Package: rgdal (via r-universe)

January 2, 2025

Title Bindings for the 'Geospatial' Data Abstraction Library

Version 1.6-7

Date 2023-05-31

Depends R (>= 3.5.0), methods, sp (>= 1.1-0)

Imports grDevices, graphics, stats, utils

LinkingTo sp

Suggests knitr, DBI, RSQLite, maptools, mapview, rmarkdown, curl, rgeos

NeedsCompilation yes

Description Provides bindings to the 'Geospatial' Data Abstraction Library ('GDAL') (>= 1.11.4) and access to projection/transformation operations from the 'PROJ' library. Please note that 'rgdal' will be retired during October 2023, plan transition to sf/stars/'terra' functions using 'GDAL' and 'PROJ' at your earliest convenience (see <https://r-spatial.org/r/2023/05/15/evolution4.html> and earlier blogs for guidance). Use is made of classes defined in the 'sp' package. Raster and vector map data can be imported into R, and raster and vector 'sp' objects exported. The 'GDAL' and 'PROJ' libraries are external to the package, and, when installing the package from source, must be correctly installed first; it is important that 'GDAL' < 3 be matched with 'PROJ' < 6. From 'rgdal' 1.5-8, installed with to 'GDAL' >=3, 'PROJ' >=6 and 'sp' >= 1.4, coordinate reference systems use 'WKT2_2019' strings, not 'PROJ' strings. 'Windows' and 'macOS' binaries (including 'GDAL', 'PROJ' and their dependencies) are provided on 'CRAN'.

License GPL (>= 2)

URL http://rgdal.r-forge.r-project.org, https://gdal.org, https://proj.org, https://r-forge.r-project.org/projects/rgdal/

SystemRequirements PROJ (>= 4.8.0, https://proj.org/download.html) and GDAL (>= 1.11.4, https://gdal.org/download.html), with versions

either (A) PROJ < 6 and GDAL < 3 or (B) PROJ >= 6 and GDAL >= 3. For degraded PROJ 6 or 7 and GDAL < 3, use the configure argument '--with-proj_api=``proj_api.h'".

VignetteBuilder knitr

Author Roger Bivand [cre, aut] (<https://orcid.org/0000-0003-2392-6140>), Tim Keitt [aut], Barry Rowlingson [aut, ctb], Edzer Pebesma [ctb], Michael Sumner [ctb], Robert Hijmans [ctb], Daniel Baston [ctb], Even Rouault [cph, ctb], Frank Warmerdam [cph, ctb], Jeroen Ooms [ctb], Colin Rundel [ctb]

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

Date/Publication 2023-05-31 22:30:02 UTC

Config/pak/sysreqs libgdal-dev gdal-bin libproj-dev

Repository https://ar-puuk.r-universe.dev

RemoteUrl https://github.com/cran/rgdal

RemoteRef HEAD

RemoteSha 5098f49a1c55782b6f52cda94f7a4750b7258ca0

Contents

closeDataset-methods *closeDataset methods*

Description

Methods for closing GDAL datasets, used internally

Usage

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

Arguments

dataset	GDAL dataset

Methods

dataset = "ANY" default method, returns error

dataset = "GDALReadOnlyDataset" closes the "GDALReadOnlyDataset"

dataset = "GDALTransientDataset" closes the "GDALTransientDataset"

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

Description

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

Usage

```
checkCRSArgs_ng(uprojargs=NA_character_, SRS_string=NULL,
  get_source_if_boundcrs=TRUE)
compare_CRS(CRS1, CRS2)
```

Arguments

uprojargs	character string PROJ.4 projection arguments
SRS_string	default NULL, experimental in connection with adaptation to GDAL>=3/PROJ>=6; a valid WKT string or SRS definition such as "OGC:CRS84"

`get_source_if_boundcrs`
: The presence of the `+towgs84=` key in a Proj4 string `projargs=` argument value may promote the output WKT2 CRS to BOUNDCRS for PROJ >= 6 and GDAL >= 3, which is a coordinate operation from the input datum to WGS84. This is often unfortunate, so a PROJ function is called through **rgdal** to retrieve the underlying source definition.

`CRS1, CRS2`
: objects of class `"CRS"`

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

https://proj.org/

Examples

```
library(sp)
data(meuse)
coordinates(meuse) <- c("x", "y")
proj4string(meuse) <- CRS("+init=epsg:28992")
CRSargs(CRS(proj4string(meuse)))
run <- new_proj_and_gdal()
if (run) {
c1 <- CRS(SRS_string="OGC:CRS84")
c2 <- CRS("+proj=longlat")
compare_CRS(c1, c2)
}
if (run) {
comment(c2) <- NULL
compare_CRS(c1, c2)
}
```

---

displayDataset *Display a GDAL dataset*

Description

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

Usage

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

https://gdal.org/

Examples

```
## Not run:
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, reset.par=FALSE)
GDAL.close(x)
GDAL.close(dx)

## End(Not run)
```

GDALDataset-class *Class "GDALDataset"*

Description

GDALDataset extends GDALReadOnlyDataset-class with data update commands.

Usage

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

Arguments

dataset	An object inheriting from class 'GDALDataset'
rasterData	A data array with 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 hande 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, begining 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()
getGDALwithGEOS()
rgdal_extSoftVersion()
get_cached_orig_PROJ_LIB()
get_cached_orig_GDAL_DATA()
get_cached_set_PROJ_LIB()
get_cached_set_GDAL_DATA()
```

Arguments

driver	An object inheriting from class 'GDALDriver'
str	A string, may be one of "--version", "VERSION_NUM", "RELEASE_DATE", "RELEASE_NAME"

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

`getGDALwithGEOS`: because drivers may behave differently if GDAL itself was built with GEOS support, the function uses a heuristic to check whether GDAL has access to the GEOS Union function or not

`get_cached_orig_PROJ_LIB`, `get_cached_orig_GDAL_DATA` The values of environment variables PROJ_LIB and GDAL_DATA as read when this package was loaded

`get_cached_set_PROJ_LIB`, `get_cached_set_GDAL_DATA` If not `""`, the values set when loading this package to point to metadata files included in CRAN binary packages

Objects from the Class

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

Slots

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

Extends

Class `"GDALMajorObject"`, directly.

Methods

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

See Also

GDALMajorObject-class

Examples

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

GDALMajorObject-class *Class "GDALMajorObject"*

Description

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

Usage

```
getDescription(object)
```

Arguments

object	an object inheriting from "GDALMajorObject"

Details

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

Objects from the Class

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

Slots

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

Methods

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

Author(s)

Timothy H. Keitt, modified by Roger Bivand

References

https://gdal.org/

See Also

GDALDriver-class, GDALReadOnlyDataset-class, GDALDataset-class and GDALTransientDataset-class

Examples

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

GDALRasterBand-class *Class "GDALRasterBand"*

Description

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

Usage

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

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

getProjectionRef(dataset, OVERRIDE_PROJ_DATUM_WITH_TOWGS84 = NULL,
 enforce_xy = NULL, get_source_if_boundcrs=TRUE)

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)
```

