

Package ‘rgdal’

November 11, 2009

Title Bindings for the Geospatial Data Abstraction Library

Version 0.6-21

Date 2009-11-07

Depends R (>= 2.3.0), methods, sp

LazyLoad yes

Description Provides bindings to Frank Warmerdam’s Geospatial Data Abstraction Library (GDAL) (>= 1.3.1) and access to projection/transformation operations from the PROJ.4 library. The GDAL and PROJ.4 libraries are external to the package, and, when installing the package from source, must be correctly installed first. Both GDAL raster and OGR vector map data can be imported into R, and GDAL raster data and OGR vector data exported. Use is made of classes defined in the sp package.

Author Timothy H. Keitt <tkeitt@mail.utexas.edu>, Roger Bivand
<Roger.Bivand@nhh.no>, Edzer Pebesma <e.pebesma@geo.uu.nl>, Barry
Rowlingson

Maintainer Roger Bivand <Roger.Bivand@nhh.no>

License GPL (>= 2)

URL <http://www.gdal.org>, <http://sourceforge.net/projects/rgdal/>

SystemRequirements for building from source: GDAL >= 1.3.1 library from <http://trac.osgeo.org/gdal/wiki/DownloadSource> and PROJ.4 (proj >= 4.4.9) from <http://trac.osgeo.org/proj/>; GDAL OSX frameworks built by William Kyngesburye at <http://www.kyngchaos.com/> may be used for source installs on OSX, rgdal OSX binary packages are also available at the same site.

Repository CRAN

Date/Publication 2009-11-11 13:10:27

R topics documented:

closeDataset-methods	2
CRS-class	3
displayDataset	4
GDALDataset-class	5
GDALDriver-class	7
GDALMajorObject-class	8
GDALRasterBand-class	9
GDALReadOnlyDataset-class	11
GDALReadOnlyDataset-methods	13
GDALTransientDataset-class	14
make_EPSG	16
project	17
projInfo	18
readGDAL	19
readOGR	23
RGB2PCT	25
SGDF2PCT	26
showWKT	28
SpatialGDAL-class	29
spTransform-methods	31
writeOGR	33
Index	36

closeDataset-methods

closeDataset methods

Description

Methods for closing GDAL datasets, used internally

Usage

```
closeDataset(dataset)
closeDataset.default(dataset)
```

Arguments

dataset GDAL dataset

Methods

dataset = "ANY" default method, returns error

dataset = "GDALReadOnlyDataset" closes the "GDALReadOnlyDataset"

dataset = "GDALTransientDataset" closes the "GDALTransientDataset"

`CRS-class`*Class "CRS" of coordinate reference system arguments*

Description

Interface class to the PROJ.4 projection system. The class is defined as an empty stub accepting value NA in the sp package. If the rgdal package is available, then the class will permit spatial data to be associated with coordinate reference systems

Objects from the Class

Objects can be created by calls of the form `CRS("projargs")`, where "projargs" is a valid string of PROJ.4 arguments; the arguments must be entered exactly as in the PROJ.4 documentation, in particular there cannot be any white space in `+<arg>=<value>` strings, and successive such strings can only be separated by blanks. The initiation function calls the PROJ.4 library to verify the argument set against those known in the library, returning error messages where necessary. The complete argument set may be retrieved by examining the second list element returned by `validObject("CRS object")` to see which additional arguments the library will use (which assumptions it is making over and above submitted arguments). The function `CRSargs()` can be used to show the expanded argument list used by the PROJ.4 library.

Slots

`projargs`: Object of class "character": projection arguments; the arguments must be entered exactly as in the PROJ.4 documentation, in particular there cannot be any white space in `+<arg>=<value>` strings, and successive such strings can only be separated by blanks.

Methods

`show signature(object = "CRS")`: print projection arguments in object

Note

Lists of projections may be seen by using the programs installed with the PROJ.4 library, in particular `proj` and `cs2cs`; with the latter, `-lp` lists projections, `-le` ellipsoids, `-lu` units, and `-ld` datum(s) known to the installed software (available in **rgdal** using `projInfo`). These are added to in successive releases, so tracking the website or compiling and installing the most recent revisions will give the greatest choice. Finding the very important datum transformation parameters to be given with the `+towgs84` tag is a further challenge, and is essential when the datums used in data to be used together differ. Tracing projection arguments is easier now than before the mass ownership of GPS receivers raised the issue of matching coordinates from different argument sets (GPS output and paper map, for example).

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References

<http://proj.maptools.org/>

Examples

```
CRSargs(CRS("+proj=longlat +datum=NAD27"))
CRSargs(CRS("+init=epsg:4267"))
CRSargs(CRS("+init=epsg:26978"))
CRSargs(CRS("+proj=stere +lat_0=52.15616055555555 +lon_0=5.38763888888889 +k=0.999908 +x_0=1
# see http://trac.osgeo.org/gdal/ticket/1987
CRSargs(CRS("+init=epsg:28992"))
```

displayDataset *Display a GDAL dataset*

Description

Display a GDAL dataset allowing for subscenes and decimation, allowing very large images to be browsed

Usage

```
displayDataset(x, offset=c(0, 0), region.dim=dim(x), reduction = 1, band = 1, col =
```

Arguments

x	a three-band GDALReadOnlyDataset object
offset	Number of rows and columns from the origin (usually the upper left corner) to begin reading from; presently ordered (y,x) - this may change
region.dim	The number of rows and columns to read from the dataset; presently ordered (y,x) - this may change
reduction	a vector of length 1 or 2 recycled to 2 for decimating the input data, 1 retains full resolution, higher values decimate
band	The band number (1-based) to read from
col	default NULL, attempt to use band colour table and default to grey scale if not available
reset.par	default TRUE - reset par() settings on completion
max.dim	default 500, forcing the image to a maximum dimension of the value
...	arguments passed to image.default()

Value

a list of the image data, the colour table, and the par() values on entry.

Author(s)

Tim Keitt

References<http://www.gdal.org/>**Examples**

```
logo <- system.file("pictures/Rlogo.jpg", package="rgdal")[1]
x <- GDAL.open(logo)
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
displayDataset(x, band=1, reset.par=FALSE)
displayDataset(x, band=2, reset.par=FALSE)
displayDataset(x, band=3, reset.par=TRUE)
par(opar)
dx <- RGB2PCT(x, band=1:3)
displayDataset(dx)
GDAL.close(x)
GDAL.close(dx)
```

GDALDataset-class *Class "GDALDataset"*

Description

GDALDataset extends [GDALReadOnlyDataset-class](#) with data update commands.

Usage

```
putRasterData(dataset, rasterData, band = 1, offset = c(0, 0))
saveDataset(dataset, filename, options=NULL, returnNewObj=FALSE)
copyDataset(dataset, driver, strict = FALSE, options = NULL, fname=NULL)
deleteDataset(dataset)
saveDatasetAs(dataset, filename, driver = NULL, options=NULL)
```

Arguments

dataset	An object inheriting from class 'GDALDataset'
rasterData	A data array with <code>length(dim(rasterData)) = 2</code>
band	The band number (1-based) to read from
offset	Number of rows and columns from the origin (usually the upper left corner) to begin reading from
filename	name of file to contain raster data object

<code>returnNewObj</code>	until and including 0.5-27, <code>saveDataset</code> returned an invisible copy of the new file handle, which was then only finalized when the garbage collector ran. The old behaviour can be retained by setting to FALSE, the default behaviour is to close the handle and not return it.
<code>driver</code>	GDAL driver name to use for saving raster data object
<code>strict</code>	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>	Driver specific options (currently passed to GDAL)
<code>fname</code>	default NULL, used internally to pass through a file name with a required extension (RST driver has this problem)

Details

`putRasterData`: writes data contained in `rasterData` to the dataset, beginning at `offset` rows and columns from the origin (usually the upper left corner). Data type conversion is automatic.

`saveDataset`: saves a raster data object in a file using the driver of the object

`saveDatasetAs`: saves a raster data object in a file using the specified driver

`copyDataset`: make a copy of raster data object in a file using the specified driver

`deleteDataset`: delete the file from which the raster data object was read (should only delete files opened as GDALDataset objects)

Objects from the Class

Objects can be created by calls of the form `new("GDALDataset", filename, handle)`, where `name`: a string giving the name of a GDAL driver, `handle`: used internally; not for public consumption (default = NULL).

