Package 'arcgisbinding'

January 25, 2018

Version 1.0.1.228

Date 2018-01-25

Title Bindings for ArcGIS

Author Esri

Maintainer Esri <R_bridge@esri.com>

NeedsCompilation no

Description This package provides classes for loading, converting and exporting ArcGIS datasets and layers in R.

Depends R (>= 3.2.2)

Imports methods

Suggests sp,
sf,
raster

License Apache License 2.0

URL http://esri.com

OS_type windows

Encoding UTF-8

Archs i386, x64

RoxygenNote 5.0.1

R topics documented:

`arc.check_product` *ArcGIS product and license information*

Description

Initialize connection to ArcGIS. Any script running directly from R (i.e. without being called from a Geoprocessing script) should first call `arc.check_product` to create a connection with ArcGIS. Provides installation details on the version of ArcGIS installed that `arcgisbinding` is communicating with.

Usage

```
arc.check_product()
```

Details

Returned details include:

- Product: ArcGIS Desktop (i.e. ArcMap), or ArcGIS Pro. The name of the product connected.
- License level: Basic, Standard, or Advanced are the three licensing levels available. Each provides progressively more functionality within the software. See the "Desktop Functionality Matrix" link for details.
- Build number: The build number of the release being used. Useful in debugging and when creating error reports.
- DLL: The dynamic linked library (DLL) in use allowing ArcGIS to communicate with R.

References

ArcGIS Desktop Functionality Matrix

Note

Additional license levels are available on ArcGIS Desktop: Server, EngineGeoDB, and Engine. These license levels are currently unsupported by this package.

Examples

```
info <- arc.check_product()
info$license # ArcGIS license level
info$version # ArcGIS build number
info$app # product name
info$dll # binding DLL in use
```

---

arc.data2sf *Convert an arc.dataframe object to an sf Simple Feature object*

Description

Convert an ArcGIS `data.frame` to the equivalent `sf` object type. The output types that can be generated: `POINT`, `MULTIPOINT`, `POLYGON`, `MULTIPOLYGON`, `LINESTRING`, `MULTILINESTRING`.

Usage

```
arc.data2sf(x)
```

Arguments

`x`	`data.frame` result of `arc.select`

Examples

```
require(sf)
d <- arc.select(arc.open(system.file("extdata", "ca_ozone_pts.shp", package="arcgisbinding")), 'ozone')
x <- arc.data2sf(d)
## Not run: plot(x)
```

---

arc.data2sp *Convert an arc.dataframe or arc.raster object to an sp SpatialDataFrame object*

Description

Convert an ArcGIS `data.frame` to the equivalent `sp` data frame type. The output types that can be generated: `SpatialPointsDataFrame`, `SpatialLinesDataFrame`, or `SpatialPolygonsDataFrame`. Convert an `arc.raster` object to a `SpatialGridDataFrame` object.

Usage

```
arc.data2sp(x)
```

Arguments

x	data.frame result of arc.select or arc.raster

Examples

```
require(sp)
d <- arc.select(arc.open(system.file("extdata", "ca_ozone_pts.shp", package="arcgisbinding")), 'ozone')
sp.df <- arc.data2sp(d)
## Not run: spplot(sp.df)
```

---

arc.dataset-class *Class "arc.dataset"*

Description

arc.dataset S4 class

Details

The dataset_type slot possible values are described in the referenced "dataset properties – data type" documentation. For feature datasets, extent contains four double values: (xmin, ymin, xmax, ymax). The fields slot includes the details of the ArcGIS data types of the relevant fields, which include data types not directly representable in R.

Slots

path file path or layer name

dataset_type dataset type

extent spatial extent of the dataset

fields list of field names

shapeinfo geometry information (see arc.shapeinfo)

References

1. ArcGIS Help: Dataset properties – dataset type

Methods

arc.select

names

Examples

```
ozone.file <- system.file("extdata", "ca_ozone_pts.shp",
                          package="arcgisbinding")
d <- arc.open(ozone.file)
names(d@fields) # get all field names
arc.shapeinfo(d) # print shape info
d               # print dataset info
```

arc.datasetraster-class *Class "arc.datasetraster"*

Description

`arc.datasetraster` S4 class

Details

TODO

Slots

`.info` internal

`sr` spatial reference

`extent` spatial extent of the dataset

`pixel_type` The pixel type of the referenced raster dataset. (pixel_type)

`compression_type` The compression type

`nrow` return the number of rows

`ncol` return the number of rows

`bands` return rasterdataset bands information

Methods

arc.raster

dim

names

See Also

`arc.open`

arc.delete *delete dataset*

Description

delete dataset

Arguments

`x` `arc.dataset`

`arc.env` *Get geoprocessing environment settings*

Description

Geoprocessing environment settings are additional parameters that affect a tool's results. Unlike parameters, they are not directly input as values. Instead, they are values configured in a separate dialog box, and then and interrogated and used by the script when run.

Usage

```
arc.env()
```

Details

The geoprocessing environment can control a variety of attributes relating to where data is stored, the extent and projection of analysis outputs, tolerances of output values, and parallel processing, among other attributes. Commonly used environment settings include `workspace`, which controls the default location for geoprocessing tool inputs and outputs. See the topics listed under "References" for details on the full range of environment settings that Geoprocessing scripts can utilize.

