Package ‘rgdal’

September 28, 2017

Title  Bindings for the ‘Geospatial’ Data Abstraction Library
Version  1.2-12
Date  2017-09-25
Depends  R (>= 3.3.0), methods, sp (>= 1.1-0)
Imports  grDevices, graphics, stats, utils
LinkingTo  sp
Description  Provides bindings to the ‘Geospatial’ Data Abstraction Library (‘GDAL’) (>= 1.6.3) 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. Windows and Mac Intel OS X binaries (including ‘GDAL’, ‘PROJ.4’ and ‘Expat’) are provided on ‘CRAN’.

License  GPL (>= 2)

NeedsCompilation  yes
Author  Roger Bivand [cre, aut],
        Tim Keitt [aut],
        Barry Rowlingson [aut, ctb],
        Edzer Pebesma [ctb],
        Michael Sumner [ctb],
        Robert Hijmans [ctb],
        Even Rouault [ctb]
Maintainer  Roger Bivand <Roger.Bivand@nhh.no>
Repository  CRAN
Date/Publication  2017-09-28 07:45:15 UTC
**R topics documented:**

- closeDataset-methods ................................................. 2
- CRS-class .............................................................. 3
- displayDataset ......................................................... 5
- GDALcall ............................................................... 6
- GDALDataset-class .................................................. 7
- GDALDriver-class .................................................... 8
- GDALMajorObject-class ............................................. 10
- GDALRasterBand-class .............................................. 11
- GDALReadOnlyDataset-class ...................................... 13
- GDALReadOnlyDataset-methods ................................... 15
- GDALTransientDataset-class ..................................... 17
- GridsDatums .......................................................... 18
- llgridlines ............................................................ 19
- make_EPSG ............................................................ 20
- nor2k ................................................................. 21
- project ............................................................... 22
- projInfo .............................................................. 24
- readGDAL .............................................................. 25
- readOGR ............................................................... 31
- RGB2PCT ............................................................... 36
- SGDF2PCT ............................................................. 37
- showWKT .............................................................. 39
- SpatialGDAL-class .................................................. 40
- spTransform-methods ............................................... 42
- writeOGR ............................................................. 47

| Index | 51 |

---

### closeDataset-methods  
**closeDataset methods**

#### Description

Methods for closing GDAL datasets, used internally

#### Usage

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

#### Arguments

- **dataset**  
  GDAL dataset
CRS-class

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.

Usage

checkCRSArgs(uprojargs)

Arguments

uprojargs character string PROJ.4 projection arguments

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). See GridsDatums and showEPSG for help in finding CRS definitions.

The 4.9.1 release of PROJ.4 omitted a small file of defaults, leading to reports of “major axis or radius = 0 or not given” errors. From 0.9-3, rgdal checks for the presence of this file (proj_def.dat), and if not found, and under similar conditions to those used by PROJ.4, adds “+ellps=WGS84” to the input string being checked by checkCRSArgs. The “+no_defs” tag ignores the file of defaults, and the default work-around implemented to get around this problem; strings including “init” and “datum” tags also trigger the avoidance of the work-around. Now messages are issued when a candidate CRS is checked; they may be suppressed using suppressMessages.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References

http://proj.maptools.org/

Examples

```r
CRSargs(CRS("+proj=longlat"))
try(CRS("+proj=longlat +no_defs"))
CRSargs(CRS("+proj=longlat +datum=NAD27"))
CRSargs(CRS("+init=epsg:4267"))
CRSargs(CRS("+init=epsg:26978"))
CRSargs(CRSpaste("+proj=stere +lat_0=52.15616055555555",
"+lon_0=5.387638888888889 +k=0.999908 +x_0=155000 +y_0=463000 +ellps=bessel",
"+towgs84=565.237,50.0087,465.658,-0.406857,0.350733,-1.87035,4.0812",
"+units=m")))
# see http://trac.osgeo.org/gdal/ticket/1987
CRSargs(CRS("+init=epsg:28992"))
crs <- CRS("+init=epsg:28992")
CRSargs(CRSargs(crs))
library(sp)
data(meuse)
coordinates(meuse) <- c("x", "y")
proj4string(meuse) <- CRS("+init=epsg:28992")
CRSargs(CRS(proj4string(meuse)))
```
**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,
bond = 1, col = NULL, reset.par = TRUE, max.dim = 500, ...)`

**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

```r
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)
```

---

**Description**

These functions allow more direct access to some of the rgdal C API. These are advanced methods intended for package developers only.

**Usage**

```r
GDALcall(object, option, ...)
rawTransform(projfrom, projto, n, x, y, z=NULL)
```

**Arguments**

- **object**: GDALTransientDataset (option = 'SetGeoTransform', 'SetProject') or GDALRasterBand (the other options)
- **option**: character. One of 'SetGeoTransform', 'SetProject', 'SetNoDataValue', 'SetStatistics', 'SetRasterColorTable' or 'SetCategoryNames')
- **...**: additional arguments. The values to be set
- **projfrom**: character. PROJ.4 coordinate reference system (CRS) description
- **projto**: character. PROJ.4 CRS description
- **n**: number of coordinates
- **x**: x coordinates
- **y**: y coordinates
- **z**: z coordinates

**Value**

GDALcall does not return anything. rawTransform returns a matrix of transformed coordinates.

**Author(s)**

Robert Hijmans
GDALDataset-class

Class “GDALDataset”

Description

GDALDataset extends GDALReadOnlyDataset-class with data update commands.

Usage

```r
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 `length(dim(rasterData)) = 2`
- **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; will be normalized with `normalizePath`
- **returnNewObj**: Until and including 0.5-27, `saveDataset` 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.
- **driver**: GDAL driver name to use for saving raster data object
- **strict**: TRUE if the copy must be strictly equivalent, or more normally FALSE indicating that the copy may adapt as needed for the output format
- **options**: Driver specific options (currently passed to GDAL)
- **fname**: 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(str = "--version")`
- `getGDALCheckVersion()`
- `getCPLConfigOption(ConfigOption)`
- `setCPLConfigOption(ConfigOption, value)`
**GDALDriver-class**

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>driver</td>
<td>An object inheriting from class <code>GDALDriver</code></td>
</tr>
<tr>
<td>str</td>
<td>A string, may be one of &quot;--version&quot;, &quot;VERSION_NUM&quot;, &quot;RELEASE_DATE&quot;, &quot;RELEASE_NAME&quot;</td>
</tr>
<tr>
<td>configoption</td>
<td>CPL configure option documented in <a href="http://trac.osgeo.org/gdal/wiki/ConfigOptions">http://trac.osgeo.org/gdal/wiki/ConfigOptions</a> and elsewhere in GDAL source code</td>
</tr>
<tr>
<td>value</td>
<td>a string value to set a CPL option; NULL is used to unset the CPL option</td>
</tr>
</tbody>
</table>

**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.

getGDALCheckVersion: checks the version of the GDAL headers used when building the package (GDAL\_VERSION\_MAJOR, GDAL\_VERSION\_MINOR) - if the two versions differ, problems may arise (the C++ API/ABI may have changed), and rgdal should be re-installed

**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**

<table>
<thead>
<tr>
<th>Slot</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>Object of class &quot;externalptr&quot;, from class &quot;GDALMajorObject&quot;, used internally; not for public consumption</td>
</tr>
</tbody>
</table>

**Extends**

Class "GDALMajorObject", directly.

**Methods**

initialize signature(.Object = "GDALDriver"): drivername: 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 bundled with the package.

**Author(s)**

Timothy H. Keitt, modified by Roger Bivand
GDALMajorObject-class

Description

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

Usage

ggetDescription(object)

Arguments

object an object inheriting from "GDALMajorObject"

Details

ggetDescription: 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
GDALRasterBand-class

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, list_out=FALSE)

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

getProjectionRef(dataset, OVERRIDE_PROJ_DATUM_WITH_TOWGS84 = NULL)

getRasterBand(dataset, band = 1)

getRasterBlockSize(raster)

toSigned(x, base)

toUnsigned(x, base)

get_OVERRIDE_PROJ_DATUM_WITH_TOWGS84()

set_OVERRIDE_PROJ_DATUM_WITH_TOWGS84(value)

References
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

Arguments

- **dataset**: An object inheriting from class `GDALReadOnlyDataset`
- **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; 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
- **output.dim**: Number of rows and columns in the output data; if smaller than region.dim the data will be subsampled
- **interleave**: Element and row stride while reading data; rarely needed
- **as.is**: If false, scale the data to its natural units; if the case of thematic data, return the data as factors
- **list_out**: default FALSE, return array, if TRUE, return a list of vector bands
- **raster**: An object of class GDALRasterBand
- **x**: integer variable for conversion
- **base**: If Byte input, 8, if Int16 or UInt16, 16
- **OVERRIDE_PROJ_DATUM_WITH_TOWGS84**: logical value, default NULL, which case the cached option set by set_OVERRIDE_PROJ_DATUM_WITH_TOWGS84 is used. Ignored if the GDAL version is less than “1.8.0” or if the CPLConfigOp- tion variable is already set
- **value**: logical value to set OVERRIDE_PROJ_DATUM_WITH_TOWGS84

Details

- **getRasterData**: retrieves data from the dataset as an array or list of bands; will try to convert relevant bands to factor if category names are available in the GDAL driver when returning a list.
- **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
**GDALReadOnlyDataset-class**

**Extends**

Class "GDALMajorObject", directly.

**Methods**

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

**Note**

The OVERRIDE_PROJ_DATUM_WITH_TOWGS84 argument is used to revert GDAL behaviour to pre-1.8.0 status; from 1.8.0, any input datum may be discarded if the input also includes a towgs84 tag in conversion to the PROJ.4 representation, see [http://trac.osgeo.org/gdal/ticket/4880](http://trac.osgeo.org/gdal/ticket/4880) and [http://lists.osgeo.org/pipermail/gdal-dev/2012-November/034550.html](http://lists.osgeo.org/pipermail/gdal-dev/2012-November/034550.html). The cached value of OVERRIDE_PROJ_DATUM_WITH_TOWGS84 will also be used in open.SpatialGDAL, sub.GDROD, and asGDALROD_SGDF, which do not have a suitable argument.

**Author(s)**

Timothy H. Keitt, modified by Roger Bivand

**See Also**

See also **GDALDriver-class**, **GDALDataset-class**, **GDALTransientDataset-class**

**Examples**

