Package: gdalBindings (via r-universe)

September 3, 2024

Type Package

Title GDAL Classes Wrapper for Reading and Writing Raster Blocks

Version 0.1.33

Description Wraps around 'Geospatial' Data Abstraction Library (GDAL) raster and band classes for reading and writing directly from RasterBlock in R semantic `[[]]` and familiar syntax for accessing RasterBand and reading/writing to blocks (see <https://gdal.org/>).

License GPL-3

Depends R (>= 4.2)

Imports data.table, methods, R6, Rcpp, stats

LinkingTo Rcpp

NeedsCompilation yes

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Roxygen list(markdown = TRUE)

BugReports https://github.com/caiohamamura/gdalBindings-R/issues

URL https://github.com/caiohamamura/gdalBindings-R

SystemRequirements GDAL, PROJ

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

RemoteUrl https://github.com/caiohamamura/gdalBindings-R

RemoteRef HEAD

RemoteSha a155929d47a3cabcba4b06eeda9ef70f17adcf68

Contents

`close.GDALDataset` *Method to close GDALDataset*

Description

Closing ensures the data is actually flushed (saved) to the dataset also it releases the lock from the file.

Usage

```
## S3 method for class 'GDALDataset'
close(con, ...)
```

Arguments

`con`	GDALDataset to be closed.
`...`	not used, inherited from generic close function.

`createDataset` *Creates a new GDALDataset*

Description

Create a new raster file based on specified data. It will output a *.tif file.

Usage

```
createDataset(
  raster_path,
  nbands,
  datatype,
  projstring,
  lr_lat,
  ul_lat,
  ul_lon,
  lr_lon,
  res,
  nodata,
  co = c("TILED=YES", "BLOCKXSIZE=512", "BLOCKYSIZE=512", "COMPRESSION=LZW")
)
```

Arguments

raster_path	Character. The output path for the raster data
nbands	Integer. Number of bands. Default 1.
datatype	GDALDataType. The GDALDataType to use for the raster, use (GDALDataType$) to find the options. Default GDALDataType$GDT_Float64
projstring	The projection string, either proj or WKT is accepted.
lr_lat	Numeric. The lower right latitude.
ul_lat	Numeric. The upper left latitude.
ul_lon	Numeric. The upper left longitude.
lr_lon	Numeric. The lower right longitude.
res	Numeric. The resolution of the output raster
nodata	Numeric. The no data value for the raster. Default 0.
co	CharacterVector. A CharacterVector of creation options for GDAL. Default NULL

Value

An object from GDALDataset R6 class.

Examples

```
# Parameters
raster_path <- tempfile(fileext = ".tif")
ul_lat <- -15
ul_lon <- -45
lr_lat <- -25
lr_lon <- -35
res <- c(0.01, -0.01)
datatype <- GDALDataType$GDT_Int32
nbands <- 1
```

```
projstring <- "EPSG:4326"
nodata <- -1
co <- c("TILED=YES", "BLOCKXSIZE=512", "BLOCKYSIZE=512", "COMPRESSION=LZW")

# Create a new raster dataset
ds <- createDataset(
  raster_path = raster_path,
  nbands = nbands,
  datatype = datatype,
  projstring = projstring,
  lr_lat = lr_lat,
  ul_lat = ul_lat,
  ul_lon = ul_lon,
  lr_lon = lr_lon,
  res = res,
  nodata = nodata,
  co = co
)

# Get the GDALRasterBand for ds
band <- ds[[1]]

# Set some dummy values
band[[0, 0]] <- 1:(512 * 512)

ds$Close()
```

`formulaCalculate` *Calculate raster values based on a formula*

Description

Calculate raster values based on a formula

Usage

```
formulaCalculate(formula, data, updateBand)
```

Arguments

`formula`	Formula. A formula to apply to the RasterBands from `data`
`data`	List. A named list with the used variables in the textual formula
`updateBand`	GDALRasterBand. The GDALRasterBand which will be updated with the calculated values.

Value

Nothing, it just updates the band of interest.

