreationOptions(format, filter = NULL)
```

Arguments

format	Raster format short name (e.g., "GTiff").
filter	Optional character vector of creation option names. Controls only the amount of information printed to the console. By default, information for all creation options is printed. Can be set to empty string "" to disable printing information to the console.

Value

Invisibly, an XML string that describes the full list of creation options or empty string "" (full output of GDALGetDriverCreationOptionList() in the GDAL API).

See Also

[GDALRaster-class](#), [create\(\)](#), [createCopy\(\)](#)

Examples

```
getCreationOptions("GTiff", filter="COMPRESS")
```

get_cache_used	<i>Get the size of memory in use by the GDAL block cache</i>
----------------	--

Description

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

Usage

```
get_cache_used()
```

Value

Integer. Amount of cache memory in use in MB.

See Also

[GDAL Block Cache](#)

Examples

```
get_cache_used()
```

get_config_option	<i>Get GDAL configuration option</i>
-------------------	--------------------------------------

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()
vignette("gdal-config-quick-ref")
```

Examples

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

get_pixel_line	<i>Raster pixel/line from geospatial x,y coordinates</i>
----------------	--

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/storm1_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()
```

g_buffer	<i>Compute buffer of a WKT geometry</i>
----------	---

Description

`g_buffer()` builds a new geometry containing the buffer region around the geometry on which it is invoked. The buffer is a polygon containing the region within the buffer distance of the original geometry. Requires GDAL built with the GEOS library.

Usage

```
g_buffer(wkt, dist, quad_segs = 30)
```

Arguments

wkt	Character. OGC WKT string for a simple feature 2D geometry.
dist	Numeric buffer distance in units of the wkt geometry.
quad_segs	Integer number of segments used to define a 90 degree curve (quadrant of a circle). Large values result in large numbers of vertices in the resulting buffer geometry while small numbers reduce the accuracy of the result.

Value

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

See Also

[bbox_from_wkt\(\)](#), [bbox_to_wkt\(\)](#)

Examples

```
g_buffer(wkt = "POINT (0 0)", dist = 10)
```

has_geos	<i>Is GEOS available?</i>
----------	---------------------------

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	<i>Invert geotransform</i>
------------------	----------------------------

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 coordinate system
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/storm1_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,
  maxColorValue = 1,
  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 = FALSE,
    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 attempt to 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 or five columns. Column 1 contains the numeric pixel values. Columns 2:4 contain the intensities of the red, green and blue primaries (0:1 by default, or use integer 0:255 by setting maxColorValue = 255). An optional column 5 may contain alpha transparency values, 0 for fully transparent to 1 (or maxColorValue) for opaque (the default if column 5 is missing). If data is a GDALRaster object, a built-in color table will be used automatically if one exists in the dataset.
maxColorValue	A number giving the maximum of the color values range in col_tbl (see above). The default is 1.
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 <code>grDevices::gray</code> for single-band data or <code>grDevices::rgb</code> for 3-band). Ignored if <code>col_tbl</code> is used. Set <code>normalize</code> to <code>FALSE</code> 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 (<code>c(0, xsize)</code>).
ylim	Numeric vector of length two giving the y coordinate range. The default uses pixel/line coordinates (<code>c(ysize, 0)</code>).
interpolate	Logical indicating whether to apply linear interpolation to the image when drawing (default <code>TRUE</code>).
asp	Numeric. The aspect ratio y/x (see <code>?plot.window</code>).
axes	Logical. <code>TRUE</code> to draw axes (the default).
main	The main title (on top).
xlab	Title for the x axis (see <code>?title</code>).
ylab	Title for the y axis (see <code>?title</code>).
xaxs	The style of axis interval calculation to be used for the x axis (see <code>?par</code>).
yaxs	The style of axis interval calculation to be used for the y axis (see <code>?par</code>).
legend	Logical indicating whether to include a legend on the plot. Currently, legends are only supported for continuous data. A color table will be used if one is specified or the raster has a built-in color table, otherwise the value for <code>col_map_fn</code> will be used.
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 (<code>"#00000000"</code> , the return value of <code>rgb(0,0,0,0)</code>).
...	Other parameters to be passed to <code>plot.default()</code> .

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, legend=TRUE, 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, legend=TRUE,
            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()

```

Description

`polygonize()` creates vector polygons for all connected regions of pixels in a source raster sharing a common pixel value. Each polygon is created with an attribute indicating the pixel value of that polygon. A raster mask may also be provided to determine which pixels are eligible for processing. The function will create the output vector layer if it does not already exist, otherwise it will try to append to an existing one. This function is a wrapper of `GDALPolygonize` in the GDAL Algorithms API. It provides essentially the same functionality as the `gdal_polygonize.py` command-line program (https://gdal.org/programs/gdal_polygonize.html).