```r
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

```r
gdalNcloseHdatasetI
gdalNopenHfilenameL readNonly = TRUE, silent=FALSE,
        allowedDrivers = NULL, options=NULL)
getDriver(dataset)
```

colorTable(dataset, band = 1)
getGeoTransFunc(dataset)

Arguments

dataset An object inheriting from class 'GDALReadOnlyDataset'
filename A string giving the file to read from
band The band number (1-based) to read from
readNonly A logical flag indicating whether to open the file as a GDALReadOnlyDataset or
             as a writable GDALDataset
silent logical; if TRUE, comment and non-fatal CPL driver errors suppressed
allowedDrivers a character vector of suggested driver short names may be provided starting from
               GDAL 2.0
options open options may be passed to raster drivers starting from GDAL 2.0; very few
          drivers support these options

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.gdal.org/

See Also

See also `GDALDriver-class`, `GDALDataset-class`, `GDALTransientDataset-class`.

Examples

```r
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"))[1,], 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)
```

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 Pebesma

See Also

See also `readGDAL GDALDriver-class, GDALDataset-class, GDALTransientDataset-class, SpatialGridDataFrame-class`

Examples

```r
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)
```
GDALTransientDataset-class

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())

---

GridsDatums
Grids and Datums PE&RS listing

Description
A data.frame of years and months of Grids & Datums column publications by country and country code.

Usage
data("GridsDatums")

Format
A data frame with 207 observations on the following 4 variables.
country name of PE&RS column
month issue month
year publication year
ISO ISO code for country

Details
The journal *Photogrammetric Engineering & Remote Sensing*, run by the American Society for Photogrammetry and Remote Sensing (ASPRS), began publishing a more-or-less monthly column on the spatial reference systems used in different countries, including their datums. The column first appeared in September 1997, and continued until November 2015. Some also cover other topics, such as world and Martian spatial reference systems. They are written by Clifford J. Mugnier, Louisiana State University, Fellow Emeritus ASPRS. To access the columns, visit [http://www.asprs.org/Grids-Datums.html](http://www.asprs.org/Grids-Datums.html), and choose the 1997–2008, 2009–2010, or 2011–2015 subset as appropriate.
llgridlines

Plot long-lat grid over projected data

Usage

llgridlines(obj, easts, norths, ndiscr = 20, lty = 2, offset=0.5, side="WS", llcrs = "+proj=longlat +datum=WGS84", plotLines = TRUE, plotLabels = TRUE, ...)

Arguments

obj  object, deriving from Spatial having projection specified

easts  numeric; see gridlines

norths  numeric; see gridlines

ndiscr  numeric; see gridlines

offset  numeric; see gridat

side  character, default “WS”; see gridat; available from sp 0.9-84

lty  line type to be used for grid lines

llcrs  proj4string of longitude - latitude

plotLines  logical; plot lines?

plotLabels  logical; plot labels?

...  graphics arguments passed to plot function for lines and text function for labels

Value

none; side effect is that grid lines and labels are plotted

See Also

is.projected, CRS-class
Examples

```r
data(meuse)
coordinates(meuse) = ~x+y
proj4string(meuse) <- CRS("+init=epsg:28992")
plot(meuse)
llgridlines(meuse, lty=3)
plot(meuse)
llgridlines(meuse, lty=3, side = "EN", offset = 0.2)
```

**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**

- `file` file name of the file matching EPSG codes and PROJ.4 arguments, should usually be autodetected

**Value**

returns a data frame with columns:

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

**Note**


**Author(s)**

Roger Bivand
References

http://www.epsg.org/

Examples

```r
EPSG <- make_EPSG()
EPSG[grep("Oslo", EPSG$note), 1:2]
EPSC[grep("Poland", EPSG$note), 1:2]
EPSG[grep("Amersfoort", EPSG$note), 1:2]
EPSG[grep("North Carolina", EPSG$note), 1:2]
EPSG[202, 3]
```

```
nor2k                Norwegian peaks over 2000m
```

Description

Norwegian peaks over 2000m, 3D SpatialPoints data.

Usage

```r
data(nor2k)
```

Format

The format is: Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots .@ data .@ data.frame': 300 obs. of 3 variables: ...$ Nr. : int [1:300] 1 2 3 4 5 6 7 8 9 10 ... ...$ Navn : chr [1:300] "Galdhøpiggen" "Glittertinden" "Skagastølstinden, Store (Storen)" "Styggedalstinden, Store, ?ttospenen" ... ...$ Kommune: chr [1:300] "Lom" "Lom" "Luster / Ardal" "Luster" ... ...@ coords.nrs : num(0) ...@ coords : num [1:300, 1:3] 463550 476550 439850 441450 441100 ... ... - attr(*, "dimnames")=List of 2 ... ...$ : NULL ... ...$ : chr [1:3] "East" "North" "Height" ... @ bbox : num [1:3, 1:2] 404700 6804200 2001 547250 6910050 ... ... - attr(*, "dimnames")=List of 2 ... ...$ : chr [1:3] "East" "North" "Height" ... ...$ : chr [1:2] "min" "max" ... @ proj4string: Formal class 'CRS' [package "sp"] with 1 slots ... ... - projargs: chr "+proj=utm +zone=32 +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"

Details

Norwegian peaks over 2000m, coordinates in EUREF89/WGS84 UTM32N, names not fully updated, here converted to ASCII.

Source

project

**Project 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**

```r
project(xy, proj, inv = FALSE, use_ob_tran=FALSE, legacy=TRUE)
```

**Arguments**

- **xy**
  - 2-column matrix of coordinates
- **proj**
  - 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 `+<arg>=<value>` strings, and successive such strings can only be separated by blanks.
- **inv**
  - default FALSE, if TRUE inverse projection to geographical coordinates
- **use_ob_tran**
  - default FALSE, if TRUE and `"+proj=ob_tran"`, use General Oblique Transformation with internalised from/to projection reversal; the user oblique transforms forward rather than inverse.
- **legacy**
  - default TRUE, if FALSE, use transform C functions (enforced internally for Windows 32-bit platforms)

**Details**

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

Note that from PROJ.4 4.9.3, the definition of UTM is changed from TMERC to ETMERC; see example.