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_TOW is used. Ignored if the GDAL version is less than "1.8.0" or if the CPLConfigOption variable is already set
enforce_xy	(PROJ6+/GDAL3+) either use global setting (default NULL) or override policy for coordinate ordering easting/x as first axis, northing/y as second axis.
get_source_if_boundcrs	The presence of the +towgs84= key in a Proj4 string projargs= argument value may promote the output WKT2 CRS to BOUNDCRS for PROJ >= 6 and GDAL >= 3, which is a coordinate operation from the input datum to WGS84. This is often unfortunate, so a PROJ function is called through **rgdal** to retrieve the underlying source definition.
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

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 https://trac.osgeo.org/gdal/ticket/4880 and https://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

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

GDALReadOnlyDataset-class

Class "GDALReadOnlyDataset"

Description

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

Usage

```
GDAL.close(dataset)
GDAL.open(filename, read.only = TRUE, silent=FALSE,
                  allowedDrivers = NULL, options=NULL)
getDriver(dataset)
getColorTable(dataset, band = 1)
getGeoTransFunc(dataset)
```

Arguments

dataset	An object inheriting from class 'GDALReadOnlyDataset'
filename	name of file to contain raster data object; will be normalized with normalizePath if it is a file
band	The band number (1-based) to read from
read.only	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

https://gdal.org/

See Also

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

Examples

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

GDALReadOnlyDataset-methods
subset methods for "GDALReadOnlyDataset"

Description

subsets GDAL objects, returning a SpatialGridDataFrame object

Details

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

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

as.is see getRasterData

interleave see getRasterData

output.dim see getRasterData

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

An GDALReadOnlyDataset object can be coerced directly to a SpatialGridDataFrame

Methods

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

Author(s)

Edzer Pebesma

See Also

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

Examples

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

```
GDAL.close(x)

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

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

GDALTransientDataset-class *Class "GDALTransientDataset"*

Description

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

Objects from the Class

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

driver A "GDALDriver" object that determines the storage format

rows Number of rows in the newly created dataset

cols Number of columns in the newly created dataset

bands Number of bands to create

type A GDAL type name as listed in .GDALDataTypes

options Driver specific options

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

handle Used internally; not for public consumption

Slots

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

Extends

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

Methods

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

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

Author(s)

Timothy H. Keitt, modified by Roger Bivand

See Also

See also GDALDriver-class, GDALReadOnlyDataset-class

Examples

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

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 241 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 March 2016; subsequent columns are updated reprints of previous ones. 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 https://www.asprs.org/asprs-publications/grids-and-datums.

Source

https://www.asprs.org/asprs-publications/grids-and-datums

Examples

```
data(GridsDatums)
GridsDatums[grep("Norway", GridsDatums$country),]
GridsDatums[grep("Google", GridsDatums$country),]
GridsDatums[grep("^Mars$", GridsDatums$country),]
```

---

is_proj_CDN_enabled *PROJ search paths and content download network handling*

Description

From PROJ 7 (and partly 7.1), it is becoming possible to use transformation grids downloaded on demand to improve coordinate operation accuracy from a content download network (CDN). These functions report on and control the use of the CDN.

Usage

```
is_proj_CDN_enabled()
enable_proj_CDN()
disable_proj_CDN()
proj_CDN_user_writable_dir()
get_proj_search_paths()
set_proj_search_paths(paths)
```

Arguments

paths	a character vector of existing directories

Details

The PROJ user-writable CDN directory is set as soon as the internal search path is queried, and for most uses, the default value will allow all programs using PROJ such as R packages, QGIS, GRASS, etc., to access any downloaded grids. Grids are checked for staleness at regular intervals. This directory may be set to a non-default value with the PROJ_USER_WRITABLE_DIRECTORY environment variable before **rgdal** (and any other package using PROJ) is loaded and attached, from PROJ >= 7.1.0.

Value

Logical values and/or character vector search paths, often NULL for earlier versions of PROJ.

Author(s)

Roger Bivand

References

https://cdn.proj.org/.

Examples

```
is_proj_CDN_enabled()
proj_CDN_user_writable_dir()
get_proj_search_paths()
```

---

list_coordOps *List PROJ 6 coordinate operations*

Description

List PROJ 6 coordinate operations for a pair of source/target coordinate reference systems

Usage

```
list_coordOps(src_crs, tgt_crs, area_of_interest = as.numeric(NA),
 strict_containment = FALSE, visualization_order = NULL)
best_instantiable_coordOp(x)
## S3 method for class 'coordOps'
print(x, ...)
```

Arguments

src_crs	Source coordinate reference system string
tgt_crs	Target coordinate reference system string

area_of_interest
: Numeric vector; either NA, or the xmin, ymin, xmax, ymax of the bounding box of the area of interest. This may be used to restrict the search for coordinate operations

strict_containment
: default FALSE, permit partial matching of the area of interest; if TRUE strictly contain the area of interest. The area of interest is either as given, or as implied by the source/target coordinate reference systems (FIXME)

visualization_order
: default NULL, taking the value of get_enforce_xy(); if TRUE always choose x or longitude for the first axis; if FALSE, follow the axis orders given by the coordinate reference systems when constructing the coordinate operation

x
: an object of class "coordOps"

...
: arguments possibly passed through, unused

Details

(FIXME)

Value

A data frame with rows showing the coordinate operations found, and columns:

description	String describing the operation
definition	PROJ pipeline for executing the operation
accuracy	Accuracy in meters, if negative, unknown
instantiable	Can this operation be carried out with available resources
ballpark	Does this operation only have ballpark accuracy
number_grids	The number of grids required for the operation

The object has a "grids" attribute containing a nested list of grids for each coordinate operations found; if number_grids == 0, NULL, otherwise a list of grids. For each grid required, the short and long names of the grid are given, the package name if available in a PROJ grid package, and the download URL for that package. Three logical variables report whether the grid may be downloaded directly, whether it has an open license, and whether it is available.

Note

Fragile: work in progress

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References

https://proj.org/

Examples

```
run <- new_proj_and_gdal()
if (run) {
  discarded_datum <- showSRID("EPSG:27700", "PROJ")
  (x <- list_coordOps(paste0(discarded_datum, " +type=crs"), "OGC:CRS84"))
}
if (run) {
  best_instantiable_coordOp(x)
}
if (run) {
  restored_datum <- showSRID("EPSG:27700", "PROJ")
  list_coordOps(paste0(restored_datum, " +datum=OSGB36 +type=crs"), "OGC:CRS84")
}
if (run) {
  wkt_datum <- showSRID("EPSG:27700", "WKT2")
  (x <- list_coordOps(wkt_datum, "OGC:CRS84"))
}
if (run) {
  best_instantiable_coordOp(x)
}
if (run) {
  list_coordOps("EPSG:27700", "OGC:CRS84")
}
if (run) {
}
if (run) {
  discarded_datum <- showSRID("EPSG:22525", "PROJ")
  list_coordOps(paste0(discarded_datum, " +type=crs"), "EPSG:31985")
}
if (run) {
}
if (run) {
  wkt_datum <- showSRID("EPSG:22525", "WKT2")
  list_coordOps(wkt_datum, "EPSG:31985")
}
if (run) {
  (x <- list_coordOps("EPSG:22525", "EPSG:31985"))
}
if (run) {
  best_instantiable_coordOp(x)
}
```

`llgridlines` *Plot long-lat grid over projected data*

Description

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-class having projection specified
easts	numeric; see gridlines
norths	numeric; see gridlines
ndiscr	numeric; see gridlines
offset	numeric; see gridlines
side	character, default "WS"; see gridlines; 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 lables are plotted

See Also

is.projected, CRS-class

Examples