Usage

```

polygonize(
    raster_file,
    out_dsn,
    out_layer,
    fld_name = "DN",
    out_fmt = NULL,
    connectedness = 4,
    src_band = 1,
    mask_file = NULL,
    nomask = FALSE,
    overwrite = FALSE,
    dsco = NULL,
    lco = NULL
)

```

Arguments

<code>raster_file</code>	Filename of the source raster.
<code>out_dsn</code>	The destination vector filename to which the polygons will be written (or database connection string).
<code>out_layer</code>	Name of the layer for writing the polygon features. For single-layer file formats such as "ESRI Shapefile", the layer name is the same as the filename without the path or extension (e.g., <code>out_dsn = "path_to_file/polygon_output.shp"</code> , the layer name is "polygon_output").
<code>fld_name</code>	Name of an integer attribute field in <code>out_layer</code> to which the pixel values will be written. Will be created if necessary when using an existing layer.
<code>out_fmt</code>	GDAL short name of the output vector format. If unspecified, the function will attempt to guess the format from the filename/connection string.
<code>connectedness</code>	Integer scalar. Must be either 4 or 8. For the default 4-connectedness, pixels with the same value are considered connected only if they touch along one of the four sides, while 8-connectedness also includes pixels that touch at one of the corners.
<code>src_band</code>	The band on <code>raster_file</code> to build the polygons from (default is 1).
<code>mask_file</code>	Use the first band of the specified raster as a validity mask (zero is invalid, non-zero is valid). If not specified, the default validity mask for the input band (such as nodata, or alpha masks) will be used (unless <code>nomask</code> is set to TRUE).

nomask	Logical scalar. If TRUE, do not use the default validity mask for the input band (such as nodata, or alpha masks). Default is FALSE.
overwrite	Logical scalar. If TRUE, overwrite out_layer if it already exists. Default is FALSE.
dsco	Optional character vector of format-specific creation options for out_dsn ("NAME=VALUE" pairs).
lco	Optional character vector of format-specific creation options for out_layer ("NAME=VALUE" pairs).

Details

Polygon features will be created on the output layer, with polygon geometries representing the polygons. The polygon geometries will be in the georeferenced coordinate system of the raster (based on the geotransform of the source dataset). It is acceptable for the output layer to already have features. If the output layer does not already exist, it will be created with coordinate system matching the source raster.

The algorithm attempts to minimize memory use so that very large rasters can be processed. However, if the raster has many polygons or very large/complex polygons, the memory use for holding polygon enumerations and active polygon geometries may grow to be quite large.

The algorithm will generally produce very dense polygon geometries, with edges that follow exactly on pixel boundaries for all non-interior pixels. For non-thematic raster data (such as satellite images) the result will essentially be one small polygon per pixel, and memory and output layer sizes will be substantial. The algorithm is primarily intended for relatively simple thematic rasters, masks, and classification results.

Note

The source pixel band values are read into a signed 64-bit integer buffer (Int64) by GDALPolygonize, so floating point or complex bands will be implicitly truncated before processing.

When 8-connectedness is used, many of the resulting polygons will likely be invalid due to ring self-intersection (in the strict OGC definition of polygon validity). They may be suitable as-is for certain purposes such as calculating geometry attributes (area, perimeter). Package `sf` has `st_make_valid()`, PostGIS has `ST_MakeValid()`, and QGIS has vector processing utility "Fix geometries" (single polygons can become MultiPolygon in the case of self-intersections).

If writing to a SQLite database format as either GPKG (GeoPackage vector) or SQLite (Spatialite vector), setting the `SQLITE_USE_OGR_VFS` and `OGR_SQLITE_JOURNAL` configuration options may increase performance substantially. If writing to PostgreSQL (PostGIS vector), setting `PG_USE_COPY=YES` is faster:

```
# SQLite: GPKG (.gpkg) and Spatialite (.sqlite)
# enable extra buffering/caching by the GDAL/OGR I/O layer
set_config_option("SQLITE_USE_OGR_VFS", "YES")}
# set the journal mode for the SQLite database to MEMORY
set_config_option("OGR_SQLITE_JOURNAL", "MEMORY")

# PostgreSQL / PostGIS
# use COPY for inserting data rather than INSERT
set_config_option("PG_USE_COPY", "YES")
```

See Also

[rasterize\(\)](#)
[vignette\("gdal-config-quick-ref"\)](#)

Examples

```

evt_file <- system.file("extdata/storml_evt.tif", package="gdalraster")
dsn <- paste0(tempdir(), "/", "storm_lake.gpkg")
layer <- "lf_evt"
fld <- "evt_value"
set_config_option("SQLITE_USE_OGR_VFS", "YES")
set_config_option("OGR_SQLITE_JOURNAL", "MEMORY")
polygonize(evt_file, dsn, layer, fld)
set_config_option("SQLITE_USE_OGR_VFS", "")
set_config_option("OGR_SQLITE_JOURNAL", "")

```

proj_networking	<i>Check, enable or disable PROJ networking capabilities</i>
-----------------	--

Description

proj_networking() returns the status of PROJ networking capabilities, optionally enabling or disabling first. Requires GDAL 3.4 or later and PROJ 7 or later.