Slots

`handle`: Object of class "externalptr", from class "GDALReadOnlyDataset", used internally; not for public consumption

Extends

Class "GDALReadOnlyDataset", directly. Class "GDALMajorObject", by class "GDALReadOnlyDataset".

Methods

initialize `signature(.Object = "GDALDataset"):`...

Author(s)

Timothy H. Keitt, modified by Roger Bivand

See Also

[GDALDriver-class](#), [GDALReadOnlyDataset-class](#), [GDALTransientDataset-class](#)

GDALDriver-class *Class "GDALDriver": GDAL Driver Object*

Description

GDALDriver objects encapsulate GDAL file format drivers. GDALDriver inherits from [GDALMajorObject-class](#).

Usage

```
getGDALDriverNames()
gdalDrivers()
getDriverName(driver)
getDriverLongName(driver)
getGDALVersionInfo()
```

Arguments

driver An object inheriting from class 'GDALDriver'

Details

getGDALDriverNames, gdalDrivers: returns all driver names currently installed in GDAL, with their declared create and copy status (some drivers can create datasets, others can only copy from a prototype with a different driver).

getDriverName: returns the GDAL driver name associated with the driver object.

getDriverLongName: returns a longer driver name.

getGDALVersionInfo: returns the version of the GDAL runtime shared object.

Objects from the Class

Objects can be created by calls of the form `new("GDALDriver", name, handle)`, where name: a string giving the name of a GDAL driver, handle: used internally; not for public consumption (default = NULL).

Slots

handle: Object of class "externalptr", from class "GDALMajorObject", used internally; not for public consumption

Extends

Class "GDALMajorObject", directly.

Methods

initialize signature(.Object = "GDALDriver"): drivename: a string giving the name of a GDAL driver, handle: used internally; not for public consumption (default = NULL)

Note

Loading the rgdal package changes the GDAL_DATA environmental variable to the GDAL support files bundles with the package.

Author(s)

Timothy H. Keitt, modified by Roger Bivand

See Also

[GDALMajorObject-class](#)

Examples

```
gdalDrivers()
logo <- system.file("pictures/logo.jpg", package="rgdal")[1]
x <- new("GDALReadOnlyDataset", logo)
getDriver(x)
getDriverLongName(getDriver(x))
GDAL.close(x)
```

GDALMajorObject-class

Class "GDALMajorObject"

Description

"GDALMajorObject" is a virtual base class for all GDAL objects.

Usage

```
getDescription(object)
```

Arguments

object an object inheriting from "GDALMajorObject"

Details

`getDescription`: returns a description string associated with the object. No setter method is defined because GDAL dataset objects use the description to hold the filename attached to the dataset. It would not be good to change that mid-stream.

Objects from the Class

Objects can be created by calls of the form `new("GDALMajorObject", ...)`, but are only created for classes that extend this class.

Slots

handle: Object of class "externalptr", used internally; not for public consumption

Methods

No methods defined with class "GDALMajorObject" in the signature.

Author(s)

Timothy H. Keitt, modified by Roger Bivand

References

<http://www.keittlab.org/>, <http://www.gdal.org/>

See Also

[GDALDriver-class](#), [GDALReadOnlyDataset-class](#), [GDALDataset-class](#) and [GDALTransientDataset class](#)

Examples

```
driver <- new('GDALDriver', as.character(getGDALDriverNames()[1,1]))
driver
rm(driver)
logo <- system.file("pictures/logo.jpg", package="rgdal")[1]
x <- new("GDALReadOnlyDataset", logo)
x
getDescription(x)
dim(x)
GDAL.close(x)
```

GDALRasterBand-class

Class "GDALRasterBand"

Description

Returns a two-dimensional array with data from a raster band, used internally within functions

Usage

```
getRasterData(dataset, band = NULL, offset = c(0, 0),
              region.dim = dim(dataset), output.dim = region.dim,
              interleave = c(0, 0), as.is = FALSE)

getRasterTable(dataset, band = NULL, offset = c(0, 0), region.dim = dim(dataset))
```

```

getProjectionRef(dataset)

getRasterBand(dataset, band = 1)

getRasterBlockSize(raster)

toSigned(x, base)

toUnsigned(x, base)

```

Arguments

<code>dataset</code>	An object inheriting from class 'GDALReadOnlyDataset'
<code>band</code>	The band number (1-based) to read from
<code>offset</code>	Number of rows and columns from the origin (usually the upper left corner) to begin reading from; presently ordered (y,x) - this may change
<code>region.dim</code>	The number of rows and columns to read from the dataset; presently ordered (y,x) - this may change
<code>output.dim</code>	Number of rows and columns in the output data; if smaller than <code>region.dim</code> the data will be subsampled
<code>interleave</code>	Element and row stride while reading data; rarely needed
<code>as.is</code>	If false, scale the data to its natural units; if the case of thematic data, return the data as factors
<code>raster</code>	An object of class GDALRasterBand
<code>x</code>	integer variable for conversion
<code>base</code>	If Byte input, 8, if Int16 or UInt16, 16

Details

`getRasterData`: retrieves data from the dataset as array.

`getRasterTable`: retrieves data from the dataset as data frame.

`getProjectionRef`: returns the geodetic projection in Well Known Text format.

`getRasterBand`: returns a raster band

`getRasterBlockSize`: returns the natural block size of the raster band. Use this for efficient tiled IO.

`toSigned`: used to convert a band read as unsigned integer to signed integer

`toUnsigned`: used to convert a band read as signed integer to unsigned integer

Objects from the Class

Objects can be created by calls of the form `new("GDALRasterBand", dataset, band)`.

Slots

handle: Object of class "externalptr", from class "GDALMajorObject", used internally; not for public consumption

Extends

Class "GDALMajorObject", directly.

Methods

dim signature(x = "GDALRasterBand"): ...
initialize signature(.Object = "GDALRasterBand"): ...

Author(s)

Timothy H. Keitt, modified by Roger Bivand

See Also

See also [GDALDriver-class](#), [GDALDataset-class](#), [GDALTransientDataset-class](#)

Examples

```
logo <- system.file("pictures/logo.jpg", package="rgdal")[1]
x <- new("GDALReadOnlyDataset", logo)
plot(density(getRasterTable(x)$band1))
GDAL.close(x)
```

GDALReadOnlyDataset-class
Class "GDALReadOnlyDataset"

Description

GDALReadOnlyDataset is the base class for a GDAL Dataset classes. Only read operations are supported. Both GDALDataset and GDALTransientDataset inherit these read operations while providing additional write operations (see [GDALDataset-class](#)). GDALReadOnlyDataset-class inherits from [GDALMajorObject-class](#).

Usage

```
GDAL.close(dataset)
GDAL.open(filename, read.only = TRUE, silent=FALSE)
getDriver(dataset)
```

```
getColorTable(dataset, band = 1)
getGeoTransFunc(dataset)
```

Arguments

<code>dataset</code>	An object inheriting from class 'GDALReadOnlyDataset'
<code>filename</code>	A string giving the file to read from
<code>band</code>	The band number (1-based) to read from
<code>read.only</code>	A logical flag indicating whether to open the file as a GDALReadOnlyDataset or as a writable GDALDataset
<code>silent</code>	logical; if TRUE, comment and non-fatal CPL driver errors suppressed

Details

`GDAL.open` and `GDAL.close` are shorter versions of `new("GDALReadOnlyDataset", ...)` and `closeDataset()`. Because `GDAL.close` through `closeDataset()` uses the finalization mechanism to destroy the handles to the dataset and its driver, messages such as:

```
"Closing GDAL dataset handle 0x8ff7900... destroyed ... done."
```

may appear when `GDAL.close` is run, or at some later stage. `getDriver` returns an object inheriting from class 'GDALDriver'.

`getColorTable` returns the dataset colour table (currently does not support RGB imaging). `getGeoTransFunc` returns a warping function.

Objects from the Class

Objects can be created by calls of the form `new("GDALReadOnlyDataset", filename, handle)`. ~~ describe objects here ~~

Slots

```
handle: Object of class "externalptr", from class "GDALMajorObject" ~~
```

Extends

Class "GDALMajorObject", directly.

Methods

```
closeDataset signature(dataset = "GDALReadOnlyDataset"):...
```

```
dim signature(x = "GDALReadOnlyDataset"):...
```

```
initialize signature(.Object = "GDALReadOnlyDataset"):...
```

Author(s)

Timothy H. Keitt, modified by Roger Bivand

References

<http://www.keittlab.org/>, <http://www.gdal.org/>

See Also

See also [GDALDriver-class](#), [GDALDataset-class](#), [GDALTransientDataset-class](#).