```
set_thin_PROJ6_warnings(TRUE)
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)
```

make_EPSG *Make a data frame of EPSG projection codes*

Description

Make a data frame of the European Petroleum Survey Group (EPSG) geodetic parameter dataset as distributed with PROJ.4 software (prior to PROJ 6.0.0, March 2019, only the CSV file, from March 2019 with PROJ >= 6 from the SQLite database). Because finding the correct projection specification is not easy, lists still known as EPSG lists are maintained, and more generally retrieved from databases. The data collated here are as distributed with PROJ.4.

Usage

```
make_EPSG(file)
EPSG_version()
```

Arguments

file	file name of the file matching EPSG codes and PROJ.4 arguments, should usually be autodetected; not used for PROJ >= 6

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
prj_method	extra character column from PROJ 6 showing the projection method

...

Note

See also Clifford J. Mugnier's Grids & Datums columns in Photogrammetric Engineering & Remote Sensing, https://www.asprs.org/a/resources/grids/, see also GridsDatums.

Author(s)

Roger Bivand

References

(unlinked because of certificate issues: https://epsg.org/home.html.

Examples

```
EPSG <- try(make_EPSG())
# from PROJ 6.0.0, EPSG data is no longer stored in a flat file
if (!inherits(EPSG, "try-error")) attr(EPSG, "metadata")
# PROJ.4 version 5 and later include the EPSG version as an attribute
if (!inherits(EPSG, "try-error")) EPSG[grep("Oslo", EPSG$note), 1:2]
if (!inherits(EPSG, "try-error")) EPSG[1925:1927, 3]
if (!inherits(EPSG, "try-error")) EPSG[grep("Poland", EPSG$note), 1:2]
if (!inherits(EPSG, "try-error")) EPSG[grep("Amersfoort", EPSG$note), 1:2]
if (!inherits(EPSG, "try-error")) EPSG[grep("North Carolina", EPSG$note), 1:2]
if (!inherits(EPSG, "try-error")) EPSG[2202, 3]
```

---

nor2k *Norwegian peaks over 2000m*

Description

Norwegian peaks over 2000m, 3D SpatialPoints data.

Usage

```
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] "Galdh?piggen" "Glittertinden" "Skagast?lstinden, Store (Storen)" "Styggedalstinden, Store, ?st-toppen" ... .$ 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

http://www.nfo2000m.no/; http://www.nfo2000m.no/Excel/2000m_data.xls

Examples

```
data(nor2k)
summary(nor2k)
## maybe str(nor2k) ; plot(nor2k) ...
```

---

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

```
projInfo(type = "proj")
getPROJ4VersionInfo()
getPROJ4libPath()
projNAD()
GDAL_OSR_PROJ()
GDALis3ormore()
PROJis6ormore()
new_proj_and_gdal()
```

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. From PROJ 6, "datum" is not available. From PROJ 7.1.0, "units" returns the conversion factor as numeric, not character.

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 may change the PROJ_LIB environmental variable to the PROJ.4 support files if bundled with binary packages.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References

https://proj.org/

Examples

```
getPROJ4VersionInfo()
projInfo()
projInfo("ellps")
projInfo("units")
```

RGB2PCT *Convert RGB three band to single band colour table*

Description

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

Usage

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

Arguments

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 GDALTransientDataset or a list of a GDALTransientDataset and a colour table.

Author(s)

Tim Keitt

References

https://gdal.org/

Examples

```
## Not run:
logo <- system.file("pictures/Rlogo.jpg", package="rgdal")[1]
x <- GDAL.open(logo)
dim(x)
dx <- RGB2PCT(x, band=1:3)
displayDataset(dx, reset.par=FALSE)
dim(dx)
GDAL.close(x)
GDAL.close(dx)

## End(Not run)
```

---

rgdal-deprecated *Deprecated functions and methods*

Description

Cumulative deprecated functions and methods from rgdal prior to package retirement/archiving during 2023.

Usage

```
project(xy, proj, inv = FALSE, use_ob_tran=FALSE, legacy=TRUE,
 allowNAs_if_not_legacy=FALSE, coordOp = NULL, verbose = FALSE,
 use_aoi=TRUE)
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, enforce_xy = 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, enforce_xy = NULL)
writeGDAL(dataset, fname, drivername = "GTiff", type = "Float32",
 mvFlag = NA, options=NULL, copy_drivername = "GTiff", setStatistics=FALSE,
 colorTables = NULL, catNames=NULL, enforce_xy = NULL)
create2GDAL(dataset, drivername = "GTiff", type = "Float32", mvFlag = NA,
 options=NULL, fname = NULL, setStatistics=FALSE, colorTables = NULL,
 catNames=NULL, enforce_xy = NULL)
GDALinfo(fname, silent=FALSE, returnRAT=FALSE, returnCategoryNames=FALSE,
 returnStats=TRUE, returnColorTable=FALSE,
 OVERRIDE_PROJ_DATUM_WITH_TOWGS84=NULL, returnScaleOffset=TRUE,
 allowedDrivers = NULL, enforce_xy = NULL, options=NULL)
GDALSpatialRef(fname, silent=FALSE, OVERRIDE_PROJ_DATUM_WITH_TOWGS84=NULL,
 allowedDrivers = NULL, enforce_xy = NULL, get_source_if_boundcrs=TRUE, options=NULL)
readOGR(dsn, layer, verbose = TRUE, p4s=NULL,
 stringsAsFactors=as.logical(NA),
 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, morphFromESRI = NULL,
 dumpSRS = FALSE, enforce_xy = NULL, D3_if_2D3D_points=FALSE, missing_3D=0)
ogrInfo(dsn, layer, encoding=NULL,
 use_iconv=FALSE, swapAxisOrder=FALSE, require_geomType = NULL,
 morphFromESRI = NULL, dumpSRS = FALSE, enforce_xy = NULL,
 D3_if_2D3D_points=FALSE)
ogrFIDs(dsn, layer)
ogrDrivers()
OGRSpatialRef(dsn, layer, morphFromESRI=NULL, dumpSRS = FALSE, driver = NULL,
 enforce_xy = NULL, get_source_if_boundcrs=TRUE)
ogrListLayers(dsn)
## S3 method for class 'ogrinfo'
print(x, ...)
writeOGR(obj, dsn, layer, driver, dataset_options = NULL,
 layer_options=NULL, verbose = FALSE, check_exists=NULL,
 overwrite_layer=FALSE, delete_dsn=FALSE, morphToESRI=NULL,
 encoding=NULL, shp_edge_case_fix=FALSE, dumpSRS = FALSE)
checkCRSArgs(uprojargs)
showWKT(p4s, file = NULL, morphToESRI = FALSE, enforce_xy = NULL)
showEPSG(p4s, enforce_xy = NULL)
getCPLConfigOption(ConfigOption)
setCPLConfigOption(ConfigOption, value)
GDALcall(object, option, ...)
rawTransform(projfrom, projto, n, x, y, z=NULL, wkt=FALSE)
```