Usage

```
proj_networking(enabled = NULL)
```

Arguments

enabled	Optional logical scalar. Set to TRUE to enable networking capabilities or FALSE to disable.
---------	---

Value

Logical TRUE if PROJ networking capabilities are enabled (as indicated by the return value of OSRGetPROJEnableNetwork() in the GDAL Spatial Reference System C API). Logical NA is returned if GDAL < 3.4.

See Also

[gdal_version\(\)](#), [proj_version\(\)](#), [proj_search_paths\(\)](#)
[PROJ-data on GitHub](#), [PROJ Content Delivery Network](#)

Examples

```
proj_networking()
```

proj_search_paths *Get or set search path(s) for PROJ resource files*

Description

proj_search_paths() returns the search path(s) for PROJ resource files, optionally setting them first. Requires GDAL 3.0.3 or later.

Usage

```
proj_search_paths(paths = NULL)
```

Arguments

paths Optional character vector containing one or more directory paths to set.

Value

A character vector containing the currently used search path(s) for PROJ resource files. An empty string ("") is returned if no search paths are returned by the function OSRGetPROJSearchPaths() in the GDAL Spatial Reference System C API. NA is returned if GDAL < 3.0.3.

See Also

[gdal_version\(\)](#), [proj_version\(\)](#), [proj_networking\(\)](#)

Examples

```
proj_search_paths()
```

proj_version *Get PROJ version*

Description

proj_version() returns version information for the PROJ library in use by GDAL. Requires GDAL >= 3.0.1.

Usage

```
proj_version()
```

Value

A list of length four containing:

- name - a string formatted as "major.minor.patch"
- major - major version as integer
- minor - minor version as integer
- patch - patch version as integer

List elements will be NA if GDAL < 3.0.1.

See Also

[gdal_version\(\)](#), [proj_search_paths\(\)](#), [proj_networking\(\)](#)

Examples

```
proj_version()
```

rasterFromRaster	<i>Create a raster from an existing raster as template</i>
------------------	--

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. Commonly used types include "Byte", "Int16", "UInt16", "Int32" and "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\(\)](#), [bandCopyWholeRaster\(\)](#), [translate\(\)](#)

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")
opt = c("COMPRESS=LZW")
rasterFromRaster(srcfile = lcp_file,
                 dstfile = slpp_file,
                 nbands = 1,
                 dtName = "Int16",
                 options = opt,
                 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()

```

rasterize

Burn vector geometries into a raster

Description

rasterize() burns vector geometries (points, lines, or polygons) into the band(s) of a raster dataset. Vectors are read from any GDAL OGR-supported vector format. This function is a wrapper for the gdal_rasterize command-line utility (https://gdal.org/programs/gdal_rasterize.html).

Usage