Examples

```
logo <- system.file("pictures/logo.jpg", package="rgdal")[1]
x <- new("GDALReadOnlyDataset", logo)
dim(x)
plot(density(getRasterTable(x)$band1))
#displayDataset(x)
#displayDataset(x, col=function(x){rev(cm.colors(x))})
#im <- displayDataset(x, col=function(x){rev(cm.colors(x))}, reset.par=FALSE)
#contour(1:attr(im, "size")[2], 1:attr(im, "size")[1],
# t(attr(im, "index"))[,attr(im, "size")[1]:1], nlevels = 1,
# levels = 100, col = 'black', add = TRUE)
GDAL.close(x)
logo <- system.file("pictures/Rlogo.jpg", package="rgdal")[1]
x <- new("GDALReadOnlyDataset", logo)
dim(x)
#displayDataset(x)
GDAL.close(x)
```

GDALReadOnlyDataset-methods

subset methods for "GDALReadOnlyDataset"

Description

subsets GDAL objects, returning a SpatialGridDataFrame object

Details

The `[` method subsets a GDAL data set, returning a SpatialGridDataFrame object. Reading is done on the GDAL side, and only the subset requested is ever read into memory.

Further named arguments to `[` are to either `getRasterTable` or `getRasterData`:

as.is see [getRasterData](#)

interleave see [getRasterData](#)

output.dim see [getRasterData](#)

the other arguments, `offset` and `region.dim` are derived from row/column selection values.

An GDALReadOnlyDataset object can be coerced directly to a SpatialGridDataFrame

Methods

"[" signature(.Object = "GDALReadOnlyDataset"): requires package sp; selects rows and columns, and returns an object of class SpatialGridDataFrame if the grid is not rotated, or else of class SpatialPointsDataFrame. Any arguments passed to getRasterData (or in case of rotation getRasterTable) may be passed as named arguments; the first three unnamed arguments are row,col,band

Author(s)

Edzer J. Pebesma

See Also

See also [readGDAL](#) [GDALDriver-class](#), [GDALDataset-class](#), [GDALTransientDataset-class](#), [SpatialGridDataFrame-class](#).

Examples

```
library(grid)
logo <- system.file("pictures/logo.jpg", package="rgdal")[1]
x <- new("GDALReadOnlyDataset", logo)
dim(x)
x.sp = x[20:50, 20:50]
class(x.sp)
summary(x.sp)
spplot(x.sp)
GDAL.close(x)

logo <- system.file("pictures/Rlogo.jpg", package="rgdal")[1]
x.gdal <- new("GDALReadOnlyDataset", logo)
x = x.gdal[, , 3]
dim(x)
summary(x)
spplot(x)
spplot(x.gdal[])
GDAL.close(x.gdal)

logo <- system.file("pictures/Rlogo.jpg", package="rgdal")[1]
x.gdal <- new("GDALReadOnlyDataset", logo)
x.as <- as(x.gdal, "SpatialGridDataFrame")
GDAL.close(x.gdal)
summary(x.as)
```

GDALTransientDataset-class

Class "GDALTransientDataset"

Description

GDALTransientDataset is identical to [GDALDataset-class](#) except that transient datasets are not associated with any user-visible file. Transient datasets delete their associated file data when closed. See [saveDataset](#) and [saveDatasetAs](#).

Objects from the Class

Objects can be created by calls of the form `new("GDALTransientDataset", driver, rows, cols, bands, type, options, fname, handle)`.

driver A "GDALDriver" object that determines the storage format

rows Number of rows in the newly created dataset

cols Number of columns in the newly created dataset

bands Number of bands to create

type A GDAL type name as listed in `.GDALDataTypes`

options Driver specific options

fname default NULL, used internally to pass through a file name with a required extension (RST driver has this problem)

handle Used internally; not for public consumption

Slots

handle: Object of class "externalptr", from class "GDALDataset", used internally; not for public consumption

Extends

Class "GDALDataset", directly. Class "GDALReadOnlyDataset", by class "GDALDataset".
Class "GDALMajorObject", by class "GDALDataset".

Methods

closeDataset signature(`dataset = "GDALTransientDataset"`):...

initialize signature(`.Object = "GDALTransientDataset"`):...

Author(s)

Timothy H. Keitt, modified by Roger Bivand

See Also

See also [GDALDriver-class](#), [GDALReadOnlyDataset-class](#)

Examples

```
list.files(tempdir())
x <- new('GDALTransientDataset', driver=new('GDALDriver', "GTiff"), rows=100,
  cols=100, bands=3, type='Byte')
dim(x)
list.files(tempdir())
GDAL.close(x)
list.files(tempdir())
```

make_EPSG

Make a data frame of EPSG projection codes

Description

Make a data frame of the now-defunct European Petroleum Survey Group (EPSG) geodetic parameter dataset as distributed with PROJ.4 software and included in this package. Because finding the correct projection specification is not easy, lists still known as EPSG lists are maintained, and more generally retrieved from Access databases. The data collated here are as distributed with PROJ.4.

Usage

```
make_EPSG(file)
```

Arguments

<code>file</code>	file name of the file matching EPSG codes and PROJ.4 arguments, should usually be autodetected
-------------------	--

Value

returns a data frame with columns:

<code>code</code>	integer column of EPSG code numbers
<code>note</code>	character column of notes as included in the file
<code>prj4</code>	character column of PROJ.4 arguments for the equivalent projection definitions
...	

Note

See also Clifford J. Mugnier's Grids & Datums columns in Photogrammetric Engineering & Remote Sensing, <http://www.asprs.org/resources/GRIDS/>

Author(s)

Roger Bivand

References

<http://www.epsg.org/>

Examples

```
EPSG <- make_EPSG()
EPSG[grep("Oslo", EPSG$note), 1:2]
EPSG[1925:1927, 3]
EPSG[grep("Poland", EPSG$note), 1:2]
EPSG[grep("Amersfoort", EPSG$note), 1:2]
EPSG[grep("North Carolina", EPSG$note), 1:2]
EPSG[2202, 3]
```

project

Projection of coordinate matrices

Description

Interface to the PROJ.4 library of projection functions for geographical position data, no datum transformation possible. Use `spTransform()` for extended support.

Usage

```
project(xy, proj, inv = FALSE)
```

Arguments

<code>xy</code>	2-column matrix of coordinates
<code>proj</code>	character string of projection arguments; the arguments must be entered exactly as in the PROJ.4 documentation, in particular there cannot be any white space in <code>+<arg>=<value></code> strings, and successive such strings can only be separated by blanks.
<code>inv</code>	default FALSE, if TRUE inverse projection to geographical coordinates

Details

Full details of projection arguments available from website below, and examples in file "epsg" in the data directory installed with PROJ.4.

Value

A two column matrix with projected coordinates.

Note

The locations of Hawaii and Alaska in the data source are (putting it mildly) arbitrary, please avoid airlines using these positions.

Author(s)

Barry Rowlingson, Roger Bivand <Roger.Bivand@nhh.no>

References

<http://proj.maptools.org/>

See Also

[CRS-class](#), [spTransform-methods](#)

Examples

```
data(state)
res <- project(cbind(state.center$x, state.center$y), "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=0")
res1 <- project(res, "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100", inv=TRUE)
summary(res1 - cbind(state.center$x, state.center$y))
plot(cbind(state.center$x, state.center$y), asp=1, type="n")
text(cbind(state.center$x, state.center$y), state.abb)
plot(res, asp=1, type="n")
text(res, state.abb)
```

projInfo

List PROJ.4 tag information

Description

The `projInfo` function lists known values and descriptions for PROJ.4 tags for tag in `c("proj", "ellps", "datum"); getPROJ4VersionInfo` returns the version of the underlying PROJ.4 release, `getPROJ4libPath` returns the value of the `PROJ_LIB` environment variable, `projNAD` detects the presence of NAD datum conversion tables (looking for `conus`).

Usage

```
projInfo(type = "proj")
getPROJ4VersionInfo()
getPROJ4libPath()
projNAD()
```

Arguments

`type` One of these tags: `c("proj", "ellps", "datum")`

Details

The output data frame lists the information given by the `proj` application with flags `-lp`, `-le` or `-ld`.

Value

A data frame with a name and description column, and two extra columns for the "ellps" and "datum" tags.