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)
allowNAs_if_not_legacy	used if legacy is FALSE, default FALSE; introduced to handle use of NAs as object separators in **oce**
coordOp	default NULL, for PROJ >= 6 used to pass through a pre-defined coordinate operation
verbose	default FALSE, for PROJ >=6 used to show the coordinate operation used

use_aoi — With PROJ >= 6, use the area of interest defined as the range of xy in limiting the search for candidate coordinate operations; set FALSE if use_ob_tran is TRUE

fname — file name of grid map; in create2GDAL provides a way to pass through a file name with driver-required extension for sensitive drivers

x — A 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

output.dim — 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

band — if missing, all bands are read

p4s — PROJ4 string defining CRS, if default (NULL), the value is read from the GDAL data set

half.cell — 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

silent — logical; if TRUE, comment and non-fatal CPL driver errors suppressed

OVERRIDE_PROJ_DATUM_WITH_TOWGS84 — logical value, default NULL, which case the cached option set by set_OVERRIDE_PROJ_DATUM_WITH_TOW(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

allowedDrivers — a character vector of suggested driver short names may be provided starting from GDAL 2.0

... — 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 copyDatset 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 tests

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.

enforce_xy — (PROJ6+/GDAL3+) either use global setting (default NULL) or override policy for coordinate ordering easting/x as first axis, northing/y as second axis.

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 avaliable (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

`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.

`stringsAsFactors`
logical: should character vectors be converted to factors? Default NA, which uses the deprecated `default.stringsAsFactors()` in R < 4.1.0 (see `link[base]{data.frame}`). Before R 4, strings were converted to factors by default, as argument value TRUE. See https://developer.r-project.org/Blog/public/2020/02/16/stringsasfactors/index.html for details of changes.

`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 alway discarded for line and polygon geometries

`dropNULLGeometries`
default TRUE, drop both declared NULL geometries, and empty geometries with no coordinates; if FALSE, return a data frame with the attribute values of the NULL and empty geometries. From 1.3-6, setting FALSE also works when there are no geometries at all, returning a data.frame including all FIDs

`useC` default TRUE, if FALSE use original interpreted code in a loop

`disambiguateFIDs`
: default FALSE, if TRUE, and FID values are not unique, they will be set to unique values 1:N for N features; problem observed in GML files

`addCommentsToPolygons`
: default TRUE, may be set FALSE for legacy behaviour; used to indicate which interior rings are holes in which exterior rings in conformance with OGC SFS specifications

`encoding`
: default NULL, if set to a character string, and the driver is "ESRI Shapefile", and use_iconv is FALSE, it is passed to the CPL Option "SHAPE_ENCODING" immediately before reading the DBF of a shapefile. If use_iconv is TRUE, and encoding is not NULL, it will be used to convert input strings from the given value to the native encoding for the system/platform.

`use_iconv`
: default FALSE; if TRUE and encoding is not NULL, it will be used to convert input strings from the given value to the native encoding for the system/platform.

`swapAxisOrder`
: default FALSE, if TRUE, treat y coordinate as Easting, x as Northing, that is the opposite to the assumed order; this may be needed if some OGR read drivers do not behave as expected

`require_geomType`
: , default NULL, if one of: `c("wkbPoint", "wkbLineString", "wkbPolygon")`, then in input with multiple geometry types, the chosen type will be read

`integer64`
: default "no.loss" (from rgdal 1.2.*). From GDAL 2, fields to be read may also take Integer64 values. As R has no such storage mode, three options are offered, analogous with `type.convert` for numeric conversion: "allow.loss" which clamps to 32-bit signed integer (default < rgdal 1.2), "warn.loss" - as "allow.loss" but warns when clamping occurs, and "no.loss", which reads as a character string using the formatting applied by default by GDAL (default >= rgdal 1.2). The use of 64-bit integers is usually a misunderstanding, as such data is almost always a long key ID.

`GDAL1_integer64_policy`
: default FALSE, if TRUE, Integer64 fields are read as doubles

`morphFromESRI`
: default NULL, morph from ESRI WKT1 dialect

`dumpSRS`
: dump SRS to stdout from inside GDAL to debug conversion - developer use only

`get_source_if_boundcrs`
: The presence of the `+towgs84=` key in a Proj4 string `projargs=` argument value may promote the output WKT2 CRS to BOUNDCRS for PROJ >= 6 and GDAL >= 3, which is a coordinate operation from the input datum to WGS84. This is often unfortunate, so a PROJ function is called through **rgdal** to retrieve the underlying source definition.

`D3_if_2D3D_points`
: https://github.com/r-spatial/sf/issues/1683 case of mixed 2D/3D track points - set TRUE to 3D to pass

`missing_3D`
: default 0, may be finite real numbers; https://github.com/r-spatial/sf/issues/1683

`driver`
: default NULL, driver found using `ogrListLayers` from the data source; otherwise already known and passed through from a calling function

obj	a SpatialPointsDataFrame, SpatialLinesDataFrame, or a SpatialPolygonsDataFrame object.
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
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
shp_edge_case_fix	default FALSE, if TRUE, attempt to work around MULTIPOLYGON to POLYGON degradation in ESRI Shapefile output with two touching exterior rings in a single feature (not yet implemented).
uprojargs	character string PROJ.4 projection arguments
file	if not NULL, a file name to which the output Well-Known Text representation should be written
ConfigOption	CPL configure option documented in https://trac.osgeo.org/gdal/wiki/ConfigOptions and elsewhere in GDAL source code
value	a string value to set a CPL option; NULL is used to unset the CPL option
object	GDALTransientDataset (option = 'SetGeoTransform', 'SetProject') or GDALRasterBand (the other options)
option	character. One of 'SetGeoTransform', 'SetProject', 'SetNoDataValue', 'SetStatistics', 'SetRasterColorTable' or 'SetCategoryNames')
projfrom	character. PROJ.4 coordinate reference system (CRS) description
projto	character. PROJ.4 CRS description
n	number of coordinates
y	y coordinates
z	z coordinates
wkt	default FALSE, if TRUE, the caller determines that projfrom and projto are wkt and that new_proj_and_gdal() returns TRUE to avoid multiple warnings when the function is called repetitively