**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.

**Examples**

```r
data(nor2k)
summary(nor2k)
## maybe str(nor2k) ; plot(nor2k) ... ```
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=llc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84")
res1 <- project(res, "+proj=llc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84",
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)
crds <- matrix(data=c(5.05, 48.52), ncol=2)
a <- project(crdts, paste("+proj=ob_tran +o_proj=longlat",
"+o_lon_p=-162 +o_lat_p=39.25 +lon_0=180 +ellps=sphere +no_defs"),
use_ob_tran=TRUE)
a #should be (-5.917698, -1.87195)
project(a, paste("+proj=ob_tran +o_proj=longlat",
"+o_lon_p=-162 +o_lat_p=39.25 +lon_0=180 +ellps=sphere +no_defs"),
inv=TRUE, use_ob_tran=TRUE)
#added after posting by Martin Ivanov

getPROJ4VersionInfo()
# Test for UTM == TMCRC (<= 4.9.2) or UTM == ETMCRC (> 4.9.2)
nhh <- matrix(c(5.304234, 60.422311), ncol=2)
nhh_utm_32N_P4 <- project(nhh, "+init=epsg:3844")
nhh_tmerc_P4 <- project(nhh, paste("+proj=tmerc +k=0.9996 +lon_0=9",
"+x_0=500000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"))
nhh_etmerc_P4 <- project(nhh, paste("+proj=etmerc +k=0.9996 +lon_0=9",
"+x_0=500000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"))
all.equal(nhh_utm_32N_P4, hh_tmerc_P4, tolerance=le-9, scale=1)
# UTM == TMRC: PROJ <= 4.9.2
all.equal(nhh_utm_32N_P4, hh_etmerc_P4, tolerance=le-9, scale=1)
# UTM == ETMCRC: PROJ > 4.9.2
unis <- matrix(c(15.653453, 78.222504), ncol=2)
unis_utm_33N_P4 <- project(unis, "+init=epsg:3045")
unis_tmerc_P4 <- project(unis, paste("+proj=tmerc +k=0.9996 +lon_0=15",
"+x_0=500000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"))
unis_etmerc_P4 <- project(unis, paste("+proj=etmerc +k=0.9996 +lon_0=15",
"+x_0=500000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"))
all.equal(unis_utm_33N_P4, unis_tmerc_P4, tolerance=1e-9, scale=1)
# UTM == TMEC: PROJ4 <=4.9.2
all.equal(unis_utm_33N_P4, unis_etmerc_P4, tolerance=1e-9, scale=1)
# UTM == ETMEC: PROJ4 > 4.9.2
# available projections and their inverses if provided
# For >=4.9.3 returns non-finite points rather than needing crash protection
getPROJ4VersionInfo()
projs <- as.character(projInfo()$name)
res <- logical(length(projs))
names(res) <- projs
msgs <- character(length(projs))
names(msgs) <- projs
owarn <- options("warn")$warn
options(warn=2L)
for (i in seq(along=res)) {
    iprs <- paste("*proj=" , projs[i], sep="")
    xy <- try(project(cbind(0, 0), iprs, legacy=TRUE), silent=TRUE)
    if (class(xy) == "try-error") {
        res[i] <- NA
        msgs[i] <- paste("fwd:", strsplit(xy, "\n")[[1]][2])
    } else {
        out <- try(project(xy, iprs, inv=TRUE, legacy=TRUE), silent=TRUE)
        if (class(out) == "try-error") {
            res[i] <- NA
            msgs[i] <- paste("inv:", strsplit(out, "\n")[[1]][2])
        } else res[i] <- isTRUE(all.equal(cbind(0,0), out))
    }
}
options(warn=owarn)
df <- data.frame(res=unname(res), msgs=unname(msgs), row.names=names(res))
# projection and inverse projection failures
# fwd: missing parameters
# inv: mostly inverse not defined
df[is.na(df$res),]
# inverse not equal to input
# (see http://lists.maptools.org/pipermail/proj/2011-December/005615.html)
df[is.na(df$res) & !df$res,]
# inverse equal to input
row.names(df[!is.na(df$res) & df$res,])

---

**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", "units")`. `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**

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

**Arguments**

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

**Details**

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

**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**

```r
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), silent = FALSE, OVERRIDE_PROJ_DATUM_WITH_TOWGS84=NULL, allowedDrivers = NULL, options=NULL)

asSGDF_GROD(x, offset, region.dim, output.dim, p4s=NULL, ..., half.cell=c(0.5,0.5), OVERRIDE_PROJ_DATUM_WITH_TOWGS84=NULL)

writeGDAL(dataset, fname, drivername = "GTiff", type = "Float32", mvFlag = NA, options=NULL, copy_drivername = "GTiff", setStatistics=FALSE, colorTables = NULL, catNames=NULL)

create2GDAL(dataset, drivername = "GTiff", type = "Float32", mvFlag = NA, options=NULL, fname = NULL, setStatistics=FALSE, colorTables = NULL, catNames=NULL)

GDALinfo(fname, silent=FALSE, returnRAT=FALSE, returnCategoryNames=FALSE, returnStats=TRUE, returnColorTable=FALSE, OVERRIDE_PROJ_DATUM_WITH_TOWGS84=NULL, returnScaleOffset=TRUE, allowedDrivers = NULL, options=NULL)

GDALSpatialRef(fname, silent=FALSE, OVERRIDE_PROJ_DATUM_WITH_TOWGS84=NULL, allowedDrivers = NULL, options=NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fname</td>
<td>file name of grid map; in create2GDAL provides a way to pass through a file name with driver-required extension for sensitive drivers</td>
</tr>
<tr>
<td>x</td>
<td>A GDALReadOnlyDataset object</td>
</tr>
<tr>
<td>offset</td>
<td>Number of rows and columns from the origin (usually the upper left corner) to begin reading from; presently ordered (y,x) - this may change</td>
</tr>
<tr>
<td>region.dim</td>
<td>The number of rows and columns to read from the dataset; presently ordered (y,x) - this may change</td>
</tr>
<tr>
<td>output.dim</td>
<td>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</td>
</tr>
<tr>
<td>band</td>
<td>if missing, all bands are read</td>
</tr>
<tr>
<td>p4s</td>
<td>PROJ4 string defining CRS, if default (NULL), the value is read from the GDAL data set</td>
</tr>
<tr>
<td>half.cell</td>
<td>Used to adjust the intra-cell offset from corner to centre, usually as default, but may be set to c=(0,0) if needed; presently ordered (y,x) - this may change</td>
</tr>
<tr>
<td>silent</td>
<td>logical; if TRUE, comment and non-fatal CPL driver errors suppressed</td>
</tr>
<tr>
<td>OVERRIDE_PROJ_DATUM_WITH_TOWGS84</td>
<td>logical value, default NULL, which case the cached option set by set_OVERRIDE_PROJ_DATUM_WITH_TOWGS84 is used. Ignored if the GDAL version is less than “1.8.0” or if the CPLConfigOption variable is already set; see getProjectionRef for further details</td>
</tr>
<tr>
<td>allowedDrivers</td>
<td>a character vector of suggested driver short names may be provided starting from GDAL 2.0</td>
</tr>
</tbody>
</table>
... arguments passed to either `getRasterData`, or `getRasterTable`, depending on rotation angles (see below); see the rgdal documentation for the available options (subsetting etc.)

dataset object of class `SpatialGridDataFrame-class` or `SpatialPixelsDataFrame-class`

drivername, `copy_drivername` GDAL driver name; if the chosen driver does not support dataset creation, an attempt is made to use the `copy_drivername` driver to create a dataset, and `copyDataset` to copy to the target driver.

type GDAL write data type, one of: `‘Byte’`, `‘Int16’`, `‘Int32’`, `‘Float32’`, `‘Float64’`; `‘UInt16’`, `‘UInt32’` are available but have not been tested.