```

rasterize(
  src_dsn,
  dstfile,
  band = NULL,
  layer = NULL,
  where = NULL,
  sql = NULL,
  burn_value = NULL,
  burn_attr = NULL,
  invert = NULL,
  te = NULL,
  tr = NULL,
  tap = NULL,
  ts = NULL,
  dtName = NULL,
  dstnodata = NULL,
  init = NULL,
  fmt = NULL,
  co = NULL,
  add_options = NULL
)

```

Arguments

<code>src_dsn</code>	Data source name for the input vector layer (filename or connection string).
<code>dstfile</code>	Filename of the output raster. Must support update mode access. This file will be created (or overwritten if it already exists - see Note).
<code>band</code>	Numeric vector. The band(s) to burn values into (for existing <code>dstfile</code>). The default is to burn into band 1. Not used when creating a new raster.
<code>layer</code>	Character vector of layer names(s) from <code>src_dsn</code> that will be used for input features. At least one layer name or a <code>sql</code> option must be specified.
<code>where</code>	An optional SQL WHERE style query string to select features to burn in from the input layer(s).
<code>sql</code>	An SQL statement to be evaluated against <code>src_dsn</code> to produce a virtual layer of features to be burned in (alternative to <code>layer</code>).
<code>burn_value</code>	A fixed numeric value to burn into a band for all features. A numeric vector can be supplied, one burn value per band being written to.
<code>burn_attr</code>	Character string. Name of an attribute field on the features to be used for a burn-in value. The value will be burned into all output bands.
<code>invert</code>	Logical scalar. TRUE to invert rasterization. Burn the fixed burn value, or the burn value associated with the first feature, into all parts of the raster not inside the provided polygon.
<code>te</code>	Numeric vector of length four. Sets the output raster extent. The values must be expressed in georeferenced units. If not specified, the extent of the output raster will be the extent of the vector layer.
<code>tr</code>	Numeric vector of length two. Sets the target pixel resolution. The values must be expressed in georeferenced units. Both must be positive.
<code>tap</code>	Logical scalar. (target aligned pixels) Align the coordinates of the extent of the output raster to the values of <code>tr</code> , such that the aligned extent includes the minimum extent. Alignment means that <code>xmin / resx</code> , <code>ymin / resy</code> , <code>xmax / resx</code> and <code>ymax / resy</code> are integer values.
<code>ts</code>	Numeric vector of length two. Sets the output raster size in pixels (<code>xsize</code> , <code>ysize</code>). Note that <code>ts</code> cannot be used with <code>tr</code> .
<code>dtName</code>	Character name of output raster data type, e.g., <code>Byte</code> , <code>Int16</code> , <code>UInt16</code> , <code>Int32</code> , <code>UInt32</code> , <code>Float32</code> , <code>Float64</code> . Defaults to <code>Float64</code> .
<code>dstnodata</code>	Numeric scalar. Assign a nodata value to output bands.
<code>init</code>	Numeric vector. Pre-initialize the output raster band(s) with these value(s). However, it is not marked as the nodata value in the output file. If only one value is given, the same value is used in all the bands.
<code>fmt</code>	Output raster format short name (e.g., <code>"GTiff"</code>). Will attempt to guess from the output filename if <code>fmt</code> is not specified.
<code>co</code>	Optional list of format-specific creation options for the output raster in a vector of "NAME=VALUE" pairs (e.g., <code>options = c("TILED=YES", "COMPRESS=LZW")</code>) to set LZW compression during creation of a tiled GTiff file).
<code>add_options</code>	An optional character vector of additional command-line options to <code>gdal_rasterize</code> (see the <code>gdal_rasterize</code> documentation at the URL above for all available options).

Value

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

Note

The function creates a new target raster when any of the `fmt`, `dstnodata`, `init`, `co`, `te`, `tr`, `tap`, `ts`, or `dtName` arguments are used. The resolution or size must be specified using the `tr` or `ts` argument for all new rasters. The target raster will be overwritten if it already exists and any of these creation-related options are used.

See Also

[polygonize\(\)](#)

Examples

```
# MTBS fire perimeters for Yellowstone National Park 1984-2022
dsn <- system.file("extdata/ynp_fires_1984_2022.gpkg", package="gdalraster")
sql <- "SELECT * FROM mtbs_perims ORDER BY mtbs_perims.ig_year"
out_file <- paste0(tempdir(), "/", "ynp_fires_1984_2022.tif")

rasterize(src_dsn = dsn,
          dstfile = out_file,
          sql = sql,
          burn_attr = "ig_year",
          tr = c(90,90),
          tap = TRUE,
          dtName = "Int16",
          dstnodata = -9999,
          init = -9999,
          co = c("TILED=YES", "COMPRESS=LZW"))

ds <- new(GDALRaster, out_file, TRUE)
pal <- scales::viridis_pal(end = 0.8, direction = -1)(6)
ramp <- scales::colour_ramp(pal)
plot_raster(ds, legend = TRUE, col_map_fn = ramp, na_col = "#d9d9d9",
            main="YNP Fires 1984-2022 - Most Recent Burn Year")

ds$close()
```

rasterToVRT

Create a GDAL virtual raster derived from one source dataset

Description

`rasterToVRT()` creates a virtual raster dataset (VRT format) derived from one source dataset with options for virtual subsetting, virtually resampling the source data at a different pixel resolution, or applying a virtual kernel filter. (See [buildVRT\(\)](#) for virtual mosaicing.)

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. <ul style="list-style-type: none"> • 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	<p>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:</p> <pre>krnl <- c(0.11111, 0.11111, 0.11111, 0.11111, 0.11111, 0.11111, 0.11111, 0.11111, 0.11111)</pre> <p>A kernel cannot be applied to sub-sampled or over-sampled data.</p>
normalized	<p>Logical. Indicates whether the kernel is normalized. Defaults to TRUE.</p>

Details

rasterToVRT() can be used to virtually clip and pixel-align various raster layers with each other or 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 temp file (the default).

GDAL VRT format has several capabilities and uses beyond those covered by rasterToVRT(). See the URL 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.


```

        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 that is already open in a GDALRaster object. By default, it attempts to read the full raster extent from all bands at full resolution. read_ds() is sometimes more convenient than GDALRaster\$read(), e.g., to read specific multiple bands for display with plot_raster(), or simply for the argument defaults to read an entire raster into memory (see Note).

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 in pixels of the output buffer into which the desired region will be read (e.g., to read a reduced resolution overview).
out_ysize	Integer. The height in pixels of the output buffer into which the desired region will be read (e.g., to read a reduced resolution overview).