Examples

```
data(state)
res <- project(cbind(state.center$x, state.center$y),
 "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84")
res1 <- project(res, "+proj=lcc +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)
broke_proj <- FALSE
pv <- .Call("PROJ4VersionInfo", PACKAGE="rgdal")[[2]]
# https://github.com/OSGeo/PROJ/issues/1525
if (pv >= 600 && pv < 620) broke_proj <- TRUE
if (!broke_proj) {
crds <- matrix(data=c(9.05, 48.52), ncol=2)
a <- project(crds, 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 == TMERC (<= 4.9.2) or UTM == ETMERC (> 4.9.2)
nhh <- matrix(c(5.304234, 60.422311), ncol=2)
nhh_utm_32N_P4 <- project(nhh, "+init=epsg:3044")
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, nhh_tmerc_P4, tolerance=1e-9, scale=1)
# UTM == TMERC: PROJ4 <=4.9.2
all.equal(nhh_utm_32N_P4, nhh_etmerc_P4, tolerance=1e-9, scale=1)
# UTM == ETMERC: PROJ4 > 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 == TMERC: PROJ4 <=4.9.2
all.equal(unis_utm_33N_P4, unis_etmerc_P4, tolerance=1e-9, scale=1)
# UTM == ETMERC: PROJ4 > 4.9.2
#pv <- attr(getPROJ4VersionInfo(), "short")
#if (pv < 500) {
```

```
# valgrind leakages in some cases for PROJ >= 5; many non-projection proj values added
# available projections and their inverses if provided
# For >=4.9.3 returns non-finite points rather than needing crash protection
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, use_aoi=FALSE), silent=TRUE)
  if (inherits(xy, "try-error")) {
    res[i] <- NA
    msgs[i] <- paste("fwd:", strsplit(xy, "\n")[[1]][2])
  } else if(any(abs(xy) > 1e+08)) {
    res[i] <- NA
    msgs[i] <- paste("fwd: huge value")
  } else {
    out <- try(project(xy, iprs, inv=TRUE, legacy=TRUE, use_aoi=FALSE), silent=TRUE)
    if (inherits(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$res),]
# inverse not equal to input
# (see http://lists.maptools.org/pipermail/proj/2011-November/006015.html)
df[!is.na(df$res) & !df$res,]
# inverse equal to input
row.names(df[!is.na(df$res) & df$res,])
#}
# oce data representation with NAs
ll <- structure(c(12.1823368669203, 11.9149630062421, 12.3186076188739,
12.6207597184845, 12.9955172054652, 12.6316117692658, 12.4680041846297,
12.4366882666609, NA, NA, -5.78993051516384, -5.03798674888479,
-4.60623015708619, -4.43802336997614, -4.78110320396188, -4.99127125409291,
-5.24836150474498, -5.68430388755925, NA, NA), .Dim = c(10L,
2L), .Dimnames = list(NULL, c("longitude", "latitude")))
try(xy0 <- project(ll, "+proj=moll", legacy=TRUE))
if (!PROJis6ormore()) { # legacy=TRUE PROJ >= 6
try(xy1 <- project(ll, "+proj=moll", legacy=FALSE, allowNAs_if_not_legacy=FALSE))
try(xy2 <- project(ll, "+proj=moll", legacy=FALSE, allowNAs_if_not_legacy=TRUE))
if (exists("xy0")) all.equal(xy0, xy2)
```

```
}
if (!exists("xy0")) xy0 <- structure(c(1217100.8468177, 1191302.229156,
1232143.28841193, 1262546.27733232, 1299648.82357849, 1263011.18154638,
1246343.17808186, 1242654.33986052, NA, NA, -715428.207551599,
-622613.577983058, -569301.605757784, -548528.530156422, -590895.949857199,
-616845.926397351, -648585.161643274, -702393.1160979, NA, NA),
.Dim = c(10L, 2L), .Dimnames = list(NULL, c("longitude", "latitude")))
try(ll0 <- project(xy0, "+proj=moll", inv=TRUE, legacy=TRUE))
if (!PROJis6ormore()) { # legacy=TRUE PROJ >= 6
try(ll1 <- project(xy0, "+proj=moll", inv=TRUE, legacy=FALSE, allowNAs_if_not_legacy=FALSE))
try(ll2 <- project(xy0, "+proj=moll", inv=TRUE, legacy=FALSE, allowNAs_if_not_legacy=TRUE))
if (exists("ll0")) all.equal(ll0, ll2)
}
if (exists("ll0")) all.equal(ll0, ll)
## Not run:
set_thin_PROJ6_warnings(TRUE)
library(grid)
GDALinfo(system.file("external/test.ag", package="sp")[1])
x <- readGDAL(system.file("external/test.ag", package="sp")[1])
class(x)
image(x)
summary(x)
x@data[[1]][x@data[[1]] > 10000] <- NA
summary(x)
image(x)

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

```
summary(SP27GTIF)
slot(SP27GTIF, "proj4string")
if (new_proj_and_gdal()) comment(slot(SP27GTIF, "proj4string"))
image(SP27GTIF, col=grey(1:99/100))
GDALinfo(system.file("pictures/cea.tif", package = "rgdal")[1])
(o <- GDALSpatialRef(system.file("pictures/cea.tif", package = "rgdal")[1]))
if (new_proj_and_gdal()) comment(o)
cea <- readGDAL(system.file("pictures/cea.tif", package = "rgdal")[1],
output.dim=c(100,100))
summary(cea)
image(cea, col=grey(1:99/100))
slot(cea, "proj4string")
if (new_proj_and_gdal()) comment(slot(cea, "proj4string"))
fn <- system.file("pictures/erdas_spnad83.tif", package = "rgdal")[1]
erdas_spnad83 <- readGDAL(fn, offset=c(50, 100), region.dim=c(400, 400),
output.dim=c(100,100))
summary(erdas_spnad83)
slot(erdas_spnad83, "proj4string")
if (new_proj_and_gdal()) comment(slot(erdas_spnad83, "proj4string"))
image(erdas_spnad83, col=grey(1:99/100))
erdas_spnad83a <- readGDAL(fn, offset=c(50, 100), region.dim=c(400, 400))
bbox(erdas_spnad83)
bbox(erdas_spnad83a)
gridparameters(erdas_spnad83)
gridparameters(erdas_spnad83a)
tf <- tempfile()
writeGDAL(erdas_spnad83, tf, drivername="GTiff", type="Byte", options=NULL)
erdas_spnad83_0 <- readGDAL(tf)
slot(erdas_spnad83_0, "proj4string")
if (new_proj_and_gdal()) comment(slot(erdas_spnad83_0, "proj4string"))
all.equal(erdas_spnad83, erdas_spnad83_0)
writeGDAL(erdas_spnad83, tf, drivername="GTiff", type="Byte",
options="INTERLEAVE=PIXEL")
erdas_spnad83_1 <- readGDAL(tf)
slot(erdas_spnad83_1, "proj4string")
if (new_proj_and_gdal()) comment(slot(erdas_spnad83_1, "proj4string"))
all.equal(erdas_spnad83, erdas_spnad83_1)
writeGDAL(erdas_spnad83, tf, drivername="GTiff", type="Byte",
options=c("INTERLEAVE=PIXEL", "COMPRESS=DEFLATE"))
erdas_spnad83_2 <- readGDAL(tf)
slot(erdas_spnad83_2, "proj4string")
if (new_proj_and_gdal()) comment(slot(erdas_spnad83_2, "proj4string"))
all.equal(erdas_spnad83, erdas_spnad83_2)

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

tf <- tempfile()
xx <- create2GDAL(erdas_spnad83, type="Byte")
xxx <- copyDataset(xx, driver="PNG")
```

```
saveDataset(xxx, tf)
GDAL.close(xx)
GDAL.close(xxx)
GDALinfo(tf)

tf2 <- tempfile()
writeGDAL(erdas_spnad83, tf2, 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")
t1 <- readGDAL(tf)
image(t1, red=1, green=2, blue=3)
title(main="output PNG RGB")
par(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),
 "CATlist")[[1]]
ds <- GDAL.open(tf)
displayDataset(ds, reset.par=FALSE)
t(col2rgb(getColorTable(ds)[1:4]))
GDAL.close(ds)
fn <- system.file("pictures/test_envi_class.envi", package = "rgdal")[1]
Gi <- GDALinfo(fn, returnColorTable=TRUE, returnCategoryNames=TRUE)
CT <- attr(Gi, "ColorTable")[[1]]
CT
attr(Gi, "CATlist")[[1]]
with <- readGDAL(fn)
with <- readGDAL(fn, silent=TRUE)
table(with$band1)
```