Note

Loading the rgdal package changes the PROJ_LIB environmental variable to the PROJ.4 support files bundles with the package.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References

<http://proj.maptools.org/>

Examples

```
getPROJ4VersionInfo()
projInfo()
```

readGDAL

Read/write between GDAL grid maps and Spatial objects

Description

The functions read or write GDAL grid maps. They will set the spatial reference system if available. GDALinfo reports the size and other parameters of the dataset. create2GDAL creates a GDAL data set from a SpatialGridDataFrame object, in particular to be able to save to GDAL driver formats that only permit copying rather than creation.

Usage

```
readGDAL(fname, offset, region.dim, output.dim, band, p4s=NULL, ..., half.cell=c(0.5,0.5))
asSGDF_GROD(x, offset, region.dim, output.dim, p4s=NULL, ..., half.cell=c(0.5,0.5))
writeGDAL(dataset, fname, drivename = "GTiff", type = "Float32", mvFlag = NA, options=NULL)
create2GDAL(dataset, drivename = "GTiff", type = "Float32", mvFlag = NA, options=NULL)
GDALinfo(fname, silent=FALSE)
```

Arguments

<code>fname</code>	file name of grid map; in <code>create2GDAL</code> provides a way to pass through a file name with driver-required extension for sensitive drivers
<code>x</code>	A <code>GDALReadOnlyDataset</code> object
<code>offset</code>	Number of rows and columns from the origin (usually the upper left corner) to begin reading from; presently ordered (y,x) - this may change
<code>region.dim</code>	The number of rows and columns to read from the dataset; presently ordered (y,x) - this may change
<code>output.dim</code>	The number of rows and columns to return in the created object using GDAL's method to take care of image decimation / replication; presently ordered (y,x) - this may change
<code>band</code>	if missing, all bands are read
<code>p4s</code>	PROJ4 string defining CRS, if default (NULL), the value is read from the GDAL data set
<code>half.cell</code>	Used to adjust the intra-cell offset from corner to centre, usually as default, but may be set to <code>c=(0,0)</code> if needed; presently ordered (y,x) - this may change
<code>silent</code>	logical; if TRUE, comment and non-fatal CPL driver errors suppressed
<code>...</code>	arguments passed to either <code>getRasterData</code> , or <code>getRasterTable</code> , depending on rotation angles (see below); see the <code>rgdal</code> documentation for the available options (subsetting etc.)
<code>dataset</code>	object of class <code>SpatialGridDataFrame-class</code> or <code>SpatialPixelsDataFrame-class</code>
<code>drivername, copy_drivername</code>	GDAL driver name; if the chosen driver does not support dataset creation, an attempt is made to use the <code>copy_drivername</code> driver to create a dataset, and <code>copyDatset</code> to copy to the target driver
<code>type</code>	GDAL write data type (others than this default have not been tested)
<code>mvFlag</code>	missing value flag for output file
<code>options</code>	driver-specific options to be passed to the GDAL driver

Value

`read.GDAL` returns the data in the file as a Spatial object.

Usually, GDAL maps will be north-south oriented, in which case the `rgdal` function `getRasterData` is used to read the data, and an object of class `SpatialGridDataFrame-class` is returned.

Some map formats supported by GDAL are not north-south oriented grids. If this is the case, `readGDAL` returns the data as a set of point data, being of class `SpatialPointsDataFrame-class`. If the points are on a 45 or 90 degree rotated grid, you can try to enforce gridding later on by e.g. using `gridded-methods(x)=TRUE`.

Note

Non-fatal CPL errors may be displayed for some drivers, currently for the AIG ArcInfo 9.3 binary raster driver using GDAL \geq 1.6.2; the data has been read correctly, but the contents of the info directory did not meet the specifications used to reverse engineer the driver used in GDAL (see <http://trac.osgeo.org/gdal/ticket/3031>)

Author(s)

Edzer J.\ Pebesma, Roger Bivand

See Also

[image](#), [asciigrid](#)

Examples

```

library(grid)
GDALinfo(system.file("external/test.ag", package="sp")[1])
x <- readGDAL(system.file("external/test.ag", package="sp")[1])
class(x)
image(x)
summary(x)
x@data[[1]][x@data[[1]] > 10000] <- NA
summary(x)
image(x)

x <- readGDAL(system.file("external/simple.ag", package="sp")[1])
class(x)
image(x)
summary(x)
x <- readGDAL(system.file("pictures/big_int_arc_file.asc", package="rgdal")[1])
summary(x)
cat("if the range is not 10000, 77590, your GDAL does not detect big\n")
cat("integers for this driver\n")
y = readGDAL(system.file("pictures/Rlogo.jpg", package = "rgdal")[1], band=1)
summary(y)
y = readGDAL(system.file("pictures/Rlogo.jpg", package = "rgdal")[1])
summary(y)
splot(y, zcol=1:3, names.attr=c("red","green","blue"),
      col.regions=grey(0:100/100),
      main="example of three-layer (RGB) raster image", as.table=TRUE)
data(meuse.grid)
gridded(meuse.grid) = ~x+y
proj4string(meuse.grid) = CRS("+init=epsg:28992")
fn <- tempfile()
writeGDAL(meuse.grid["dist"], fn)
mg2 <- readGDAL(fn)
proj4string(mg2)
SP27GTIF <- readGDAL(system.file("pictures/SP27GTIF.TIF", package = "rgdal")[1], output.dim=
summary(SP27GTIF)
image(SP27GTIF, col=grey(1:99/100))

GDALinfo(system.file("pictures/cea.tif", package = "rgdal")[1])
cea <- readGDAL(system.file("pictures/cea.tif", package = "rgdal")[1], output.dim=c(100,100))
summary(cea)
image(cea, col=grey(1:99/100))
fn <- system.file("pictures/erdas_spnad83.tif", package = "rgdal")[1]
erdas_spnad83 <- readGDAL(fn, offset=c(50, 100), region.dim=c(400, 400), output.dim=c(100,100))
summary(erdas_spnad83)

```

```

image(erdas_spnad83, col=grey(1:99/100))
erdas_spnad83a <- readGDAL(fn, offset=c(50, 100), region.dim=c(400, 400))
bbox(erdas_spnad83)
bbox(erdas_spnad83a)
gridparameters(erdas_spnad83)
gridparameters(erdas_spnad83a)
tf <- tempfile()
writeGDAL(erdas_spnad83, tf, drivename="GTiff", type="Byte", options=NULL)
all.equal(erdas_spnad83, readGDAL(tf))
writeGDAL(erdas_spnad83, tf, drivename="GTiff", type="Byte", options="INTERLEAVE=PIXEL")
all.equal(erdas_spnad83, readGDAL(tf))

x <- GDAL.open(system.file("pictures/erdas_spnad83.tif", package = "rgdal")[1])
erdas_spnad83 <- asSGDF_GROD(x, output.dim=c(100,100))
GDAL.close(x)
summary(erdas_spnad83)
image(erdas_spnad83, col=grey(1:99/100))

tf <- tempfile()
xx <- create2GDAL(erdas_spnad83, type="Byte")
xxx <- copyDataset(xx, driver="PNG")
saveDataset(xxx, tf)
GDAL.close(xx)
GDAL.close(xxx)
GDALinfo(tf)

tf2 <- tempfile()
writeGDAL(erdas_spnad83, tf2, drivename="PNG", type="Byte")
GDALinfo(tf2)

GT <- GridTopology(c(0.5, 0.5), c(1, 1), c(10, 10))
set.seed(1)
SGDF <- SpatialGridDataFrame(GT, data=data.frame(z=runif(100)))
opar <- par(mfrow=c(2,2), mar=c(1,1,4,1))
image(SGDF, "z", col=colorRampPalette(c("blue", "yellow"))(20))
title(main="input values")
pfunc <- colorRamp(c("blue", "yellow"))
RGB <- pfunc(SGDF$z)
SGDF$red <- RGB[,1]
SGDF$green <- RGB[,2]
SGDF$blue <- RGB[,3]
image(SGDF, red="red", green="green", blue="blue")
title(main="input RGB")
tf <- tempfile()
writeGDAL(SGDF[c("red", "green", "blue")], tf, type="Byte", drivename="PNG")
t1 <- readGDAL(tf)
image(t1, red=1, green=2, blue=3)
title(main="output PNG RGB")
par(opar)

```

readOGR

*Read OGR vector maps into Spatial objects***Description**

The function reads an OGR data source and layer into a suitable Spatial vector object. It can only handle layers with conformable geometry features (not mixtures of points, lines, or polygons in a single layer). It will set the spatial reference system if the layer has such metadata.