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$read()` directly. This would only matter if calling the function repeatedly to read a raster in chunks. For the case of reading a large raster in many chunks, it will be optimal performance-wise to call `GDALRaster$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 $(xsize * ysize * \text{number of bands} * 4)$ for data read as integer, and $(xsize * ysize * \text{number of bands} * 8)$ for data read as double (plus small object overhead for the vector).

See Also

[GDALRaster\\$read\(\)](#)

Examples

```

# read three bands from a multi-band dataset
lcp_file <- system.file("extdata/storm_lake.lcp", package="gdalraster")
ds <- new(GDALRaster, lcp_file, read_only=TRUE)

```

```
r <- read_ds(ds, bands=c(6,5,4))
typeof(r)
length(r)
object.size(r)

ds$close()
```

renameDataset	<i>Rename a dataset</i>
---------------	-------------------------

Description

renameDataset() renames a dataset in a format-specific way (e.g., rename associated files as appropriate). This could include moving the dataset to a new directory or even a new filesystem. The dataset should not be open in any existing GDALRaster objects when renameDataset() is called. Wrapper for GDALRenameDataset() in the GDAL API.

Usage

```
renameDataset(new_filename, old_filename, format = "")
```

Arguments

new_filename	New name for the dataset.
old_filename	Old name for the dataset (should not be open in a GDALRaster object).
format	Raster format short name (e.g., "GTiff"). If set to empty string "" (the default), will attempt to guess the raster format from old_filename.

Value

Logical TRUE if no error or FALSE on failure.

Note

If format is set to an empty string "" (the default) then the function will try to identify the driver from old_filename. This is done internally in GDAL by invoking the Identify method of each registered GDALDriver in turn. The first driver that successfully identifies the file name will be returned. An error is raised if a format cannot be determined from the passed file name.

See Also

[GDALRaster-class](#), [create\(\)](#), [createCopy\(\)](#), [deleteDataset\(\)](#), [copyDatasetFiles\(\)](#)

Examples

```

b5_file <- system.file("extdata/sr_b5_20200829.tif", package="gdalraster")
b5_tmp <- paste0(tempdir(), "/", "b5_tmp.tif")
file.copy(b5_file, b5_tmp)

ds <- new(GDALRaster, b5_tmp, read_only=TRUE)
ds$buildOverviews("BILINEAR", levels = c(2, 4, 8), bands = c(1))
ds$getFileList()
ds$close()
b5_tmp2 <- paste0(tempdir(), "/", "b5_tmp_renamed.tif")
renameDataset(b5_tmp2, b5_tmp)
ds <- new(GDALRaster, b5_tmp2, read_only=TRUE)
ds$getFileList()
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 scalar. 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. `RunningStats` is a C++ class exposed directly to R (via `RCPPEXPOSED_CLASS`). Methods of the class are accessed in R using the `$` operator.

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.

Note

The intended use is computing summary statistics for specific subsets or zones of a raster that could be defined in various ways and are generally not contiguous. The algorithm as implemented here incurs the cost of floating point division for each new value updated (i.e., per pixel), but is reasonably efficient for the use case. Note that GDAL internally uses an optimized version of Welford's algorithm to compute raster statistics as described in detail by Rouault, 2016 (<https://github.com/OSGeo/gdal/blob/master/gcore/statistics.txt>). The class method `GDALRaster$getStatistics()` is a GDAL API wrapper that computes statistics for a whole raster band.

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 1 minute on most 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()
vignette("gdal-config-quick-ref")
```

Examples

```
set_config_option("GDAL_CACHEMAX", "10%")
get_config_option("GDAL_CACHEMAX")
## unset:
set_config_option("GDAL_CACHEMAX", "")
```

sieveFilter	<i>Remove small raster polygons</i>
-------------	-------------------------------------

Description

sieveFilter() is a wrapper for GDALSieveFilter() in the GDAL Algorithms API. It removes raster polygons smaller than a provided threshold size (in pixels) and replaces them with the pixel value of the largest neighbour polygon.

Usage

```
sieveFilter(
  src_filename,
  src_band,
  dst_filename,
  dst_band,
  size_threshold,
  connectedness,
  mask_filename = "",
  mask_band = 0L,
  options = NULL
)
```

Arguments

src_filename	Filename of the source raster to be processed.
src_band	Band number in the source raster to be processed.
dst_filename	Filename of the output raster. It may be the same as src_filename to update the source file in place.
dst_band	Band number in dst_filename to write output. It may be the same as src_band to update the source raster in place.
size_threshold	Integer. Raster polygons with sizes (in pixels) smaller than this value will be merged into their largest neighbour.
connectedness	Integer. Either 4 indicating that diagonal pixels are not considered directly adjacent for polygon membership purposes, or 8 indicating they are.

mask_filename	Optional filename of raster to use as a mask.
mask_band	Band number in mask_filename to use as a mask. All pixels in the mask band with a value other than zero will be considered suitable for inclusion in polygons.
options	Algorithm options as a character vector of name=value pairs. None currently supported.