```
table(as.numeric(with$band1))
with1 <- readGDAL(fn, as.is=TRUE)
table(with1$band1)
spplot(with, col.regions=CT)
tf <- tempfile()
cN <- levels(with$band1)
with$band1 <- as.integer(with$band1)-1
writeGDAL(with, tf, drivername="ENVI", type="Int16", colorTable=list(CT),
 catNames=list(cN), mvFlag=11L)
cat(paste(readLines(paste(tf, "hdr", sep=".")), "\n", sep=""), "\n")
wGi <- GDALinfo(tf, returnColorTable=TRUE, returnCategoryNames=TRUE)
CTN <- attr(wGi, "ColorTable")[[1]]
CTN
attr(wGi, "CATlist")[[1]]
withN <- readGDAL(tf)
table(withN$band1)
withN1 <- readGDAL(tf, as.is=TRUE)
table(withN1$band1)
spplot(withN, col.regions=CTN)


# a file with scale and offset
fn <- system.file("pictures/scaleoffset.vrt", package = "rgdal")[1]
g <- GDALinfo(fn)
attr(g, 'ScaleOffset')
g

fl <- system.file("pictures/MR5905167_372.nc", package="rgdal")[1]
if (file.exists(fl)) {
  flstr <- paste0("NETCDF:\"", fl, "\":TEMP")
  if ("netCDF" %in% gdalDrivers()$name) GDALinfo(flstr)
}

set_thin_PROJ6_warnings(TRUE)
ogrDrivers()
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")$warn
options("warn"=1)
cities <- readOGR(dsn=dsn, layer="cities")
str(slot(cities, "data"))
if (new_proj_and_gdal()) comment(slot(cities, "proj4string"))
cities$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)
```

```
str(slot(cities, "data"))
options("warn"=ow)
summary(cities)
table(Encoding(as.character(cities$NAME)))
ogrInfo(dsn=dsn, layer="kiritimati_primary_roads")
OGRSpatialRef(dsn=dsn, layer="kiritimati_primary_roads")
kiritimati_primary_roads <- readOGR(dsn=dsn, layer="kiritimati_primary_roads")
summary(kiritimati_primary_roads)
if (new_proj_and_gdal()) comment(slot(kiritimati_primary_roads, "proj4string"))
ogrInfo(dsn=dsn, layer="scot_BNG")
OGRSpatialRef(dsn=dsn, layer="scot_BNG")
scot_BNG <- readOGR(dsn=dsn, layer="scot_BNG")
summary(scot_BNG)
if (new_proj_and_gdal()) comment(slot(scot_BNG, "proj4string"))
if ("GML" %in% ogrDrivers()$name) {
  dsn <- system.file("vectors/airports.gml", package = "rgdal")[1]
  airports <- try(readOGR(dsn=dsn, layer="airports"))
  if (!inherits(airports, "try-error")) {
    summary(airports)
    if (new_proj_and_gdal()) comment(slot(airports, "proj4string"))
  }
}
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)
if (new_proj_and_gdal()) comment(slot(ps_cant_31, "proj4string"))
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")[1]
ogrInfo(dsn=dsn, layer="Up")
Up <- readOGR(dsn=dsn, layer="Up")
summary(Up)
if (new_proj_and_gdal()) comment(slot(Up, "proj4string"))
dsn <- system.file("vectors/test_trk2.gpx", package = "rgdal")[1]
test_trk2 <- try(readOGR(dsn=dsn, layer="tracks"))
if (!inherits(test_trk2, "try-error")) {
  summary(test_trk2)
  if (new_proj_and_gdal()) comment(slot(test_trk2, "proj4string"))
}
test_trk2pts <- try(readOGR(dsn=dsn, layer="track_points"))
if (!inherits(test_trk2pts, "try-error")) {
  summary(test_trk2pts)
  if (new_proj_and_gdal()) comment(slot(test_trk2pts, "proj4string"))
}
dsn <- system.file("vectors", package = "rgdal")[1]
ogrInfo(dsn=dsn, layer="trin_inca_pl03")
birds <- readOGR(dsn=dsn, layer="trin_inca_pl03")
summary(birds)
if (new_proj_and_gdal()) comment(slot(birds, "proj4string"))
dsn <- system.file("vectors/PacoursIKA2.TAB", package = "rgdal")[1]
```

```
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")[1])
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")$val)
setwd(odir)
options("warn"=ow)
set_thin_PROJ6_warnings(TRUE)
cities <- readOGR(system.file("vectors", package = "rgdal")[1], "cities")
is.na(cities$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="LAUNDER=NO"))
}
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)))
```

```
}

## End(Not run)

## Not run: if ("GML" %in% ogrDrivers()$name) {
  airports <- try(readOGR(system.file("vectors/airports.gml",
    package = "rgdal")[1], "airports"))
  if (class(airports) != "try-error") {
    writeOGR(cities, paste(td, "cities.gml", sep="/"), "cities", driver="GML")
    cities3 <- readOGR(paste(td, "cities.gml", sep="/"), "cities")
  }
}
## End(Not run)
if (!exists("td")) {
  td <- file.path(tempdir(), "rgdal_examples"); dir.create(td)
# BDR 2016-12-15 (MapInfo driver fails writing to directory with ".")
  if(nchar(Sys.getenv("OSGEO4W_ROOT")) > 0) {
    OLDPWD <- getwd()
    setwd(td)
    td <- "."
  }
}
# The GML driver does not support coordinate reference systems
if ("KML" %in% ogrDrivers()$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("OSGEO4W_ROOT")) > 0) {
    setwd(OLDPWD)
}
set_thin_PROJ6_warnings(TRUE)
CRSargs(CRS("+proj=longlat"))
try(CRS("+proj=longlat"))
CRSargs(CRS("+proj=longlat +datum=NAD27"))
CRSargs(CRS("+init=epsg:4267"))
CRSargs(CRS("+init=epsg:26978"))
CRSargs(CRS(paste("+proj=stere +lat_0=52.15616055555555",
"+lon_0=5.38763888888889 +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(CRS(CRSargs(crs)))
set_thin_PROJ6_warnings(TRUE)
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")
```

SGDF2PCT *Convert RGB three band to single band colour table*

Description

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

Usage

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

Arguments

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 streched 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

https://gdal.org/

Examples