mvFlag default NA, missing value flag for output file; the default value works for `‘Int32’`, `‘Float32’`, `‘Float64’`, but suitable in-range value that fits the data type should be used for other data types, for example 255 for `‘Byte’`, -32768 for `‘Int16’`, and so on; see Details below.

options driver-specific options to be passed to the GDAL driver; only available for opening datasets from GDAL 2.0; see copying and creation details below.

setStatistics default FALSE, if TRUE, attempt to set per-band statistics in the output file (driver-dependent)

colortables default NULL, if not NULL, a list of length equal to the number of bands, with NULL components for bands with no color table, or either an integer matrix of red, green, blue and alpha values (0-255), or a character vector of colours. The number of colours permitted may vary with driver.

catNames default NULL, if not NULL, a list of length equal to the number of bands, with NULL components for bands with no category names, or a string vector of category names.

returnRAT default FALSE, if TRUE, return a list with a Raster Attribute Table or NULL for each band.

returnCategoryNames default FALSE, if TRUE, return a list with a character vector of CategoryNames or NULL for each band.

returnStats default TRUE, return band-wise statistics if available (from 0.7-20 set to NA if not available).

returnColorTable default FALSE; if TRUE return band-wise colour tables in a list attribute “ColorTables”

returnScaleOffset default TRUE, return a matrix of bandwise scales and offsets.

Details

In `writeGDAL`, if types other than `‘Int32’`, `‘Float32’`, `‘Float64’` are used, the “mvFlag” argument should be used to set a no data value other than the default NA. Note that the flag only replaces NA values in the data being exported with the value of the argument - it does not mark data values equal to “mvFlag” as missing. The value is stored in the file being written in driver-specific ways, and may be used when the file is read. When the default “mvFlag=NA” is used, no NoDataValue is written to the file, and the input data is written as is.
Also in writeGDAL, the "options" argument may be used to pass a character vector of one or more options to the driver, for example 'options="INTERLEAVE=PIXEL"', or 'options=c("INTERLEAVE=PIXEL", "COMPRESS=DEFLATE")'. Typical cases are given in the examples below; it may also be necessary in some cases to escape quotation marks if included in the string passed to the driver.

Value

readGDAL 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.

Warning

Some raster files may have an erroneous positive y-axis resolution step, leading to the data being flipped on the y-axis. readGDAL will issue a warning: Y axis resolution positive, examine data for flipping, when the step is positive, but this need not mean that the data are flipped. Examine a display of the data compared with your knowledge of the file to determine whether this is the case (one known case is interpolation files created under Qgis up to February 2010 at least). To retrieve the correct orientation, use flipVertical.

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 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\nintegers 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)
spplot(y, names.attr = c("red","green","blue"),
  col.regions = grey(c(0:100/100)),
  main = "example of three-layer (RGB) raster image", as.table=TRUE)
data(meuse.grid)
grided = meuse.grid) = -x+y
proj4string(meuse.grid) = CRS("+init=epsg:28992")
fn <- tempfile()
writeGDAL(meuse.grid["dist"], fn)
GDALinfo(fn)
writeGDAL(meuse.grid["dist"], fn, setStatistics=TRUE)
GDALinfo(fn)
mg2 <- readGDAL(fn)
proj4string(mg2)
SP2GTIF <- readGDAL(system.file("pictures/SP2GTIF.TIF", package = "rgdal")[[1]], output.dim=c(100,100))
summary(SP2GTIF)
image(SP2GTIF, col=grey(1:99/100))
GDALinfo(system.file("pictures/cea.tif", package = "rgdal")[[1]])
GDALSpatialRef(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_snpad83.tif", package = "rgdal")[[1]]
erdas_snpad83 <- readGDAL(fn, offset=c(50, 100), region.dim=c(400, 400),
  output.dim=c(100,100))
summary(erdas_snpad83)
image(erdas_snpad83, col=grey(1:99/100))
erdas_snpad83a <- readGDAL(fn, offset=c(50, 100), region.dim=c(400, 400))
bbox(erdas_snpad83)
bbox(erdas_snpad83a)
gridparameters(erdas_snpad83)
gridparameters(erdas_snpad83a)
tf <- tempfile()
writeGDAL(erdas_snpad83, tf, drivername="GTiff", type="Byte", options=NULL)
all.equal(erdas_snpad83, readGDAL(tf))
writeGDAL(erdas_snpad83, tf, drivername="GTiff", type="Byte",
  options="INTERLEAVE=PIXEL")
all.equal(erdas_snpad83, readGDAL(tf))
writeGDAL(erasds_snpad83, tf, drivername="GTiff", type="Byte",
options=c("INTERLEAVE=PIXEL", "COMPRESS=DEFLATE"))
all.equal(erasds_snpad83, readGDAL(tf))

x <- GDAL.open(system.file("pictures/erasds_snpad83.tif", package = "rgdal"))[1]
erasds_snpad83 <- asSGDF_GROD(x, output.dim=c(100,100))
GDAL.close(x)
summary(erasds_snpad83)
image(erasds_snpad83, col=grey(1:99/100))

tf <- tempfile()
xx <- create2GDAL(erasds_snpad83, type="Byte")
xxx <- copyDataset(xx, driver="PNG")
saveDataset(xxx, tf)
GDAL.close(xx)
GDAL.close(xxx)
GDALinfo(tf)

tf2 <- tempfile()
writeGDAL(erasds_snpad83, tf2, drivername="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", drivername="PNG")
tl <- readGDAL(tf)
image(tl, red=1, green=2, blue=3)
title(main="output PNG RGB")
opar(opar)

t0 <- meuse.grid["ffreq"]
fullgrid(t0) <- TRUE
t0$ffreq <- as.integer(t0$ffreq)-1
# convert factor to zero-base integer
CT <- c("red", "orange", "green", "transparent")
CT
cN <- c("annual", "2-5 years", "infrequent")
tf <- tempfile()
writeGDAL(t0, tf, type="Byte", colorTable=list(CT), catNames=list(cN),
mvFlag=3L)
attr(GDALinfo(tf, returnStats=FALSE, returnCategoryNames=TRUE),
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 `boundsNshp` 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**

```r
readOGR(dsn, layer, verbose = TRUE, p4s=NULL,
stringsAsFactors=default.stringsAsFactors(),
drop_unsupported_fields=FALSE,
pointDropZ=FALSE, dropNULLGeometries=TRUE,
useC=TRUE, disambiguateFIDs=FALSE, addCommentsToPolygons=TRUE,
encoding=NULL, use_iconv=FALSE, swapAxisOrder=FALSE, require_geomType = NULL,
integer64="no.loss", GDAL1_integer64_policy=FALSE)
ogrInfo(dsn, layer, encoding=NULL,
use_iconv=FALSE, swapAxisOrder=FALSE, require_geomType = NULL)
ogrFIDs(dsn, layer)
ogrDrivers()
OGRSpatialRef(dsn, layer)
ogrListLayers(dsn)
## S3 method for class 'ogrinfo'
print(x, ...)
```

**Arguments**

- **dsn**
  - data source name (interpretation varies by driver — for some drivers, dsn is a file name, but may also be a folder)

- **layer**
  - layer name (varies by driver, may be a file name without extension). From rgdal 1.2.*, layer may be missing, in which case ogrListLayers examines the dsn, and fails if there are no layers, silently reads the only layer if only one layer is found, and reads the first layer if multiple layers are present, issuing a warning that layer should be given explicitly.

- **verbose**
  - report progress

- **p4s**
  - PROJ4 string defining CRS, if default NULL, the value is read from the OGR data set

- **stringsAsFactors**
  - logical: should character vectors be converted to factors? The ‘factory-fresh’ default is TRUE, but this can be changed by setting `options(stringsAsFactors = FALSE)` (see link[base]{data.frame}).

- **drop\_unsupported\_fields**
  - default FALSE, if TRUE skip fields other than String, Integer, and Real; Date, Time and DateTime are converted to String

- **pointDropZ**
  - default FALSE, if TRUE, discard third coordinates for point geometries; third coordinates are always discarded for line and polygon geometries
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.

From 0.7-24, if the layer has no fields, a single field containing the FID values is placed in the data slot of the returned object.