Examples

```
# Parameters
raster_path <- file.path(tempdir(), "output.tif")
ul_lat <- -15
ul_lon <- -45
lr_lat <- -25
lr_lon <- -35
res <- c(0.01, -0.01)
datatype <- GDALDataType$GDT_Int32
nbands <- 2
projstring <- "EPSG:4326"
nodata <- -1
co <- c("TILED=YES", "BLOCKXSIZE=512", "BLOCKYSIZE=512", "COMPRESSION=LZW")

# Create a new raster dataset
ds <- createDataset(
  raster_path = raster_path,
  nbands = nbands,
  datatype = datatype,
  projstring = projstring,
  lr_lat = lr_lat,
  ul_lat = ul_lat,
  ul_lon = ul_lon,
  lr_lon = lr_lon,
  res = res,
  nodata = nodata,
  co = co
)

# Get the GDALRasterBand for ds
band <- ds[[1]]

# The updateBand can be the same
# using a different one just for testing
updateBand <- ds[[2]]

# Set some dummy values
band[[0, 0]] <- 1:(512 * 512)

# Calculate the double - 10
formulaCalculate(
  formula = ~x * 2 - 10,
  data = list(x = band),
  updateBand = updateBand
)

ds$Close()
```

GDALDataset *R6 Class GDALDataset wrapping*

Description

Wrapping class for GDALDataset C++ API exporting GetRasterBand, GetRasterXSize, GetRasterYSize

Methods

Public methods:

- GDALDataset$new()
- GDALDataset$GetRasterBand()
- GDALDataset$GetRasterXSize()
- GDALDataset$GetRasterYSize()
- GDALDataset$Close()
- GDALDataset$clone()

Method `new()`**:** Create a new raster file based on specified data. It will output a *.tif file.

Usage:

```
GDALDataset$new(ds)
```

Arguments:

`ds` GDALDatasetR pointer. Should not be used

`datatype` GDALDataType. The GDALDataType to use for the raster, use (GDALDataType$) to find the options. Default GDALDataType$GDT_Float64

Returns: An object from GDALDataset R6 class.

Method `GetRasterBand()`**:** Function to retrieve the GDALRasterBand R6 Object.

Usage:

```
GDALDataset$GetRasterBand(x)
```

Arguments:

`x` Integer. The band index, starting from 1 to number of bands.

Returns: An object of GDALRasterBand R6 class.

Method `GetRasterXSize()`**:** Get the width for the raster

Usage:

```
GDALDataset$GetRasterXSize()
```

Returns: An integer indicating the raster width

Method `GetRasterYSize()`**:** Get the height for the raster

Usage:

```
GDALDataset$GetRasterYSize()
```

Returns: An integer indicating the raster height

Method `Close()`**:** Closes the GDALDataset

Usage:

```
GDALDataset$Close()
```

Returns: An integer indicating the raster width

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GDALDataset$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`GDALDataType` *List of datatypes supported by the GDALDataset R6 class*

Description

List of datatypes supported by the GDALDataset R6 class

Usage

```
GDALDataType
```

Format

An object of class `list` of length 7.

`GDALOpen` *Open GDAL raster*

Description

Function to open GDAL Dataset

Usage

```
GDALOpen(filename, readonly = TRUE)
```

Arguments

`filename`	Character. The path to a GDAL dataset.
`readonly`	Logical. Flag to open a read only GDALDataset with GA_ReadOnly or GA_Update. Default TRUE.

Value

An R6 object of GDALDataset class.

Examples

```
ds_path <- system.file("extdata", "example.tif", package="gdalBindings")

ds <- GDALOpen(ds_path)
ds$Close()
```

GDALRasterBand *R6 Class GDALRasterBand wrapping*

Description

Wrapping class for GDALRasterBand C++ API exporting GetBlockXSize, GetBlockYSize, ReadBlock, WriteBlock for better IO speed.

Methods

Public methods:

- GDALRasterBand$new()
- GDALRasterBand$ReadBlock()
- GDALRasterBand$WriteBlock()
- GDALRasterBand$GetBlockXSize()
- GDALRasterBand$GetBlockYSize()
- GDALRasterBand$GetXSize()
- GDALRasterBand$CalculateStatistics()
- GDALRasterBand$GetYSize()
- GDALRasterBand$GetNoDataValue()
- GDALRasterBand$GetRasterDataType()
- GDALRasterBand$clone()

Method `new()`**:** Creates a new GDALRasterBand

Usage:

```
GDALRasterBand$new(band)
```

Arguments:

`band` The C++ pointer to the GDALRasterBandR object.