```
logo <- system.file("pictures/Rlogo.jpg", package="rgdal")[1]
SGlogo <- readGDAL(logo)
cols <- SGDF2PCT(SGlogo)
SGlogo$idx <- cols$idx
image(SGlogo, "idx", col=cols$ct)
SGlogo <- readGDAL(logo)
cols <- SGDF2PCT(SGlogo, ncolors=64)
SGlogo$idx <- cols$idx
image(SGlogo, "idx", col=cols$ct)
SGlogo <- readGDAL(logo)
cols <- SGDF2PCT(SGlogo, ncolors=8)
SGlogo$idx <- cols$idx
image(SGlogo, "idx", col=cols$ct)
data(meuse.grid)
coordinates(meuse.grid) <- c("x", "y")
gridded(meuse.grid) <- TRUE
fullgrid(meuse.grid) <- TRUE
summary(meuse.grid$dist)
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2), mar=c(1,1,1,1)+0.1)
image(meuse.grid, "dist", breaks=seq(0,1,1/10), col=bpy.colors(10))
RGB <- vec2RGB(meuse.grid$dist, breaks=seq(0,1,1/10), col=bpy.colors(10))
summary(RGB)
meuse.grid$red <- RGB[,1]
meuse.grid$green <- RGB[,2]
meuse.grid$blue <- RGB[,3]
cols <- SGDF2PCT(meuse.grid[c("red", "green", "blue")], ncolors=10,
 adjust.bands=FALSE)
is.na(cols$idx) <- is.na(meuse.grid$dist)
meuse.grid$idx <- cols$idx
image(meuse.grid, "idx", col=cols$ct)
par(opar)
```

```
# Note: only one wrongly classified pixel after NA handling/dropping
# The functions are not written to be reversible
sort(table(findInterval(meuse.grid$dist, seq(0,1,1/10), all.inside=TRUE)))
sort(table(cols$idx))
```

---

showWKT *Show Well-Known Text spatial reference system metadata*

Description

In modern workflows with PROJ >= 6 and GDAL >= 3, use only `showSRID()` DEPRECATED: Use GDAL/OGR spatial reference objects to convert a PROJ.4 representation to a Well-Known Text representation, and report an EPSG code if it can be determined by OGR SRS services.

Usage

```
showP4(wkt, morphFromESRI=FALSE, enforce_xy = NULL)
showSRID(inSRID, format="WKT2", multiline="NO", enforce_xy = NULL, EPSG_to_init=TRUE,
 prefer_proj=NULL)
get_P6_datum_hard_fail()
set_P6_datum_hard_fail(value)
get_thin_PROJ6_warnings()
set_thin_PROJ6_warnings(value)
get_prefer_proj()
set_prefer_proj(value)
get_rgdal_show_exportToProj4_warnings()
set_rgdal_show_exportToProj4_warnings(value)
get_PROJ6_warnings_count()
OSRIsProjected(obj)
```

Arguments

`enforce_xy`	(PROJ6+/GDAL3+) either use global setting (default NULL) or override policy for coordinate ordering easting/x as first axis, northing/y as second axis.
`wkt`	A valid WKT character string representing a spatial reference system
`morphFromESRI`	default TRUE, morph the WKT string from the representation used by ESRI
`inSRID`	Input coordinate reference string
`obj`	valid CRS object
`format`	Output format, default WKT2
`multiline`	Multiline output, either `"NO"` or `"YES"`
`EPSG_to_init`	default TRUE, workaround for PROJ 6.3.0 frailty leading to the dropping of +ellps= and +units=; DATUM seems to disappear in the internal definition
`prefer_proj`	default NULL, if TRUE, use PROJ compiled code directly, rather than FALSE using PROJ via GDAL SRS; if NULL, uses value shown by `get_prefer_proj()` set on startup to TRUE.

value — a logical value. For set_P6_datum_hard_fail(): by default, a deprecated/ignored input DATUM key/value pair on reading a file with PROJ6 will give a warning (default FALSE); if TRUE, an error is triggered, which may be trapped using try. For set_thin_PROJ6_warnings() default FALSE, can be set to TRUE to report only once and count the number of non-issues warnings, retrieved by get_PROJ6_warnings_count(). For set_rgdal_show_exportToProj4_warnings(), default in **rgdal** version 1.5.* TRUE, from 1.6 FALSE. The options("rgdal_show_exportToProj4_war may be used before loading **rgdal** to set the internal logical variables; if the option is set to "all", all warnings reporting CRS degradation stemming from the GDAL OSR function exportToProj4() even if trivial are reported; if set to "thin", all warnings are detected but thinned so that one report is given per function call; if set to "none", the degradations are detected but not reported.

Value

For showWKT, a character string containing the WKT representation of the PROJ.4 string.

Note

The options("rgdal_show_exportToProj4_warnings"="x") may be used before loading **rgdal** to set the internal logical variables; if the option is set to "all", all warnings reporting CRS degradation stemming from the GDAL OSR function exportToProj4() even if trivial are reported; if set to "thin", all warnings are detected but thinned so that one report is given per function call; if set to "none", the degradations are detected but not reported.

Author(s)

Roger Bivand

References

https://gdal.org/tutorials/osr_api_tut.html

See Also

is.projected, CRS-class

Examples

```
set_thin_PROJ6_warnings(TRUE)
cities <- readOGR(system.file("vectors", package = "rgdal")[1], "cities")
readLines(system.file("vectors/cities.prj", package = "rgdal")[1])
showP4(showWKT("+init=epsg:28992"))
exts <- rgdal_extSoftVersion()
run <- new_proj_and_gdal()
if (run) {
  cat(showSRID("EPSG:27700", multiline="YES"), "\n")
}
if (run) {
(prj <- showSRID("EPSG:27700", "PROJ"))
}
```

```
if (run) {
showSRID(paste0(prj, " +datum=OSGB36"), "WKT1")
}
if (run) {
showSRID(paste0(prj, " +towgs84=370.936,-108.938,435.682"), "WKT1")
}
if (run) {
showSRID(paste0(prj, " +nadgrids=OSTN15_NTv2_OSGBtoETRS.gsb"), "WKT1")
}
if (run) {
showSRID(paste0(prj, " +datum=OSGB36"), "WKT2")
}
if (run) {
showSRID(paste0(prj, " +towgs84=370.936,-108.938,435.682"), "WKT2")
}
if (run) {
showSRID(paste0(prj, " +nadgrids=OSTN15_NTv2_OSGBtoETRS.gsb"), "WKT2")
}
if (run) {
showSRID("ESRI:102761", "WKT2")
}
if (run) {
showSRID("OGC:CRS84", "WKT2")
}
if (run) {
showSRID("urn:ogc:def:crs:OGC:1.3:CRS84", "WKT2")
}
if (run) {
try(showSRID("", "WKT2"))
}

OSRIsProjected(CRS("+proj=longlat"))
OSRIsProjected(CRS("+proj=geocent"))
OSRIsProjected(CRS("+proj=geocent +units=km"))
```

SpatialGDAL-class *Class "SpatialGDAL"*

Description

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

Usage

```
## S3 method for class 'SpatialGDAL'
open(con, ..., silent = FALSE, 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

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 https://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]])
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 (CRS) to another, prior to version 1.5 using Proj4 projection arguments. From version 1.5, Well-Known Text 2 (WKT2 2019) strings are used. For simple projection, when no Proj4 +datum tags are used, datum projection does not occur. When datum transformation is required, the datum should be defined 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 prefered to ellipsoid, because the datum always fixes the ellipsoid, but the ellipsoid never fixes the datum.

In addition, before version 1.5 the +towgs84 tag should have been used where needed to make sure that datum transformation would take place. Parameters for +towgs84 were taken from the legacy bundled EPSG file if they are known unequivocally, but could be entered manually from known authorities. Not providing the appropriate +datum and +towgs84 tags led 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.