From 0.7-24, attempts are begun to provide users with arguments to control reading from OGR/shapefile driver when the encoding is inappropriate (especially the setting of LDID in shapefile DBFs, and the SHAPE_ENCODING environment variable).

While there is no certainty, newer drivers such as KML, GML, SQLite and Geopackage (GPKG) may encode string fields as UTF-8. Users are advised to explore this on a case to case basis using Encoding on string fields of input objects.

Because of the representation of DateTime data in OGR, decimal seconds in input data are rounded to integer seconds, see: http://trac.osgeo.org/gdal/ticket/2680.

Because some drivers support reading string, integer and real list fields, support has been introduced into ogrInfo from version 0.9-1 to report their presence and the maximum counts of list items. This may lead to the introduction of the ~splitlistfields facility from the command line utility ogrinfo. In addition, ogrInfo reports that there are no features when counting FIDs in a while loop over features in ogrFIDs never enters the loop, despite the layer feature count reporting at least one feature.

Author(s)

Roger Bivand
References


See Also

SpatialPointsDataFrame-class, SpatialLinesDataFrame-class, SpatialPolygonsDataFrame-class, readShapePoly, iconv

Examples

ngrDrivers()

dsn <- system.file("vectors", package = "rgdal")[1]
ogrListLayers(dsn)
ogrInfo(dsn)
ogrInfo(dsn=dsn, layer="cities")

owd <- getwd()
setwd(dsn)
ogrInfo(dsn="cities.shp")
ogrInfo(dsn="cities.shp", layer="cities")
setwd(owd)

ow <- options("warn")$warn
options("warn"=1)
cities <- readOGR(dsn=dsn, layer="cities")

str(slot(cities, "data"))
cities$POPULATION <- type.convert(as.character(cities$POPULATION),
    na.strings="-99", numerals="no.loss")
str(slot(cities, "data"))
cities <- readOGR(dsn=dsn, layer="cities", GDAL1_integer64_policy=TRUE)

options("warn"=ow)

summary(cities)
table(Encoding(as.character(cities$NAME)))
ogrInfo(dsn=dsn, layer="kiritimati_primary_roads")
OGRSpatialRef(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")
OGRSpatialRef(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.qml", 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)
sapply(as(ps_cant_31, "data.frame"), class)

ps_cant_31 <- readOGR(dsn=dsn, layer="ps_cant_31", stringsAsFactors=FALSE)
summary(ps_cant_31)
sapply(as(ps_cant_31, "data.frame"), class)
dsn <- system.file("vectors/Up.tab", package = "rgdal")
ogrInfo(dsn=dsn, layer="Up")
Up <- readOGR(dsn=dsn, layer="Up")
summary(Up)
dsn <- system.file("vectors/test_trk2.gpx", package = "rgdal")
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")
ogrInfo(dsn=dsn, layer="trin_inca_pl03")
birds <- readOGR(dsn=dsn, layer="trin_inca_pl03")
summary(birds)
dsn <- system.file("vectors/PacoursIKA2.TAB", package = "rgdal")
try(ogrInfo(dsn, "PacoursIKA2"))
ogrInfo(dsn, "PacoursIKA2", require_geomType="wkbPoint")
plot(readOGR(dsn, "PacoursIKA2", require_geomType="wkbLineString"), col="red")
plot(readOGR(dsn, "PacoursIKA2", require_geomType="wkbPoint"), add=TRUE)
odir <- getwd()
setwd(system.file("vectors", package = "rgdal")
ow <- options("warn")$warn
options("warn"=1)
ogrInfo("test64.vrt", "test64")
str(readOGR("test64.vrt", "test64", verbose=FALSE, integer64="allow.loss")$val)
str(readOGR("test64.vrt", "test64", verbose=FALSE, integer64="warn.loss")$val)
str(readOGR("test64.vrt", "test64", verbose=FALSE, integer64="no.loss")$val)
str(readOGR("test64.vrt", "test64", verbose=FALSE, stringsAsFactors=FALSE, integer64="no.loss")$val)
setwd(odir)
options("warn"=ow)

---

### 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

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

### Arguments

- **x**: a three-band GDALReadOnlyDataset object
- **band**: a vector of numbers, recycled up to 3 in length
- **driver.name**: default MEM
ncolors          a number of colours between 2 and 256
set.ctab        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 a either GDAL.TransientDataset or a list of a GDAL.TransientDataset 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)

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

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

Value

The value returned is a list:

- **idx**: a vector of colour indices in the same spatial order as the input object
- **ct**: a vector of RGB colours

Author(s)

Roger Bivand

References

http://www.gdal.org/

Examples

```r
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)
```
showWKT <- function(pTs, file = NULL, morphToESRI = TRUE)
showP4(wkt, morphFromESRI=TRUE)
showEPSG(pTs)

Arguments

pTs   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
wkt   A valid WKT character string representing a spatial reference system
morphFromESRI default TRUE, morph the WKT string from 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/osr_tutorial.html
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

```r
# S3 method for class 'SpatialGDAL'
open(con, ..., silent = FALSE, allowedDrivers = NULL, options=NULL)
# S3 method for class 'SpatialGDAL'
close(con, ...)
copy.SpatialGDAL(dataset, fname, driver = getDriver(dataset@grod),
  strict = FALSE, options = NULL, silent = FALSE)
```

Arguments

- **con**: file name of grid map for opening, SpatialGDAL object for closing
- **...**: other arguments (currently ignored)
- **silent**: logical; if TRUE, comment and non-fatal CPL driver errors suppressed
- **dataset**: object of class SpatialGDAL
- **fname**: file name of grid map
- **driver**: GDAL driver name
- **strict**: TRUE if the copy must be strictly equivalent, or more normally FALSE indicating that the copy may adapt as needed for the output format
- **allowedDrivers**: a character vector of suggested driver short names may be provided starting from GDAL 2.0
- **options**: driver-specific options to be passed to the GDAL driver; only available for opening datasets from GDAL 2.0

See Also

- `is.projected`, `CRS-class`

Examples

```r
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")
showP4(showWKT("+init=epsg:28992"))
showEPSG("+proj=utm +zone=30")
showEPSG("+proj=longlat +ellps=WGS84")
```
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 Pebesma, <edzer.pebesma@uni-muenster.de>

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[1,1])
close(x)

spTransform-methods

## Methods for Function spTransform for map projection and datum transformation in package "rgdal"

### Description

The spTransform methods provide transformation between datum(s) and conversion between projections (also known as projection and/or re-projection), 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
"SpatialPixelsDataFrame", CRSobj = CRS  Because regular grids will usually not be regular after projection/datum transformation, the input object is coerced to a SpatialPointsDataFrame, and the transformation carried out on that object. A warning: “Grid warping not available, coercing to points” is given.

"SpatialGridDataFrame", CRSobj = CRS  Because regular grids will usually not be regular after projection/datum transformation, the input object is coerced to a SpatialPointsDataFrame, and the transformation carried out on that object. A warning: “Grid warping not available, coercing to points” is given.

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.

Also note that re-projection and/or datum transformation will usually not work for regular grids.

The term used for similar operations for regular grids is warping, which involved resampling to a regular grid in the target coordinate reference system.

The methods may take an optional argument “use_ob_tran”, default FALSE, if TRUE and “+proj=ob_tran”, use General Oblique Transformation with internalised from/to projection reversal (the user oblique transforms from longlat to oblique forward rather than inverse as suggested in PROJ.4 mailing list postings); these changes are intended to meet a need pointed out by Martin Ivanov (2012-08-15). A subsequent point raised by Martin Ivanov (2017-04-28) was that use of a projected CRS with “+proj=ob_tran” led to errors, so mixing projected CRS and “+proj=ob_tran” is blocked. Transform first “+proj=ob_tran” to or from “+proj=longlat”, and then on from geographical coordinates to those desired or the reverse - see example.

If a SpatialPoints object has three dimensions, the third will also be transformed, with the metric of the third dimension assumed to be meters if the vertical units metric is not given in the projection description with +vunits= or +vto_meter= (which is 1.0 by default) http://proj4.org/parameters.html.