Details

Polygons are determined as regions of the raster where the pixels all have the same value, and that are contiguous (connected). Pixels determined to be "nodata" per the mask band will not be treated as part of a polygon regardless of their pixel values. Nodata areas will never be changed nor affect polygon sizes. Polygons smaller than the threshold with no neighbours that are as large as the threshold will not be altered. Polygons surrounded by nodata areas will therefore not be altered.

The algorithm makes three passes over the input file to enumerate the polygons and collect limited information about them. Memory use is proportional to the number of polygons (roughly 24 bytes per polygon), but is not directly related to the size of the raster. So very large raster files can be processed effectively if there aren't too many polygons. But extremely noisy rasters with many one pixel polygons will end up being expensive (in memory) to process.

The input dataset is read as integer data which means that floating point values are rounded to integers.

Value

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

Examples

```
## remove single-pixel polygons from the vegetation type layer (EVT)
evt_file <- system.file("extdata/storml_evt.tif", package="gdalraster")

# create a blank raster to hold the output
evt_mmu_file <- paste0(tempdir(), "/", "storml_evt_mmu2.tif")
rasterFromRaster(srcfile = evt_file,
                 dstfile = evt_mmu_file,
                 init = 32767)

# create a mask to exclude water pixels from the algorithm
# recode water (7292) to 0
expr <- "ifelse(EVT == 7292, 0, EVT)"
mask_file <- calc(expr = expr,
                 rasterfiles = evt_file,
                 var.names = "EVT")

# create a version of EVT with two-pixel minimum mapping unit
sieveFilter(src_filename = evt_file,
            src_band = 1,
            dst_filename = evt_mmu_file,
            dst_band = 1,
            size_threshold = 2,
```

```
connectedness = 8,  
mask_filename = mask_file,  
mask_band = 1)
```

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(eps_g_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 to 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	<i>Transform geospatial x/y coordinates</i>
--------------	---

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))
```

translate	<i>Convert raster data between different formats</i>
-----------	--

Description

translate() is a wrapper of the gdal_translate command-line utility (see https://gdal.org/programs/gdal_translate.html). The function can be used to convert raster data between different formats, potentially performing some operations like subsetting, resampling, and rescaling pixels in the process. Refer to the GDAL documentation at the URL above for a list of command-line arguments that can be passed in cl_arg.

Usage

```
translate(src_filename, dst_filename, cl_arg = NULL)
```

Arguments

src_filename	Character string. Filename of the source raster.
dst_filename	Character string. Filename of the output raster.
cl_arg	Optional character vector of command-line arguments for gdal_translate.

Value

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

See Also

[GDALRaster-class](#), [rasterFromRaster\(\)](#), [warp\(\)](#)

Examples

```
# convert the elevation raster to Erdas Imagine format and resample to 90m
elev_file <- system.file("extdata/storml_elev.tif", package="gdalraster")

# command-line arguments for gdal_translate
args <- c("-tr", "90", "90", "-r", "average")
args <- c(args, "-of", "HFA", "-co", "COMPRESSED=YES")
```

```
img_file <- paste0(tempdir(), "/", "storm1_elev_90m.img")
translate(elev_file, img_file, args)

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

warp

*Raster reprojection and mosaicing***Description**

warp() is a wrapper of the gdalwarp command-line utility for raster mosaicing, reprojection and warping (see <https://gdal.org/programs/gdalwarp.html>). The function can reproject to any supported spatial reference system (SRS). It can also be used to crop, resample, and optionally write output to a different raster format. See Details for a list of commonly used processing options that can be passed as arguments to warp().

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	Character string. Filename of the output raster.
t_srs	Character string. Target spatial reference system. Usually an EPSG code ("EPSG:#####") or a well known text (WKT) SRS definition. If empty string "", the spatial reference of src_files[1] will be used (see Note).
cl_arg	Optional character vector of command-line arguments to gdalwarp in addition to -t_srs (see Details).

Details

Several processing options can be performed in one call to warp() by passing the necessary command-line arguments. The following list describes several commonly used arguments. Note that gdalwarp supports a large number of arguments that enable a variety of different processing options. Users are encouraged to review the original source documentation provided by the GDAL project at the URL above for the full list.

- -te <xmin> <ymin> <xmax> <ymax>
Georeferenced extents of output file to be created (in target SRS by default).
- -te_srs <srs_def>
SRS in which to interpret the coordinates given with -te (if different than t_srs).