If reading a shapefile, the data source name (`dsn=` argument) is the folder (directory) where the shapefile is, and the layer is the name of the shapefile (without the `.shp` extension). For example to read `bounds.shp` from `C:/Maps`, do `map <- readOGR(dsn="C:/Maps", layer="bounds")`. The logic behind this is that typically one keeps all the shapefiles for a project in one folder (directory).

As noted below, for other file type drivers, the `dsn=` argument is interpreted differently, and may be the file name itself, as for example with the GPX driver for reading GPS data as `layer="tracks"` lines or `layer="track_points"` points.

Usage

```
readOGR(dsn, layer, verbose = TRUE, p4s=NULL, drop_unsupported_fields=FALSE,
  input_field_name_encoding=NULL, pointDropZ=FALSE, dropNULLGeometries=TRUE)
ogrInfo(dsn, layer, input_field_name_encoding=NULL)
ogrFIDs(dsn, layer)
ogrDrivers()
## S3 method for class 'ogrinfo':
print(x, ...)
```

Arguments

<code>dsn</code>	data source name (interpretation varies by driver — for some drivers, <code>dsn</code> is a file name, but may also be a folder)
<code>layer</code>	layer name (varies by driver, may be a file name without extension)
<code>verbose</code>	report progress
<code>p4s</code>	PROJ4 string defining CRS, if default NULL, the value is read from the OGR data set
<code>drop_unsupported_fields</code>	default FALSE, if TRUE skip fields other than String, Integer, and Real; Date, Time and DateTime are converted to String
<code>input_field_name_encoding</code>	default NULL; if not NULL, should be a valid encoding name for the source platform of the data source and layer, see iconv
<code>pointDropZ</code>	default FALSE, if TRUE, discard third coordinates for point geometries; third coordinates are always discarded for line and polygon geometries

```
dropNULLGeometries
    default TRUE, drop both declared NULL geometries, and empty geometries
    with no coordinates; if FALSE, return a data frame with the attribute values of
    the NULL and empty geometries
x
    ogrinfo object
...
    other arguments to print method
```

Details

The drivers available will depend on the installation of GDAL/OGR, and can vary; the `ogrDrivers()` function shows which are available, and which may be written (but all are assumed to be readable). Note that stray files in data source directories (such as *.dbf) may lead to suprious errors that accompanying *.shp are missing.

Value

A Spatial object is returned suiting the vector data source, either a `SpatialPointsDataFrame` (using an `AttributeList` for its data slot directly), a `SpatialLinesDataFrame`, or a `SpatialPolygonsDataFrame`.

Note

The bases for this implementation are taken from functions in Barry Rowlingson's draft Rmap package, and from Radim Blazek's `v.in.ogr` program in GRASS.

Please note that the OGR drivers used may not handle missing data gracefully, and be prepared to have to correct for this manually. From `rgdal` 0.5-27, missing value handling has been improved, and OGR unset field values are set to NA in R, but drivers and external files may vary in their representations of missing values.

In addition, from 0.6-9 date and time fields are read as strings rather than being treated as unsupported; NULL geometries are identified and dropped. There are differences in the reporting of NULL geometries between `ogrInfo` and `readOGR` - in `ogrInfo`, only declared NULL geometries are reported, but in `readOGR`, any line or polygon geometries with no coordinates are assigned NULL geometry status as well. An attempt is made to close unclosed rings in polygon geometries.

For reading GPX files, refer to the OGR GPX format documentation for the use of layer tags: "waypoints", "tracks", "routes", "track_points" and "route_points" - reading GPX files requires a build of GDAL/OGR with the expat XML library.

From 0.6-10, attempts are made to detect deleted features still present in the layer, but not read. Apparently features deleted in Qgis are only marked as deleted, but are still in the layer. These are not NULL geometries, but still need to be handled. An attempt is made to check the FID values, and `ogrFIDs` now returns attributes permitting this oddity to be detected. Such deleted features were seen as NULL in 0.6-9, but are not such.

Author(s)

Roger Bivand

References

<http://www.gdal.org/ogr/>, http://www.gdal.org/ogr/ogr_formats.html, <http://examples.oreilly.com/webmapping/>

See Also

[SpatialPointsDataFrame-class](#), [SpatialLinesDataFrame-class](#), [SpatialPolygonsDataFrame-class](#), [readShapePoly](#), [iconv](#)

Examples

```
ogrDrivers()
dsn <- system.file("vectors", package = "rgdal")[1]
ogrInfo(dsn=dsn, layer="cities")
cities <- readOGR(dsn=dsn, layer="cities")
summary(cities)
ogrInfo(dsn=dsn, layer="kiritimati_primary_roads")
kiritimati_primary_roads <- readOGR(dsn=dsn, layer="kiritimati_primary_roads")
summary(kiritimati_primary_roads)
ogrInfo(dsn=dsn, layer="scot_BNG")
scot_BNG <- readOGR(dsn=dsn, layer="scot_BNG")
summary(scot_BNG)
if ("GML" %in% ogrDrivers()$name) {
  dsn <- system.file("vectors/airports.gml", package = "rgdal")[1]
  airports <- try(readOGR(dsn=dsn, layer="airports"))
  if (class(airports) != "try-error") summary(airports)
}
dsn <- system.file("vectors/ps_cant_31.MIF", package = "rgdal")[1]
ogrInfo(dsn=dsn, layer="ps_cant_31")
ps_cant_31 <- readOGR(dsn=dsn, layer="ps_cant_31")
summary(ps_cant_31)
dsn <- system.file("vectors/Up.tab", package = "rgdal")[1]
ogrInfo(dsn=dsn, layer="Up")
Up <- readOGR(dsn=dsn, layer="Up")
summary(Up)
dsn <- system.file("vectors/test_trk2.gpx", package = "rgdal")[1]
test_trk2 <- try(readOGR(dsn=dsn, layer="tracks"))
if (class(test_trk2) != "try-error") summary(test_trk2)
test_trk2pts <- try(readOGR(dsn=dsn, layer="track_points"))
if (class(test_trk2pts) != "try-error") summary(test_trk2pts)
dsn <- system.file("vectors", package = "rgdal")[1]
ogrInfo(dsn=dsn, layer="trin_inca_pl03")
birds <- readOGR(dsn=dsn, layer="trin_inca_pl03")
summary(birds)
```

RGB2PCT

Convert RGB three band to single band colour table

Description

This function converts a three-band GDALReadOnlyDataset into a single band of colour indices as a GDALTransientDataset.

Usage

```
RGB2PCT(x, band, driver.name = 'MEM', ncolors = 256, set.ctab = TRUE)
```

Arguments

<code>x</code>	a three-band GDALReadOnlyDataset object
<code>band</code>	a vector of numbers, recycled up to 3 in length
<code>driver.name</code>	default MEM
<code>ncolors</code>	a number of colours between 2 and 256
<code>set.ctab</code>	default TRUE, when the dithered dataset handle is returned, otherwise a list of the dataset and the PCT colour table

Value

The value returned is either a GDALTransientDataset or a list of a GDALTransientDataset and a colour table.

Author(s)

Tim Keitt

References

<http://www.gdal.org/>

Examples

```
logo <- system.file("pictures/Rlogo.jpg", package="rgdal")[1]
x <- GDAL.open(logo)
dim(x)
dx <- RGB2PCT(x, band=1:3)
displayDataset(dx)
dim(dx)
GDAL.close(x)
GDAL.close(dx)
```

SGDF2PCT

Convert RGB three band to single band colour table

Description

This function converts a three-band SpatialGridDataFrame into a single band of colour indices and a colour look-up table using RGB2PCT. `vec2RGB` uses given breaks and colours (like `image`) to make a three column matrix of red, green, and blue values for a numeric vector.