Note that WGS84 is both an ellipse and a datum, and that since 1984 there have been changes in the relative positions of continents, leading to a number of modifications. This is discussed for example in http://www.uvm.edu/giv/resources/WGS84_NAD83.pdf; there are then multiple transformations between NAD83 and WGS84 depending on the WGS84 definition used. One would expect that “+towgs84=” is a no-op for WGS84, but this only applies sometimes, and as there are now at least 30 years between now and 1984, things have shifted. It may be useful to note that “+nadgrids=@null” can help, see these threads: https://stat.ethz.ch/pipermail/r-sig-geo/2014-August/021611.html, http://lists.maptools.org/pipermail/proj/2014-August/006894.html, with thanks to Hermann Peifer for assistance.

Note that from PROJ.4 4.9.3, the definition of UTM is changed from TMERC to ETMERC; see example.

Author(s)

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

data(state)
states <- data.frame(state.x77, state.center)
states <- states[states$x > -120,]
coordinates(states) <- c("x", "y")
proj4string(states) <- CRS("+proj=longlat +ellps=clrk66")
summary(states)
state.1183 <- spTransform(states, CRS("+proj=longlat +ellps=GRS80"))
summary(state.1183)
state.merc <- spTransform(states, CRS="+proj=merc +ellps=GRS80")
summary(state.merc)
state.merc <- spTransform(states, CRS="+proj=merc +ellps=GRS80 +units=us-mi")
summary(state.merc)
if (projNAD()) {
states <- data.frame(state.x77, state.center)
states <- states[states$x > -120,]
coordinates(states) <- c("x", "y")
proj4string(states) <- CRS("+init=epsg:4267")
print(summary(states))
state.1183 <- spTransform(states, CRS("+init=epsg:4269"))
print(summary(state.1183))
state.kansasSlcc <- spTransform(states, CRS="+init=epsg:26978")
print(summary(state.kansasSlcc))
SPoint_NAD83 <- SpatialPoints(matrix(c(-103.869667, 44.461676, nrow=1),
proj4string=CRS("+init=epsg:4269"))
SPoint_NAD27 <- spTransform(SPoint_NAD83, CRS("+init=epsg:4267"))
print(all.equal(coordinates(SPoint_NAD83), coordinates(SPoint_NAD27)))
print(coordinates(SPoint_NAD83), digits=12)
print(coordinates(SPoint_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_primary_roads")
kiritimati_primary_roads_ll <- spTransform(kiritimati_primary_roads,
CRS("+proj=longlat +datum=WGS84"))
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"))
spTransform-methods

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)
crds <- matrix(data=c(9.05, 48.52), ncol=2)
spPoint <- SpatialPoints(coords=crds, proj4string=CRS("+proj=longlat +ellps=sphere +no_defs"))
a <- spTransform(spPoint, CRS(paste("+proj=ob_tran +o_proj=longlat",
                      "+o_lon_p=-162 +o_lat_p=39.25 +lon_0=180 +ellps=sphere +no_defs")),
                 use_ob_tran=TRUE)

a

# should be (-5.917698, -1.87195)

spTransform(a, CRS("+proj=longlat +ellps=sphere +no_defs"),
             use_ob_tran=TRUE)

try(spTransform(a, CRS(paste("+proj=tmerc +lat_0=0 +lon_0=90 +k=1",
                      "+x_0=35000000 +y_0=0 +ellps=bessel +units=m +no_defs")),
             use_ob_tran=TRUE))

spTransform(spPoint, CRS(paste("+proj=tmerc +lat_0=0 +lon_0=90 +k=1",
                      "+x_0=35000000 +y_0=0 +ellps=bessel +units=m +no_defs")))

spTransform(spTransform(a, CRS("+proj=longlat +ellps=sphere +no_defs"),
                        use_ob_tran=TRUE), CRS(paste("+proj=tmerc +lat_0=0 +lon_0=90 +k=1",
                        "+x_0=35000000 +y_0=0 +ellps=bessel +units=m +no_defs")))

crds1 <- matrix(data=c(7, 51, 8, 52, 9, 52, 10, 51, 7, 51), ncol=2, byrow=TRUE, dimnames=list(NULL, c("lon", "lat")));
crds2 <- matrix(data=c(8, 48, 9, 49, 11, 49, 9, 48, 8, 48), ncol=2, byrow=TRUE, dimnames=list(NULL, c("lon", "lat")));
crds3 <- matrix(data=c(6, 47, 6, 55, 15, 55, 15, 47, 6, 47), ncol=2, byrow=TRUE, dimnames=list(NULL, c("lon", "lat")));

spLines <- SpatialLines(list(Line(Line(crds1), Line(crds2), Line(crds3)), ID="a"));
spLines@proj4string <- CRS("+proj=longlat +ellps=sphere +no_defs")
bbox(spLines);

spLines_tr <- spTransform(spLines, CRS("+proj=ob_tran +o_proj=longlat +o_lon_p=-162 +o_lat_p=39.25 +lon_0=180 +ellps=sphere +no_defs"),
                         use_ob_tran=TRUE);
bbox(spLines_tr)

bbox(spTransform(spLines_tr, CRS("+proj=longlat +ellps=sphere"),
                 use_ob_tran=TRUE))

spPolygons <- SpatialPolygons(list(Polygons(list(Polygon(crds1),
                                               Polygon(crds2), Polygon(crds3)), ID="a")));
spPolygons@proj4string <- CRS("+proj=longlat +ellps=sphere +no_defs")
bbox(spPolygons);

spPolygons_tr <- spTransform(spPolygons, CRS("+proj=ob_tran +o_proj=longlat +o_lon_p=-162 +o_lat_p=39.25 +lon_0=180 +ellps=sphere +no_defs"),
                              use_ob_tran=TRUE);
bbox(spPolygons_tr)
bbox(spTransform(spPolygons_tr, CRS("+proj=longlat +ellps=sphere"),
    use_ob_tran=TRUE))
# added after posting by Martin Ivanov
data(nor2k)
summary(nor2k)
nor2kNGO <- spTransform(nor2k, CRS("+init=epsg:4273"))
summary(nor2kNGO)
all.equal(coordinates(nor2k)[,3], coordinates(nor2kNGO)[,3])
# added after posting by Don MacQueen
crds <- cbind(c(-121.524764291826, -121.523480804667),
    c(37.6600366036405, 37.6543604613483))
ref <- cbind(c(1703671.30566227, 1704020.2013366),
    c(424014.398845834, 421943.708664294))
crs.step1.cf <- CRS(paste("+proj=lcc +lat_1=38.43333333333333",
    "+lat_2=37.06666666666667 +lat_0=36.5 +lon_0=-120.5",
    "+x_0=2000000.0 +y_0=500000.0 +ellps=GRS80 +units=us-ft +no_defs",
    "+towgs84=-0.9911.9072,0.5129,0.025789908,0.0096501,0.0116599,0.0"))
locs.step1.cf <- spTransform(SpatialPoints(crds,
    proj4string=CRS("+proj=longlat +datum=WGS84")), crs.step1.cf)
suppressWarnings(proj4string(locs.step1.cf) <-
    CRS(paste("+proj=lcc",
    "+lat_1=38.43333333333333 +lat_2=37.06666666666667 +lat_0=36.5",
    "+lon_0=-120.5 +x_0=2000000.0 +y_0=500000.0 +ellps=GRS80 +units=us-ft",
    "+no_defs +nadgrids=@null")))
locs.step2.cfb <- spTransform(locs.step1.cf, CRS("+init=epsg:26743"))
coordinates(locs.step2.cfb) - ref
all.equal(unname(coordinates(locs.step2.cfb)), ref)
# Test for UTM == TMERC (<= 4.9.2) or UTM == ETMERC (> 4.9.2)
nhh <- SpatialPointsDataFrame(matrix(c(5.304234, 60.422311), ncol=2),
    proj4string=CRS("+init=epsg:4326"), data=data.frame(office="RSB"))
nhh_utm_SRn_pT <- spTransform(nhh, CRS("+init=epsg:3044"))
nhh_tmerc_P4 <- spTransform(nhh, CRS(paste("+proj=tmerc +k=0.9996",
    "+lon_0=9 +x_0=500000 +ellps=GRS80 +towgs84=0,0,0,0,0,0 +units=m +no_defs")))
nhh_etmerc_P4 <- spTransform(nhh, CRS(paste("+proj=etmerc +k=0.9996",
    "+lon_0=9 +x_0=500000 +ellps=GRS80 +towgs84=0,0,0,0,0,0 +units=m +no_defs")))
all.equal(coordinates(nhh_utm_SRn_pT), coordinates(nhh_tmerc_P4),
    tolerance=1e-9, scale=1)
# UTM == TMERC: PROJ4 <=4.9.2
all.equal(coordinates(nhh_utm_SRn_pT), coordinates(nhh_etmerc_P4),
    tolerance=1e-9, scale=1)
# UTM == ETMERC: PROJ4 > 4.9.2
unis <- SpatialPointsDataFrame(matrix(c(15.653453, 78.222504), ncol=2),
    proj4string=CRS("+init=epsg:4326"), data=data.frame(office="UNIS"))
unis_utm_33N_P4 <- spTransform(unis, CRS("+init=epsg:3045"))
unis_tmerc_P4 <- spTransform(unis, CRS(paste("+proj=tmerc +k=0.9996 +lon_0=15",
    "+x_0=500000 +ellps=GRS80 +towgs84=0,0,0,0,0,0 +units=m +no_defs")))
unis_etmerc_P4 <- spTransform(unis, CRS(paste("+proj=etmerc +k=0.9996",
    "+lon_0=15 +x_0=500000 +ellps=GRS80 +towgs84=0,0,0,0,0,0 +units=m +no_defs")))
all.equal(coordinates(unis_utm_33N_P4), coordinates(unis_tmerc_P4),
    tolerance=1e-9, scale=1)
# UTM == TMERC: PROJ4 <=4.9.2
all.equal(coordinates(unis_utm_33N_P4), coordinates(unis_etmerc_P4),
    tolerance=1e-9, scale=1)
# UTM == ETMERC: PROJ4 > 4.9.2
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