- `-tr <xres> <yres>`
Output pixel resolution (in target georeferenced units).
- `-tap`
(target aligned pixels) align the coordinates of the extent of the output file to the values of the `-tr`, such that the aligned extent includes the minimum extent. Alignment means that `xmin / resx`, `ymin / resy`, `xmax / resx` and `ymax / resy` are integer values.
- `-ovr <level>|AUTO|AUTO-<n>|NONE`
Specify which overview level of source files must be used. The default choice, `AUTO`, will select the overview level whose resolution is the closest to the target resolution. Specify an integer value (0-based, i.e., 0=1st overview level) to select a particular level. Specify `AUTO-n` where `n` is an integer greater or equal to 1, to select an overview level below the `AUTO` one. Or specify `NONE` to force the base resolution to be used (can be useful if overviews have been generated with a low quality resampling method, and the warping is done using a higher quality resampling method).
- `-wo <NAME>=<VALUE>`
Set a warp option as described in the GDAL documentation for `GDALWarpOptions`. Multiple `-wo` may be given. See also `-multi` below.
- `-ot <type>`
Force the output raster bands to have a specific data type supported by the format, which may be one of the following: `Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `Float32`, `Float64`, `CInt16`, `CInt32`, `CFloat32` or `CFloat64`.
- `-r <resampling_method>`
Resampling method to use. Available methods are: `near` (nearest neighbour, the default), `bilinear`, `cubic`, `cubicspline`, `lanczos`, `average`, `rms` (root mean square, GDAL \geq 3.3), `mode`, `max`, `min`, `med`, `q1` (first quartile), `q3` (third quartile), `sum` (GDAL \geq 3.1).
- `-srcnodata "<value>[<value>]..."`
Set nodata masking values for input bands (different values can be supplied for each band). If more than one value is supplied all values should be quoted to keep them together as a single operating system argument. Masked values will not be used in interpolation. Use a value of `None` to ignore intrinsic nodata settings on the source dataset. If `-srcnodata` is not explicitly set, but the source dataset has nodata values, they will be taken into account by default.
- `-dstnodata "<value>[<value>]..."`
Set nodata values for output bands (different values can be supplied for each band). If more than one value is supplied all values should be quoted to keep them together as a single operating system argument. New files will be initialized to this value and if possible the nodata value will be recorded in the output file. Use a value of `"None"` to ensure that nodata is not defined. If this argument is not used then nodata values will be copied from the source dataset.
- `-wm <memory_in_mb>`
Set the amount of memory that the warp API is allowed to use for caching. The value is interpreted as being in megabytes if the value is <10000 . For values ≥ 10000 , this is interpreted as bytes. The warper will total up the memory required to hold the input and output image arrays and any auxiliary masking arrays and if they are larger than the "warp memory" allowed it will subdivide the chunk into smaller chunks and try again. If the `-wm` value is very small there is some extra overhead in doing many small chunks so setting it larger is better but it is a matter of diminishing returns.
- `-multi`
Use multithreaded warping implementation. Two threads will be used to process chunks of

image and perform input/output operation simultaneously. Note that computation is not multithreaded itself. To do that, you can use the `-wo NUM_THREADS=va1/ALL_CPUS` option, which can be combined with `-multi`.

- `-of <format>` Set the output raster format. Will be guessed from the extension if not specified. Use the short format name (e.g., "GTiff").
- `-co <NAME>=<VALUE>`
Set one or more format specific creation options for the output dataset. For example, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled. `getCreationOptions()` can be used to look up available creation options, but the GDAL [Raster drivers](#) documentation is the definitive reference for format specific options. Multiple `-co` may be given, e.g.,

```
c("-co", "COMPRESS=LZW", "-co", "BIGTIFF=YES")
```
- `-overwrite`
Overwrite the target dataset if it already exists. Overwriting means deleting and recreating the file from scratch. Note that if this option is not specified and the output file already exists, it will be updated in place.

The documentation for `gdalwarp` describes additional command-line options related to spatial reference systems, source nodata values, alpha bands, polygon cutlines as mask including blending, and more.

Mosaicing into an existing output file is supported if the output file already exists. The spatial extent of the existing file will not be modified to accommodate new data, so you may have to remove it in that case, or use the `-overwrite` option.

Command-line options are passed to `warp()` as a character vector. The elements of the vector are the individual options followed by their individual values, e.g.,

```
cl_arg = c("-tr", "30", "30", "-r", "bilinear"))
```

to set the target pixel resolution to 30 x 30 in target georeferenced units and use bilinear resampling.

Value

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

Note

`warp()` can be used to reproject and also perform other processing such as crop, resample, and mosaic. This processing is generally done with a single function call by passing arguments for the target (output) pixel resolution, extent, resampling method, nodata value, format, and so forth. If `warp()` is called with `t_srs` set to "" (empty string), the target spatial reference will be set to that of `src_files[1]`, so that the processing options given in `cl_arg` will be performed without reprojecting (in the case of one input raster or multiple inputs that all use the same spatial reference system, otherwise would reproject inputs to the SRS of `src_files[1]` when they are different).