`datatype` The GDALDataType for this band

Returns: An object of GDALRasterBand R6 class

Method `ReadBlock()`**:** Efficiently reads a raster block

Usage:

```
GDALRasterBand$ReadBlock(iXBlock, iYBlock)
```

Arguments:

`iXBlock` Integer. The i-th column block to access. The `iXBlock` will be offset $BLOCKXSIZE \times iXBlock$ from the origin.

`iYBlock` Integer. The i-th row block to access. The `iYBlock` will be offset $BLOCKYSIZE \times iYBlock$ from the origin.

Details: The returned Vector will be single dimensional with the length $BLOCKXSIZE \times BLOCKYSIZE$. If you use matrix(, ncol=BLOCKXSIZE) the matrix returned will actually be transposed. You should either transpose it or you can calculate the indices using $y \cdot xsize + x$

Returns: RawVector for GDALDataType$GDT_Byte, IntegerVector for int types and NumericVector for floating point types.

Method `WriteBlock()`**:** Efficiently writes a raster block

Usage:

```
GDALRasterBand$WriteBlock(iXBlock, iYBlock, buffer)
```

Arguments:

`iXBlock` Integer. The i-th column block to write. The `iXBlock` will be offset $BLOCKXSIZE \times iXBlock$ from the origin.

`iYBlock` Integer. The i-th row block to write. The `iYBlock` will be offset $BLOCKYSIZE \times iYBlock$ from the origin.

`buffer` RawVector/IntegerVector/NumericVector depending on the GDALDataType. This should be a 1D vector with size equal to raster $BLOCKXSIZE \times BLOCKYSIZE$.

Details: The returned Vector will be single dimensional with the length $BLOCKXSIZE \times BLOCKYSIZE$. If you use matrix(, ncol=BLOCKXSIZE) the matrix returned will actually be transposed. You should either transpose it or you can calculate the indices using $y \cdot xsize + x$.

Returns: Nothing

Method `GetBlockXSize()`**:** Get the block width

Usage:

```
GDALRasterBand$GetBlockXSize()
```

Returns: An integer indicating block width

Method `GetBlockYSize()`**:** Get the block height

Usage:

```
GDALRasterBand$GetBlockYSize()
```

Returns: An integer indicating block height

Method `GetXSize()`**:** Get the band width

Usage:

```
GDALRasterBand$GetXSize()
```

Returns: An integer indicating band width

Method `CalculateStatistics()`**:** Calculate statistics for the `GDALRasterBand`

Usage:

```
GDALRasterBand$CalculateStatistics()
```

Returns: nothing

Method `GetYSize()`**:** Get the band height

Usage:

```
GDALRasterBand$GetYSize()
```

Returns: An integer indicating band height

Method `GetNoDataValue()`**:** Get band no data value

Usage:

```
GDALRasterBand$GetNoDataValue()
```

Returns: Numeric indicating no data value

Method `GetRasterDataType()`**:** Get band datatype

Usage:

```
GDALRasterBand$GetRasterDataType()
```

Returns: Numeric indicating the datatype

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GDALRasterBand$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

This constructor must not be called at all, this is automatically called from GDALDataset$GetRasterBand function.

`[[.GDALDataset` *GDALDataset [[]] accessor*

Description

This function gives access to the GDALRasterBand using [[i]], where i is the band index to return.

Usage

```
## S3 method for class 'GDALDataset'
x[[slice]]
```

Arguments

`x`	GDALDatset. Automatically obtained from GDALDataset[[]] call.
`slice`	Integer. The index for the band to access.

Value

An object of GDALRasterBand R6 class.

[[.GDALRasterBand *GDALRasterBand [[]] getter*

Description

This function gives access to the GDALRasterBand using [[i]], where i is the band index to return.

Usage

```
## S3 method for class 'GDALRasterBand'
x[[blockX, blockY]]
```

Arguments

x	GDALRasterBand. Automatically obtained from GDALDataset[[]] call.
blockX	Integer. The x index for block to access.
blockY	Integer. The y index for block to access.

Value

Nothing, this is a setter

[[<-.GDALRasterBand *GDALRasterBand [[]]= setter*

Description

This function gives access to the GDALRasterBand using [[i]], where i is the band index to return.

Usage

```
## S3 replacement method for class 'GDALRasterBand'
x[[blockX, blockY]] <- value
```

Arguments

x	GDALRasterBand. Automatically obtained from GDALDataset[[]] call.
blockX	Integer. The x index for block to access.
blockY	Integer. The y index for block to access.
value	Integer. The value buffer to write

Value

Nothing, this is a setter

Index