\[
\text{writeOGR(obj, dsn, layer, driver, dataset_options = NULL,}
\]
\[
\text{layer_options= NULL, verbose = FALSE, check_exists=NULL,}
\]
\[
\text{overwrite_layer=FALSE, delete_dsn=FALSE, morphToESRI= NULL,}
\]
\[
\text{encoding= NULL)}
\]

Arguments

- **obj**: a SpatialPointsDataFrame, SpatialLinesDataFrame, or a SpatialPolygonsDataFrame object.
- **dsn**: data source name (interpretation varies by driver — for some drivers, dsn is a file name, but may also be a folder)
- **layer**: layer name (varies by driver, may be a file name without extension)
- **driver**: a character string equal to one of the driver names returned by `ogrDrivers`
- **dataset_options**: 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
- **check_exists**: default NULL, which tests for the GDAL version, and sets FALSE if < 1.8.0, or TRUE for >= 1.8.0
- **overwrite_layer**: default FALSE, if TRUE and check_exists=TRUE, delete the existing layer of the same name from the data source before writing the new layer; this will delete data and must be used with extreme caution, its behaviour varies between drivers, and accommodates changes that may appear in GDAL 1.8
- **delete_dsn**: default FALSE, may be set to TRUE if overwrite_layer reports that the data source cannot be updated; this will delete data and must be used with extreme caution, its behaviour varies between drivers, and accommodates changes that may appear in GDAL 1.8
- **morphToESRI**: default NULL, in which case set TRUE if driver is “ESRI Shapefile” or FALSE otherwise; may be used to override this default
encoding default NULL, if set to a character string, it will be used to convert output strings from the given value to UTF-8 encoding.

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. Note that for the “ESRI Shapefile” driver and GDAL >= 1.9, the layer_options value of ‘ENCODING="LDID/CP1252”’ or other values found on [http://www.autopark.ru/ASBProgrammerGuide/DBFSTRUC.HTM](http://www.autopark.ru/ASBProgrammerGuide/DBFSTRUC.HTM) to set the encoding byte of the output DBF file (link refered to in ogr/ogrsf_frmts/shape/ogrshapelayer.cpp. The effect of setting the LDID may vary depending on whether GDAL is built with iconv or not, and on the setting of the CPL Option “SHAPE_ENCODING”.

While there is no certainty, newer drivers such as KML, GML, SQLite and Geopackage (GPKG) may encode string fields as UTF-8. Users are advised to explore this on a case to case basis using Encoding on string fields of objects to be output, converting where necessary with iconv or assigning the appropriate value with Encoding.

Value

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

Warning

The overwrite_layer and delete_dsn arguments are provided only for experienced script writers who need to be able to destroy data, for example during repetitive simulation runs. They should never be used by anyone who is not confident about deleting files.

WriteOGR Polygon bug in 1.1-1

In fixing a bug in the correct handling of SFS polygon geometries in version 1.1-1, a further bug was introduced affecting cases of wkbPolygon (not wkbMultiPolygon) output where SFS hole status in the output object was (correctly) defined in the comment to Polygons objects. The error only occurred when all the Polygons objects had one exterior ring, and zero or more interior rings. The error led to the coordinates of the rings cumulating, because the rings were not emptied before assigning the next ring. Version 1.1-2 corrects the error; thanks to JamesWorrall for a complete bug report [https://stat.ethz.ch/pipermail/r-sig-geo/2015-December/023796.html](https://stat.ethz.ch/pipermail/r-sig-geo/2015-December/023796.html).

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 and GPX drivers, note that the geometries should be in geographical coordinates with datum WGS84.
writeOGR

Author(s)

Roger Bivand

References


See Also

readOGR

Examples

cities <- readOGR(system.file("vectors", package = "rgdal")[[1]], "cities")
is.na(cities$POPULATION) <- cities$POPULATION == -99
summary(cities$POPULATION)
td <- file.path(tempdir(), "rgdal_examples"); dir.create(td)
# BDR 2016-12-15 (MapInfo driver fails writing to directory with ".")
if(nchar(Sys.getenv("OSGEO4W_ROOT")) > 0) {
  OLDPWD <- getwd()
  setwd(td)
  td <- "."
}
writeOGR(cities, td, "cities", driver="ESRI Shapefile")
try(writeOGR(cities, td, "cities", driver="ESRI Shapefile"))
writeOGR(cities, td, "cities", driver="ESRI Shapefile", overwrite_layer=TRUE)
cities2 <- readOGR(td, "cities")
summary(cities2$POPULATION)
if ("SQLite" %in% ogrDrivers()$name) {
  tf <- tempfile()
  try(writeOGR(cities, tf, "cities", driver="SQLite", layer_options="LAscanner=N0"))
}
if ("GeoJSON" %in% ogrDrivers()$name) {
  js <- '{
    "type": "MultiPolygon",
    "coordinates": [[[102.0, 2.0], [103.0, 2.0], [103.0, 3.0], [102.0, 3.0],
      [102.0, 2.0]], [[[100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0],
      [100.0, 0.0]]]}
  }'
  spdf <- readOGR(js, layer="OGRGeoJSON")
in1_comms <- sapply(slot(spdf, "polygons"), comment)
print(in1_comms)
  tf <- tempfile()
writeOGR(spdf, tf, "GeoJSON", driver="GeoJSON")
#spdf1 <- readOGR(tf, "GeoJSON")
spdf1 <- readOGR(tf)
in2_comms <- sapply(slot(spdf1, "polygons"), comment)
print(in2_comms)
print(isTRUE(all.equal(in1_comms, in2_comms)))
}
## Not run: if ("GML" %in% ogrDrivers()$name) {
  airports <- try(readOGR(system.file("vectors/airports.gml",}
package = "rgdal"[1], "airports")
if (class(airports) != "try-error") {
    writeOGR(cities, paste(td, "cities.gml", sep="/"), "cities", driver="GML")
cities3 <- readOGR(paste(td, "cities.gml", sep="/"), "cities")
}
}
## End(Not run)
# The GML driver does not support coordinate reference systems
if ("KML" %in% ogrDrivers()$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)
if (strsplit(getGDALVersionInfo(), " ")[1][2] < "2") {
    # For GDAL >= 2, the TAB driver may need a BOUNDS layer option
    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)
if (strsplit(getGDALVersionInfo(), " ")[1][2] < "2") {
    # For GDAL >= 2, the TAB driver may need a BOUNDS layer option
    writeOGR(scot_BNG, td, "scot_BNG", driver="MapInfo File")
    list.files(td)
    scot_BNG2 <- readOGR(paste(td, "scot_BNG.tab", sep="/"), "scot_BNG",
        addCommentsToPolygons=FALSE)
    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("OSGEODWD_ROOT")) > 0) {
    setwd(OSGDWD)
}
Index