See Also

[GDALRaster-class](#), [srs_to_wkt\(\)](#), [translate\(\)](#)

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 and keep pixels aligned:
args <- c("-tr", "90", "90", "-r", "cubic", "-tap")
# write to Erdas Imagine format (HFA) with 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

* datasets

- DEFAULT_DEM_PROC, 26
- DEFAULT_NODATA, 27

- bandCopyWholeRaster, 5
- bandCopyWholeRaster(), 4, 59
- bbox_from_wkt, 6
- bbox_from_wkt(), 4, 7, 9, 47, 65
- bbox_intersect, 7
- bbox_intersect(), 4
- bbox_to_wkt, 8
- bbox_to_wkt(), 4, 7, 47
- bbox_union (bbox_intersect), 7
- bbox_union(), 4
- buildRAT, 9
- buildRAT(), 4, 21, 30, 40
- buildVRT, 12
- buildVRT(), 4, 62, 65

- calc, 14
- calc(), 4, 21
- CmbTable (CmbTable-class), 18
- CmbTable-class, 18
- combine, 20
- combine(), 4, 15
- copyDatasetFiles, 22
- copyDatasetFiles(), 4, 28, 69
- create, 23
- create(), 4, 6, 22, 26, 28, 44, 59, 69
- createColorRamp, 24
- createCopy, 25
- createCopy(), 4, 6, 22, 23, 28, 44, 59, 69

- DEFAULT_DEM_PROC, 26, 29
- DEFAULT_NODATA, 27
- deleteDataset, 28
- deleteDataset(), 4, 22, 69
- dem_proc, 29
- dem_proc(), 4, 27
- displayRAT, 30

- displayRAT(), 4, 11

- epsg_to_wkt, 30
- epsg_to_wkt(), 4, 77, 78

- fillNodata, 31
- fillNodata(), 4

- g_buffer, 47
- g_buffer(), 4, 9
- gdal_formats, 43
- gdal_formats(), 4
- gdal_version, 43
- gdal_version(), 4, 56–58
- GDALRaster (GDALRaster-class), 32
- gdalraster (gdalraster-package), 3
- GDALRaster-class, 32
- gdalraster-package, 3
- GDALRaster\$getChecksum(), 4
- GDALRaster\$getColorTable(), 24
- GDALRaster\$getDefaultRAT(), 4, 11, 30
- GDALRaster\$getGeoTransform(), 46, 48
- GDALRaster\$getPaletteInterp(), 24
- GDALRaster\$read(), 52, 68
- GDALRaster\$setColorTable(), 24
- GDALRaster\$setDefaultRAT(), 4, 11
- get_cache_used, 45
- get_cache_used(), 4
- get_config_option, 45
- get_config_option(), 4, 73
- get_pixel_line, 46
- get_pixel_line(), 4, 48
- getCreationOptions, 44
- getCreationOptions(), 4, 23, 26, 82

- has_geos, 47
- has_geos(), 4

- inv_geotransform, 48
- inv_geotransform(), 4, 46
- inv_project, 49

inv_project(), 4, 78

plot_raster, 50

plot_raster(), 4, 67

polygonize, 53

polygonize(), 4, 62

proj_networking, 56

proj_networking(), 4, 57, 58

proj_search_paths, 57

proj_search_paths(), 4, 56, 58

proj_version, 57

proj_version(), 4, 56, 57

rasterFromRaster, 58

rasterFromRaster(), 4, 6, 23, 26, 79

rasterize, 60

rasterize(), 4, 56

rasterToVRT, 62

rasterToVRT(), 4, 13, 15, 21

Rcpp_CmbTable (CmbTable-class), 18

Rcpp_CmbTable-class (CmbTable-class), 18

Rcpp_GDALRaster (GDALRaster-class), 32

Rcpp_GDALRaster-class
(GDALRaster-class), 32

Rcpp_RunningStats (RunningStats-class),
70

Rcpp_RunningStats-class
(RunningStats-class), 70

read_ds, 67

read_ds(), 41, 52

renameDataset, 69

renameDataset(), 4, 22, 28

RunningStats (RunningStats-class), 70

RunningStats-class, 70

set_config_option, 72

set_config_option(), 4, 31, 36, 41, 46, 52,
77

sieveFilter, 73

sieveFilter(), 4

srs_is_geographic, 75

srs_is_geographic(), 4, 76

srs_is_projected, 75

srs_is_projected(), 4, 75, 76

srs_is_same, 76

srs_is_same(), 4, 75, 76

srs_to_wkt, 77

srs_to_wkt(), 4, 31, 78, 82

transform_xy, 78

transform_xy(), 4, 50

translate, 79

translate(), 4, 26, 59, 82

warp, 80

warp(), 4, 65, 79