Usage

```
SGDF2PCT(x, ncolors = 256, adjust.bands=TRUE)
vec2RGB(vec, breaks, col)
```

Arguments

<code>x</code>	a three-band SpatialGridDataFrame object
<code>ncolors</code>	a number of colours between 2 and 256
<code>adjust.bands</code>	default TRUE; if FALSE the three bands must lie each between 0 and 255, but will not be stretched within those bounds
<code>vec</code>	a numeric vector
<code>breaks</code>	a set of breakpoints for the colours: must give one more breakpoint than colour
<code>col</code>	a list of colors

Value

The value returned is a list:

<code>idx</code>	a vector of colour indices in the same spatial order as the input object
<code>ct</code>	a vector of RGB colours

Author(s)

Roger Bivand

References

<http://www.gdal.org/>

Examples

```
logo <- system.file("pictures/Rlogo.jpg", package="rgdal")[1]
SGlogo <- readGDAL(logo)
cols <- SGDF2PCT(SGlogo)
SGlogo$idx <- cols$idx
image(SGlogo, "idx", col=cols$ct)
SGlogo <- readGDAL(logo)
cols <- SGDF2PCT(SGlogo, ncolors=64)
SGlogo$idx <- cols$idx
image(SGlogo, "idx", col=cols$ct)
SGlogo <- readGDAL(logo)
cols <- SGDF2PCT(SGlogo, ncolors=8)
SGlogo$idx <- cols$idx
image(SGlogo, "idx", col=cols$ct)
data(meuse.grid)
coordinates(meuse.grid) <- c("x", "y")
gridded(meuse.grid) <- TRUE
fullgrid(meuse.grid) <- TRUE
summary(meuse.grid$dist)
```

```

opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2), mar=c(1,1,1,1)+0.1)
image(meuse.grid, "dist", breaks=seq(0,1,1/10), col=bpy.colors(10))
RGB <- vec2RGB(meuse.grid$dist, breaks=seq(0,1,1/10), col=bpy.colors(10))
summary(RGB)
meuse.grid$red <- RGB[,1]
meuse.grid$green <- RGB[,2]
meuse.grid$blue <- RGB[,3]
cols <- SGDF2PCT(meuse.grid[c("red", "green", "blue")], ncolors=10, adjust.bands=FALSE)
is.na(cols$idix) <- is.na(meuse.grid$dist)
meuse.grid$idix <- cols$idix
image(meuse.grid, "idix", col=cols$ct)
par(opar)
# Note: only one wrongly classified pixel after NA handling/dropping
# The functions are not written to be reversible
sort(table(findInterval(meuse.grid$dist, seq(0,1,1/10), all.inside=TRUE)))
sort(table(cols$idix))

```

showWKT

Show Well-Known Text spatial reference system metadata

Description

Use GDAL/OGR spatial reference objects to convert a PROJ.4 representation to a Well-Known Text representation.

Usage

```
showWKT(p4s, file = NULL, morphToESRI = TRUE)
```

Arguments

p4s	A valid PROJ.4 string representing a spatial reference system
file	if not NULL, a file name to which the output Well-Known Text representation should be written
morphToESRI	default TRUE, morph the WKT string to the representation used by ESRI

Value

A character string containing the WKT representation of the PROJ.4 string.

Author(s)

Roger Bivand

References

http://www.gdal.org/ogr/osr_tutorial.html

See Also

[is.projected](#), [CRS-class](#)

Examples

```
cities <- readOGR(system.file("vectors", package = "rgdal")[1], "cities")
readLines(system.file("vectors/cities.prj", package = "rgdal")[1])
showWKT(proj4string(cities))
showWKT("+init=epsg:28992")
```

SpatialGDAL-class *Class "SpatialGDAL"*

Description

Class for spatial attributes that have spatial locations on a (full) regular grid on file, not (yet) actually read.

Usage

```
## S3 method for class 'SpatialGDAL':
open(con, ..., silent = FALSE)
## S3 method for class 'SpatialGDAL':
close(con, ...)
copy.SpatialGDAL(dataset, fname, driver = getDriver(dataset@grid), strict = FALSE,
```

Arguments

<code>con</code>	file name of grid map for opening, SpatialGDAL object for closing
<code>...</code>	other arguments (currently ignored)
<code>silent</code>	logical; if TRUE, comment and non-fatal CPL driver errors suppressed
<code>dataset</code>	object of class SpatialGDAL
<code>fname</code>	file name of grid map
<code>driver</code>	GDAL driver name
<code>strict</code>	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>	driver-specific options to be passed to the GDAL driver

Objects from the Class

Objects can be created by calls of the form `open.SpatialGDAL(name)`, where `name` is the name of the GDAL file.

Slots

points: see [SpatialPoints](#); points slot which is not actually filled with all coordinates (only with min/max)

grid: see [GridTopology-class](#); grid parameters

grid.index: see [SpatialPixels-class](#); this slot is of zero length for this class, as the grid is full

bbox: Object of class "matrix"; bounding box

proj4string: Object of class "CRS"; projection

data: Object of class data.frame, containing attribute data

Extends

Class [Spatial-class](#), directly.

Methods

[signature(x = "SpatialGDAL", i, j, ...): selects rows (i), columns (j), and bands (third argument); returns an object of class [SpatialGridDataFrame-class](#). Only the selection is actually read.

[[signature(i): reads band i and returns the values as a numeric vector

Note

Non-fatal CPL errors may be displayed for some drivers, currently for the AIG ArcInfo 9.3 binary raster driver using GDAL >= 1.6.2; the data has been read correctly, but the contents of the info directory did not meet the specifications used to reverse engineer the driver used in GDAL (see <http://trac.osgeo.org/gdal/ticket/3031>)

Author(s)

Edzer J. Pebesma, <e.pebesma@geo.uu.nl>

See Also

[SpatialGridDataFrame-class](#), which is actually sub-classed.

Examples

```
x <- open.SpatialGDAL(system.file("external/test.ag", package="sp")[1])
image(x[])
image(as(x, "SpatialGridDataFrame"))
summary(as(x, "SpatialGridDataFrame"))
spplot(as(x, "SpatialGridDataFrame"))
# select first 50 rows:
summary(x[1:50])
# select first 50 columns:
summary(x[,1:50])
# select band 1:
summary(x[, ,1])
```

```
# select first 50 rows, first 50 columns, band 1:
summary(x[1:50,1:50,1])
# get values of first band:
summary(x[[1]])
close(x)
```

spTransform-methods

Methods for Function spTransform in package "rgdal"

Description

The `spTransform` methods provide transformation between datum(s) and conversion between projections, from one unambiguously specified coordinate reference system to another, using PROJ.4 projection arguments. For simple projection, when no `+datum` tags are used, datum projection does not occur. When datum transformation is required, the `+datum` tag should be present with a valid value both in the CRS of the object to be transformed, and in the target CRS. In general `+datum=` is to be preferred to `+ellps=`, because the datum always fixes the ellipsoid, but the ellipsoid never fixes the datum.

In addition, the `+towgs84` tag should be used where needed to make sure that datum transformation does take place. Parameters for `+towgs84` will be taken from the bundled EPSG database if they are known unequivocally, but may be entered manually from known authorities. Not providing the appropriate `+datum` and `+towgs84` tags may lead to coordinates being out by hundreds of metres. Unfortunately, there is no easy way to provide this information: the user has to know the correct metadata for the data being used, even if this can be hard to discover.

Methods

"ANY" default void method

"SpatialPoints", CRSObj = CRS returns transformed coordinates of an "SpatialPoints" object using the projection arguments in "CRSObj", of class CRS

"SpatialPointsDataFrame", CRSObj = CRS returns transformed coordinates of an "SpatialPointsDataFrame" object using the projection arguments in "CRSObj", of class CRS

"SpatialLines", CRSObj = CRS returns transformed coordinates of an "SpatialLines" object using the projection arguments in "CRSObj", of class CRS

"SpatialLinesDataFrame", CRSObj = CRS returns transformed coordinates of an "SpatialLinesDataFrame" object using the projection arguments in "CRSObj", of class CRS

"SpatialPolygons", CRSObj = CRS returns transformed coordinates of an "SpatialPolygons" object using the projection arguments in "CRSObj", of class CRS

"SpatialPolygonsDataFrame", CRSObj = CRS returns transformed coordinates of an "SpatialPolygonsDataFrame" object using the projection arguments in "CRSObj", of class CRS

Note

The projection arguments must be entered exactly as in the PROJ.4 documentation, in particular there cannot be any white space in `+<arg>=<value>` strings, and successive such strings can only be separated by blanks. Note that warnings about different projections may be issued when the PROJ.4 library extends projection arguments; examine the warning to see if the differences are real.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