*Topic classes
  CRS-class, 3
  GDALDataset-class, 7
  GDALDriver-class, 8
  GDALMajorObject-class, 10
  GDALRasterBand-class, 11
  GDALReadOnlyDataset-class, 13
  GDALReadOnlyDataset-methods, 15
  GDALTransientDataset-class, 17
  SpatialGDAL-class, 40

*Topic datasets
  GridsDatums, 18
  nor2k, 21

*Topic methods
  closeDataset-methods, 2
  spTransform-methods, 42

*Topic spatial
  CRS-class, 3
  displayDataset, 5
  GDALcall, 6
  llgridlines, 19
  make_EPSG, 20
  project, 22
  projInfo, 24
  readGDAL, 25
  readOGR, 31
  RGB2PCT, 36
  SGDF2PCT, 37
  showWKT, 39
  spTransform-methods, 42
  writeOGR, 47
  GDALReadOnlyDataset-method
    (GDALReadOnlyDataset-methods), 15
  [, GDALReadOnlyDataset-method
    (GDALReadOnlyDataset-methods), 15
  [, SpatialGDAL-method
    (SpatialGDAL-class), 40
  ]-, SpatialGDALWrite-method
    (SpatialGDAL-class), 40
  [[, SpatialGDAL, ANY, missing-method
    (SpatialGDAL-class), 40
  ][-, SpatialGDAL, ANY, missing-method
    (SpatialGDAL-class), 40
  $, SpatialGDAL-method
    (SpatialGDAL-class), 40
  $<-, SpatialGDAL-method
    (SpatialGDAL-class), 40
  asciiGrid, 28
  asSGDF_GROD (readGDAL), 25
  checkCRSArgs (CRS-class), 3
  close.SpatialGDAL (SpatialGDAL-class),
    40
  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-method
    (GDALReadOnlyDataset-methods), 15
  coerce, SpatialGDAL, SpatialGridDataFrame-method
    (SpatialGDAL-class), 40
  coerce, SpatialGDAL, SpatialPixelsDataFrame-method
    (SpatialGDAL-class), 40
  copy.SpatialGDAL (SpatialGDAL-class), 40
  copyDataset (GDALDataset-class), 7
  create2GDAL (readGDAL), 25
  CRS (CRS-class), 3
  CRS-class, 3
  CRSargs (CRS-class), 3
deleteDataset (GDALDataset-class), 7
    dim, GDALRasterBand-method (GDALRasterBand-class), 11
    dim, GDALReadOnlyDataset-method (GDALReadOnlyDataset-class), 13
displayDataset, 5
Encoding, 34, 48
flipVertical, 28
GDAL.close (GDALReadOnlyDataset-class), 13
GDAL.open (GDALReadOnlyDataset-class), 13
GDALcall, 6
GDALDataset-class, 7
GDALDriver-class, 8
gdalDrivers (GDALDriver-class), 8
GDALInfo (readGDAL), 25
GDALMajorObject-class, 10
GDALRasterBand-class, 11
GDALReadOnlyDataset-class, 13
GDALReadOnlyDataset-meth, 15
GDALSpatialRef (readGDAL), 25
GDALTransientDataset-class, 17
get_OVERRIDE_PROJ_DATUM_WITH_TOWGS84 (GDALRasterBand-class), 11
ggetColorTable
    (GDALReadOnlyDataset-class), 13
gCPLConfigOption (GDALDriver-class), 8
gGetDescription (GDALMajorObject-class), 10
gGetDriver (GDALReadOnlyDataset-class), 13
gGetDriverLongName (GDALDriver-class), 8
gGetDriverName (GDALDriver-class), 8
gGetGDAL_DATA_Path (GDALDriver-class), 8
gGetGDALCheckVersion (GDALDriver-class), 8
gGetGDALDriverNames (GDALDriver-class), 8
gGetGDALVersionInfo (GDALDriver-class), 8
gGetGeoTransFunc (GDALReadOnlyDataset-class), 13
gGetPROJ4libPath (projInfo), 24
gGetPROJ4VersionInfo (projInfo), 24
gGetProjectionRef, 26
gGetProjectionRef (GDALRasterBand-class), 11
getRasterBand (GDALRasterBand-class), 11
gGetRasterBlockSize (GDALRasterBand-class), 11
gGetRasterData, 16
gGetRasterData (GDALRasterBand-class), 11
gGetRasterTable (GDALRasterBand-class), 11
gGridat, 19
gridlines, 19
GridsDatums, 4, 18
GridTopology-class, 41
iconv, 35, 48
image, 28
initialize, GDALDataset-method (GDALDataset-class), 7
initialize, GDALDriver-method (GDALDriver-class), 8
initialize, GDALRasterBand-method (GDALRasterBand-class), 11
initialize, GDALReadOnlyDataset-method (GDALReadOnlyDataset-class), 13
initialize, GDALTransientDataset-method (GDALTransientDataset-class), 17
is.projected, 19, 40
llgridlines, 19
make_EPSG, 19
nork2, 21
normalizePath, 7
ogrDrivers, 47
ogrDrivers (readOGR), 31
ogrFIDs (readOGR), 31
ogrInfo (readOGR), 31
ogrListLayers (readOGR), 31
OGSSpatialRef (readOGR), 31
open. SpatialGDAL (SpatialGDAL-class), 40
options, 32
print.CRS (CRS-class), 3
print.GDALobj (readGDAL), 25
print.ogrinfo (readOGR), 31
print.summary.SpatialGDAL (SpatialGDAL-class), 40
project, 22
projInfo, 24
projnad (projinfo), 24
putRasterData (GDALDataset-class), 7
rawTransform (GDALcall), 6
readGDAL, 16, 25
readOGR, 31, 49
readShapePoly, 35
RGB2PCT, 36
RGDAL_checkCRSArgs (CRS-class), 3
saveDataset, 17
saveDatasetAs (GDALDataset-class), 7
saveDatasetAs (GDALDataset-class), 7
set_OVERRIDe_PROJ_DATUM_WITH_TOwGS84
(GDALRasterBand-class), 11
setCPLConfigOption (GDALDriver-class), 8
SGDF2PCT, 37
show, CRS-method (CRS-class), 3
showEPSG, 4
showEPSG (showWKT), 39
showP4 (showWKT), 39
showWKT, 39
Spatial, 19
Spatial-class, 41
SpatialGDAL-class, 40
SpatialGDALWrite-class
(SpatialGDAL-class), 40
SpatialGridDataFrame-class, 27, 28, 41
SpatialPixels-class, 41
SpatialPixelsDataFrame-class, 27
SpatialPoints, 41
SpatialPointsDataFrame-class, 28
spTransform (spTransform-methods), 42
spTransform, SpatialGridDataFrame, CRS-method
(spTransform-methods), 42
spTransform, SpatialLines, CRS-method
(spTransform-methods), 42
spTransform, SpatialLinesDataFrame, CRS-method
(spTransform-methods), 42
spTransform, SpatialPolygonsDataFrame, CRS-method
(spTransform-methods), 42
spTransform, SpatialPolygons, CRS-method
(spTransform-methods), 42
spTransform, SpatialPointsDataFrame, CRS-method
(spTransform-methods), 42
spTransform, SpatialPoints, CRS-method
(spTransform-methods), 42
sub.GDROD
(GDALReadOnlyDataset-methods), 15
summary, SpatialGDAL-method
(SpatialGDAL-class), 40
toSigned (GDALRasterBand-class), 11
toUnSigned (GDALRasterBand-class), 11
type.convert, 33
vec2RGB (SGDF2PCT), 37
writeGDAL (readGDAL), 25
writeOGR, 47