From version 1.5, spTransform uses the modern PROJ coordinate operation framework for transformations. This avoids pivoting through WGS84 if possible, and uses WKT2 (2019) strings for

source and target CRS often constructed from the bundled EPSG SQLite database. The database is searched for feasible candidate coordinate operations, and the most accurate available is chosen. More details are available in a vignette: `vignette("CRS_projections_transformations")`.

Usage

```
get_transform_wkt_comment()
set_transform_wkt_comment(value)
get_enforce_xy()
set_enforce_xy(value)
get_last_coordOp()
```

Arguments

`value`	A non-NA logical value

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 had to 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) https://proj.org/faq.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 https://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

```
set_thin_PROJ6_warnings(TRUE)
# state
data(state)
states <- data.frame(state.x77, state.center)
states <- states[states$x > -121,]
coordinates(states) <- c("x", "y")
proj4string(states) <- CRS("+proj=longlat +ellps=clrk66")
summary(states)
state.ll83 <- spTransform(states, CRS("+proj=longlat +ellps=GRS80"))
summary(state.ll83)
state.merc <- spTransform(states, CRS=CRS("+proj=merc +ellps=GRS80"))
summary(state.merc)
state.merc <- spTransform(states,
 CRS=CRS("+proj=merc +ellps=GRS80 +units=us-mi"))
summary(state.merc)
# NAD
## Not run:
```

```
if (PROJis6ormore() || (!PROJis6ormore() && projNAD())) {
states <- data.frame(state.x77, state.center)
states <- states[states$x > -121,]
coordinates(states) <- c("x", "y")
proj4string(states) <- CRS("+init=epsg:4267")
print(summary(states))
state.ll83 <- spTransform(states, CRS("+init=epsg:4269"))
print(summary(state.ll83))
state.kansasSlcc <- spTransform(states, CRS=CRS("+init=epsg:26978"))
print(summary(state.kansasSlcc))
SFpoint_NAD83 <- SpatialPoints(matrix(c(-103.869667, 44.461676), nrow=1),
 proj4string=CRS("+init=epsg:4269"))
SFpoint_NAD27 <- spTransform(SFpoint_NAD83, CRS("+init=epsg:4267"))
print(all.equal(coordinates(SFpoint_NAD83), coordinates(SFpoint_NAD27)))
print(coordinates(SFpoint_NAD27), digits=12)
print(coordinates(SFpoint_NAD83), digits=12)
}

## End(Not run)
data(meuse)
coordinates(meuse) <- c("x", "y")
proj4string(meuse) <- CRS("+init=epsg:28992")
# 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))
## Not run:
# Kiritimati
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)
# scot_BNG
scot_BNG <- readOGR(system.file("vectors", package = "rgdal")[1],
   "scot_BNG")
scot_LL <- spTransform(scot_BNG, CRS("+proj=longlat +datum=WGS84"))
plot(scot_LL, axes=TRUE)
grdtxt_LL <- gridat(scot_LL)
grd_LL <- gridlines(scot_LL, ndiscr=100)
summary(grd_LL)
target <- CRS(proj4string(scot_BNG))
grd_BNG <- spTransform(grd_LL, target)
grdtxt_BNG <- spTransform(grdtxt_LL, target)
opar <- par(mfrow=c(1,2))
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)

## End(Not run)
# broke_proj
broke_proj <- FALSE
# https://github.com/OSGeo/PROJ/issues/1525
pv <- .Call("PROJ4VersionInfo", PACKAGE="rgdal")[[2]]
if (pv >= 600 && pv < 620) broke_proj <- TRUE
if (!broke_proj) {
crds <- matrix(data=c(9.05, 48.52), ncol=2)
spPoint <- SpatialPoints(coords=crds,
 proj4string=CRS("+proj=longlat +ellps=sphere +no_defs"))
ob_tran_def <- paste("+proj=ob_tran +o_proj=longlat",
 "+o_lon_p=-162 +o_lat_p=39.25 +lon_0=180 +ellps=sphere +no_defs")
tg <- CRS(ob_tran_def)
# proj4string not propagated in GDAL 3.0.0
a <- spTransform(spPoint, tg, use_ob_tran=TRUE)
a
}
#should be (-5.917698, -1.87195)
if (!broke_proj) {
spTransform(a, CRS("+proj=longlat +ellps=sphere +no_defs"),
 use_ob_tran=TRUE)
}
if (!broke_proj) {
try(spTransform(a, CRS(paste("+proj=tmerc +lat_0=0 +lon_0=9 +k=1",
"+x_0=3500000 +y_0=0 +ellps=bessel +units=m +no_defs")),
 use_ob_tran=TRUE))
}
if (!broke_proj) {
spTransform(spPoint, CRS(paste("+proj=tmerc +lat_0=0 +lon_0=9 +k=1",
"+x_0=3500000 +y_0=0 +ellps=bessel +units=m +no_defs")))
}
if (!broke_proj) {
spTransform(spTransform(a, CRS("+proj=longlat +ellps=sphere +no_defs"),
 use_ob_tran=TRUE), CRS(paste("+proj=tmerc +lat_0=0 +lon_0=9 +k=1",
"+x_0=3500000 +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(Lines(list(Line(crds1), Line(crds2),
Line(crds3)), ID="a")));
slot(spLines, "proj4string") <- CRS("+proj=longlat +ellps=sphere +no_defs");
bbox(spLines);
if (!broke_proj) {
spLines_tr <- spTransform(spLines, tg, use_ob_tran=TRUE);
bbox(spLines_tr)
}
if (!broke_proj) {
```

```
bbox(spTransform(spLines_tr, CRS("+proj=longlat +ellps=sphere"),
 use_ob_tran=TRUE))
}
if (!broke_proj) {
spPolygons <- SpatialPolygons(list(Polygons(list(Polygon(crds1),
Polygon(crds2), Polygon(crds3)), ID="a")));
slot(spPolygons, "proj4string") <- CRS("+proj=longlat +ellps=sphere +no_defs");
bbox(spPolygons);
}
if (!broke_proj) {
spPolygons_tr <- spTransform(spPolygons, tg, use_ob_tran=TRUE);
bbox(spPolygons_tr)
}
if (!broke_proj) {
bbox(spTransform(spPolygons_tr, CRS("+proj=longlat +ellps=sphere"),
 use_ob_tran=TRUE))
}
# added after posting by Martin Ivanov
## Not run:
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, 1704020.20113366), c(424014.398045834, 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=-0.991,1.9072,0.5129,0.025789908,0.0096501,0.0116599,0.0"))
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)

## End(Not run)
## Not run:
# new_proj_and_gdal()
run <- new_proj_and_gdal()
if (run) {
# 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(SRS_string="OGC:CRS84"), data=data.frame(office="RSB"))
nhh_utm_32N_P4 <- 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=0,0,0,0,0,0,0 +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=0,0,0,0,0,0,0 +units=m +no_defs")))
all.equal(coordinates(nhh_utm_32N_P4), coordinates(nhh_tmerc_P4),
 tolerance=1e-9, scale=1)
# UTM == TMERC: PROJ4 <=4.9.2
all.equal(coordinates(nhh_utm_32N_P4), 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(SRS_string="OGC:CRS84"), 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=0,0,0,0,0,0,0 +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=0,0,0,0,0,0,0 +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
}

## End(Not run)
```

Index