Examples

```

data(state)
states <- data.frame(state.x77, state.center)
states <- states[states$x > -121,]
coordinates(states) <- c("x", "y")
proj4string(states) <- CRS("+proj=longlat +ellps=clrk66")
summary(states)
state.ll83 <- spTransform(states, CRS("+proj=longlat +ellps=GRS80"))
summary(state.ll83)
state.merc <- spTransform(states, CRS=CRS("+proj=merc +ellps=GRS80"))
summary(state.merc)
state.merc <- spTransform(states, CRS=CRS("+proj=merc +ellps=GRS80 +units=us-mi"))
summary(state.merc)
if (projNAD()) {
states <- data.frame(state.x77, state.center)
states <- states[states$x > -121,]
coordinates(states) <- c("x", "y")
proj4string(states) <- CRS("+init=epsg:4267")
print(summary(states))
state.ll83 <- spTransform(states, CRS("+init=epsg:4269"))
print(summary(state.ll83))
state.kansasSlcc <- spTransform(states, CRS=CRS("+init=epsg:26978"))
print(summary(state.kansasSlcc))
SFpoint_NAD83 <- SpatialPoints(matrix(c(-103.869667, 44.461676), nrow=1),
proj4string=CRS("+init=epsg:4269"))
SFpoint_NAD27 <- spTransform(SFpoint_NAD83, CRS("+init=epsg:4267"))
print(all.equal(coordinates(SFpoint_NAD83), coordinates(SFpoint_NAD27)))
print(coordinates(SFpoint_NAD27), digits=12)
print(coordinates(SFpoint_NAD83), digits=12)
}
data(meuse)
coordinates(meuse) <- c("x", "y")
proj4string(meuse) <- CRS(paste("+init=epsg:28992",
"+towgs84=565.237,50.0087,465.658,-0.406857,0.350733,-1.87035,4.0812"))
# see http://trac.osgeo.org/gdal/ticket/1987
summary(meuse)
meuse.utm <- spTransform(meuse, CRS("+proj=utm +zone=32 +datum=WGS84"))
summary(meuse.utm)
cbind(coordinates(meuse), coordinates(meuse.utm))
kiritimati_primary_roads <- readOGR(system.file("vectors", package = "rgdal")[1], "kiritimati")
kiritimati_primary_roads_ll <- spTransform(kiritimati_primary_roads, CRS("+proj=longlat +dat

```

```

opar <- par(mfrow=c(1,2))
plot(kiritimati_primary_roads, axes=TRUE)
plot(kiritimati_primary_roads_ll, axes=TRUE, las=1)
par(opar)
opar <- par(mfrow=c(1,2))
scot_BNG <- readOGR(system.file("vectors", package = "rgdal")[1],
  "scot_BNG")
scot_LL <- spTransform(scot_BNG, CRS("+proj=longlat +datum=WGS84"))
plot(scot_LL, axes=TRUE)
grd_LL <- gridlines(scot_LL, ndiscr=100)
summary(grd_LL)
grd_BNG <- spTransform(grd_LL, CRS(proj4string(scot_BNG)))
grdtxt_LL <- gridat(scot_LL)
grdtxt_BNG <- spTransform(grdtxt_LL, CRS(proj4string(scot_BNG)))
plot(scot_BNG, axes=TRUE, las=1)
plot(grd_BNG, add=TRUE, lty=2)
text(coordinates(grdtxt_BNG),
  labels=parse(text=as.character(grdtxt_BNG$labels)))
par(opar)

```

writeOGR

Write spatial vector data using OGR

Description

The function is an interface with the OGR abstraction library for spatial vector data, allowing data to be written out using supported drivers. The drivers supported will depend on the local installation, and the capabilities of those drivers (many are read-only). The objects exported are `SpatialPointsDataFrame`, `SpatialLinesDataFrame`, or `SpatialPolygonsDataFrame` objects as defined in the `sp` package.

Usage

```
writeOGR(obj, dsn, layer, driver, dataset_options = NULL, layer_options=NULL, verbo
```

Arguments

<code>obj</code>	a <code>SpatialPointsDataFrame</code> , <code>SpatialLinesDataFrame</code> , or a <code>SpatialPolygonsDataFrame</code> object.
<code>dsn</code>	data source name (interpretation varies by driver — for some drivers, <code>dsn</code> is a file name, but may also be a folder)
<code>layer</code>	layer name (varies by driver, may be a file name without extension)
<code>driver</code>	a character string equal to one of the driver names returned by <code>ogrDrivers</code>
<code>dataset_options</code>	a character vector of options, which vary by driver, and should be treated as experimental

layer_options a character vector of options, which vary by driver, and should be treated as experimental

verbose if TRUE, returns a list of information about the attempted write operation

Details

Working out which combination of dsn, layer, and driver (and option) values give the desired output takes time and care, and is constrained by the ability of drivers to write output; many are read-only. Use of the references given is highly advisable, with searches in the archives of other software using GDAL/OGR.

Value

if verbose=TRUE, a list of information about the attempted write operation

Note

Only a subset of possible data slot column classes may be written out; if the function returns an error that the data type of stated columns is unknown, examine the classes and check that they are one of `c("numeric", "character", "factor", "POSIXt", "integer", "logical")`, and if not convert to such classes. Classes `c("factor", "POSIXt")` are converted to character strings, and `c("logical")` to integer internally.

For writing with the KML driver, note that the geometries should be in geographical coordinates with datum WGS84.

Author(s)

Roger Bivand

References

<http://www.gdal.org/ogr/>, http://www.gdal.org/ogr/ogr_formats.html, <http://examples.oreilly.com/webmapping/>

See Also

[readOGR](#)

Examples

```
cities <- readOGR(system.file("vectors", package = "rgdal")[1], "cities")
is.na(cities$POPULATION) <- cities$POPULATION == -99
summary(cities$POPULATION)
td <- tempdir()
if(nchar(Sys.getenv("OSGEO4W_ROOT") > 0)) {
  OLDPWD <- getwd()
  setwd(td)
  td <- "."
}
```

```

writeOGR(cities, td, "cities", driver="ESRI Shapefile")
cities2 <- readOGR(td, "cities")
summary(cities2$POPULATION)
all.equal(cities, cities2)
## Not run: if ("GML" %in% ogrDrivers()$name) {
  airports <- try(readOGR(system.file("vectors/airports.gml", package = "rgdal")[1], "airports"), silent=TRUE)
  if (class(airports) != "try-error") {
    writeOGR(cities, paste(td, "cities.gml", sep="/"), "cities", driver="GML")
    cities3 <- readOGR(paste(td, "cities.gml", sep="/"), "cities")
    all.equal(cities, cities3)
  }
}
## End(Not run)
# The GML driver does not support coordinate reference systems
if ("KML" %in% ogrDrivers()$name) {
  data(meuse)
  coordinates(meuse) <- c("x", "y")
  proj4string(meuse) <- CRS("+init=epsg:28992")
  meuse_ll <- spTransform(meuse, CRS("+proj=longlat +datum=WGS84"))
  writeOGR(meuse_ll["zinc"], paste(td, "meuse.kml", sep="/"), "zinc", "KML")
}
list.files(td)
roads <- readOGR(system.file("vectors", package = "rgdal")[1], "kiritimati_primary_roads")
summary(roads)
writeOGR(roads, td, "roads", driver="MapInfo File")
roads2 <- readOGR(paste(td, "roads.tab", sep="/"), "roads")
summary(roads2)
scot_BNG <- readOGR(system.file("vectors", package = "rgdal")[1], "scot_BNG")
summary(scot_BNG)
writeOGR(scot_BNG, td, "scot_BNG", driver="MapInfo File")
list.files(td)
scot_BNG2 <- readOGR(paste(td, "scot_BNG.tab", sep="/"), "scot_BNG")
summary(scot_BNG2)
writeOGR(scot_BNG, td, "scot_BNG", driver="MapInfo File", dataset_options="FORMAT=MIF")
list.files(td)
scot_BNG3 <- readOGR(paste(td, "scot_BNG.mif", sep="/"), "scot_BNG")
summary(scot_BNG3)
if (nchar(Sys.getenv("OSGEO4W_ROOT") > 0)) {
  setwd(OLDPWD)
}

```

Index

*Topic **classes**

CRS-class, 2
GDALDataset-class, 5
GDALDriver-class, 6
GDALMajorObject-class, 8
GDALRasterBand-class, 9
GDALReadOnlyDataset-class, 11
GDALReadOnlyDataset-methods,
13
GDALTransientDataset-class,
14
SpatialGDAL-class, 28

*Topic **methods**

closeDataset-methods, 2
spTransform-methods, 30

*Topic **spatial**

CRS-class, 2
displayDataset, 3
make_EPSG, 15
project, 16
projInfo, 18
readGDAL, 19
readOGR, 22
RGB2PCT, 25
SGDF2PCT, 26
showWKT, 27
spTransform-methods, 30
writeOGR, 32

, GDALReadOnlyDataset-method
(GDALReadOnlyDataset-methods),
13
[, GDALReadOnlyDataset-method
(GDALReadOnlyDataset-methods),
13
[, SpatialGDAL-method
(SpatialGDAL-class), 28
[<-, SpatialGDALWrite-method
(SpatialGDAL-class), 28
[[, SpatialGDAL, ANY, missing-method

(SpatialGDAL-class), 28
[[<-, SpatialGDAL, ANY, missing-method
(SpatialGDAL-class), 28
\$, SpatialGDAL, character-method
(SpatialGDAL-class), 28
\$<-, SpatialGDAL, character-method
(SpatialGDAL-class), 28

asciigrid, 20
asSGDF_GROD (readGDAL), 19

close.SpatialGDAL
(SpatialGDAL-class), 28
closeDataset
(closeDataset-methods), 2
closeDataset, ANY-method
(closeDataset-methods), 2
closeDataset, GDALReadOnlyDataset-method
(closeDataset-methods), 2
closeDataset, GDALTransientDataset-method
(closeDataset-methods), 2
closeDataset-methods, 2
closeDataset.default
(closeDataset-methods), 2
coerce, GDALReadOnlyDataset, SpatialGridDataFrame-
(GDALReadOnlyDataset-methods),
13
coerce, SpatialGDAL, SpatialGridDataFrame-method
(SpatialGDAL-class), 28
coerce, SpatialGDAL, SpatialPixelsDataFrame-method
(SpatialGDAL-class), 28
copy.SpatialGDAL
(SpatialGDAL-class), 28
copyDataset (GDALDataset-class), 5
create2GDAL (readGDAL), 19
CRS (CRS-class), 2
CRS-class, 17, 28
CRS-class, 2
CRSargs (CRS-class), 2

- deleteDataset
(GDALDataset-class), 5
- dim, GDALRasterBand-method
(GDALRasterBand-class), 9
- dim, GDALReadOnlyDataset-method
(GDALReadOnlyDataset-class), 11
- displayDataset, 3
- GDAL.close
(GDALReadOnlyDataset-class), 11
- GDAL.open
(GDALReadOnlyDataset-class), 11
- GDALDataset-class, 8, 10–14
- GDALDataset-class, 5
- GDALDriver-class, 6, 8, 10, 12, 13, 15
- GDALDriver-class, 6
- gdalDrivers (GDALDriver-class), 6
- GDALInfo (readGDAL), 19
- GDALMajorObject-class, 6, 7, 11
- GDALMajorObject-class, 8
- GDALRasterBand-class, 9
- GDALReadOnlyDataset-class, 5, 6, 8, 15
- GDALReadOnlyDataset-class, 11
- GDALReadOnlyDataset-methods, 13
- GDALTransientDataset-class, 6, 8, 10, 12, 13
- GDALTransientDataset-class, 14
- getColorTable
(GDALReadOnlyDataset-class), 11
- getDescription
(GDALMajorObject-class), 8
- getDriver
(GDALReadOnlyDataset-class), 11
- getDriverLongName
(GDALDriver-class), 6
- getDriverName (GDALDriver-class), 6
- getGDAL_DATA_Path
(GDALDriver-class), 6
- getGDALDriverNames
(GDALDriver-class), 6
- getGDALVersionInfo
(GDALDriver-class), 6
- getGeoTransFunc
(GDALReadOnlyDataset-class), 11
- getPROJ4libPath (projInfo), 18
- getPROJ4VersionInfo (projInfo), 18
- getProjectionRef
(GDALRasterBand-class), 9
- getRasterBand
(GDALRasterBand-class), 9
- getRasterBlockSize
(GDALRasterBand-class), 9
- getRasterData, 13
- getRasterData
(GDALRasterBand-class), 9
- getRasterTable
(GDALRasterBand-class), 9
- gridded-methods, 20
- GridTopology-class, 29
- iconv, 23, 24
- image, 20
- initialize, GDALDataset-method
(GDALDataset-class), 5
- initialize, GDALDriver-method
(GDALDriver-class), 6
- initialize, GDALRasterBand-method
(GDALRasterBand-class), 9
- initialize, GDALReadOnlyDataset-method
(GDALReadOnlyDataset-class), 11
- initialize, GDALTransientDataset-method
(GDALTransientDataset-class), 14
- is.projected, 28
- make_EPSG, 15
- ogrDrivers, 33
- ogrDrivers (readOGR), 22
- ogrFIDs (readOGR), 22
- ogrInfo (readOGR), 22
- open.SpatialGDAL
(SpatialGDAL-class), 28
- print.CRS (CRS-class), 2
- print.GDALobj (readGDAL), 19
- print.ogrinfo (readOGR), 22
- print.summary.SpatialGDAL
(SpatialGDAL-class), 28

- project, [16](#)
- projInfo, [18](#)
- projNAD (*projInfo*), [18](#)
- putRasterData
 - (*GDALDataset-class*), [5](#)
- readGDAL, [13](#), [19](#)
- readOGR, [22](#), [34](#)
- readShapePoly, [24](#)
- RGB2PCT, [25](#)
- saveDataset, [14](#)
- saveDataset (*GDALDataset-class*), [5](#)
- saveDatasetAs, [14](#)
- saveDatasetAs
 - (*GDALDataset-class*), [5](#)
- SGDF2PCT, [26](#)
- show, CRS-method (*CRS-class*), [2](#)
- showWKT, [27](#)
- Spatial-class, [29](#)
- SpatialGDAL-class, [28](#)
- SpatialGDALWrite-class
 - (*SpatialGDAL-class*), [28](#)
- SpatialGridDataFrame-class, [13](#), [30](#)
- SpatialGridDataFrame-class, [19](#), [20](#), [29](#)
- SpatialLinesDataFrame-class, [24](#)
- SpatialPixels-class, [29](#)
- SpatialPixelsDataFrame-class, [19](#)
- SpatialPoints, [29](#)
- SpatialPointsDataFrame-class, [24](#)
- SpatialPointsDataFrame-class, [20](#)
- SpatialPolygonsDataFrame-class, [24](#)
- spTransform
 - (*spTransform-methods*), [30](#)
- spTransform, ANY-method
 - (*spTransform-methods*), [30](#)
- spTransform, SpatialGridDataFrame, CRS-method
 - (*spTransform-methods*), [30](#)
- spTransform, SpatialLines, CRS-method
 - (*spTransform-methods*), [30](#)
- spTransform, SpatialLinesDataFrame, CRS-method
 - (*spTransform-methods*), [30](#)
- spTransform, SpatialPixelsDataFrame, CRS-method
 - (*spTransform-methods*), [30](#)
- spTransform, SpatialPoints, CRS-method
 - (*spTransform-methods*), [30](#)
- spTransform, SpatialPointsDataFrame, CRS-method
 - (*spTransform-methods*), [30](#)
- spTransform, SpatialPolygons, CRS-method
 - (*spTransform-methods*), [30](#)
- spTransform, SpatialPolygonsDataFrame, CRS-method
 - (*spTransform-methods*), [30](#)
- spTransform-methods, [17](#)
- spTransform-methods, [30](#)
- spTransform.SpatialLines
 - (*spTransform-methods*), [30](#)
- spTransform.SpatialLinesDataFrame
 - (*spTransform-methods*), [30](#)
- spTransform.SpatialPoints
 - (*spTransform-methods*), [30](#)
- spTransform.SpatialPointsDataFrame
 - (*spTransform-methods*), [30](#)
- spTransform.SpatialPolygons
 - (*spTransform-methods*), [30](#)
- spTransform.SpatialPolygonsDataFrame
 - (*spTransform-methods*), [30](#)
- sub.GDROD
 - (*GDALReadOnlyDataset-methods*), [13](#)
- summary, SpatialGDAL-method
 - (*SpatialGDAL-class*), [28](#)
- toSigned (*GDALRasterBand-class*), [9](#)
- toUnsigned
 - (*GDALRasterBand-class*), [9](#)
- vec2RGB (*SGDF2PCT*), [26](#)
- writeGDAL (*readGDAL*), [19](#)
- writeOGR, [32](#)