References

- ArcGIS Help: What is a geoprocessing environment setting?
- ArcGIS Help: Setting geoprocessing environments

Note

- This function is only available from within an ArcGIS session. Usually, it is used to get local Geoprocessing tool environment settings within the executing tool.
- This function can only read current geoprocessing settings. Settings, such as the current workspace, must be configured in the calling Geoprocessing script, not within the body of the R script.

Examples

```
## Not run:
  tool_exec <- function(in_para, out_params)
  {
    env = arc.env()
    wkspath <- env$workspace
    ...
    return(out_params)
  }

## End(Not run)
```

`arc.fromP4ToWkt` *Convert PROJ.4 Coordinate Reference System string to Well-known Text.*

Description

The `arc.fromP4ToWkt` command converts a PROJ.4 coordinate reference system (CRS) string to a well-known text (WKT) representation. Well-known text is used by ArcGIS and other applications to robustly describe a coordinate reference system. Converts PROJ.4 stings which include either the '+proj' fully specified projection parameter, or the '+init' form that takes well-known IDs (WKIDs), such as EPSG codes, as input.

Arguments

`proj4`	PROJ.4 projection string

Details

The produced WKT is equivalent to the ArcPy spatial reference exported string:

```
arcpy.Describe(layer).SpatialReference.exportToString()
```

References

1. OGC specification 12-063r5
2. ArcGIS Help: What are map projections?

Note

The '+init' method currently only works with ArcGIS Pro.

See Also

`arc.fromWktToP4`

Examples

```
arc.fromP4ToWkt("+proj=eqc") # Equirectangular

arc.fromP4ToWkt("+proj=latlong +datum=wgs84") # WGS 1984 geographic

arc.fromP4ToWkt("+init=epsg:2806") # initalize based on EPSG code
```

arc.fromWktToP4 — *Convert a Well-known Text Coordinate Reference System into a PROJ.4 string.*

Description

Convert a well-known text (WKT) coordinate reference system (CRS) string to a PROJ.4 representation. PROJ.4 strings were created as a convenient way to pass CRS information to the command-line PROJ.4 utilities, and have an expressive format. Alternatively, can accept a well-known ID (WKID), a numeric value that ArcGIS uses to specify projections. See the 'Using spatial references' resource for lookup tables which map between WKIDs and given projection names.

Arguments

wkt	WKT projection string, or a WKID integer

References

1. ArcGIS REST API: Using spatial references
2. OGC specification 12-063r5
3. ArcGIS Help: What are map projections?

See Also

arc.fromP4ToWkt

Examples

```
d <- arc.open(system.file("extdata", "ca_ozone_pts.shp",
                          package="arcgisbinding"))
arc.fromWktToP4(arc.shapeinfo(d)$WKT)

arc.fromWktToP4(4326) # use a WKID for WGS 1984, a widely
                      # used standard for geographic coordinates
```

arc.metadata — *Get metadata*

Description

Get metadata

Usage

```
arc.metadata(object)
```

Arguments

object	arc.dataset-class object

`arc.open` *Open dataset, table, or layer*

Description

Open ArcGIS datasets, tables, rasters and layers. Returns a new `arc.dataset-class` object which contains details on both the spatial information and attribute information (data frame) contained within the dataset.

Arguments

`path`	file path or layer name

Value

An `arc.dataset` object

Supported Formats

- `Feature Class`: A collection of geographic features with the same geometry type (i.e. point, line, polygon) and the same spatial reference, combined with an attribute table. Feature classes can be stored in a variety of formats, including: files (e.g. Shapefiles), Geodatabases, components of feature datasets, and as coverages. All of these types can be accessed using the full path of the relevant feature class (see note below on how to specify path names).
- `Layer`: A layer references a feature layer, but also includes additional information necessary to symbolize and label a dataset appropriately. `arc.open` supports active layers in the current ArcGIS session, which can be addressed simply by referencing the layer name as it is displayed within the application. Instead of referencing file layers on disk (i.e. `.lyr` and `.lyrx` files), the direct reference to the actual dataset should be used.
- `Table`: Tables are effectively the same as data frames, containing a collection of records (or observations) organized in rows, with columns storing different variables (or fields). Feature classes similarly contain a table, but include the additional information about geometries lacking in a standalone table. When a standalone table is queries for its spatial information, e.g. `arc.shape(table)`, it will return `NULL`. Table data types include formats such as text files, Excel spreadsheets, dBASE tables, and INFO tables.
- `rasters`: A raster dataset TODO

References

- What is the difference between a shapefile and a layer file?
- ArcGIS Help: What is a layer?
- ArcGIS Help: What are tables and attribute information?

Note

Paths must be properly quoted for the Windows platform. There are two styles of paths that work within R on Windows:

- Doubled backslashes, such as: `C:\\Workspace\\archive.gdb\\feature_class`.
- Forward-slashes such as: `C:/Workspace/archive.gdb/feature_class`.

Network paths can be accessed with a leading \\\\host\share or //host/share path. To access tables and data within a Feature Dataset, reference the full path to the dataset, which follows the structure: <directory>/<Geodatabase Name>/<feature dataset name>/<dataset name>. So for a table called table1 located in a feature dataset fdataset within a Geodatabase called data.gdb, the full path might be: C:/Workspace/data.gdb/fdataset/table1

See Also

arc.dataset-class

arc.datasetraster-class

Examples

```
## open feature
filename <- system.file("extdata", "ca_ozone_pts.shp",
                        package="arcgisbinding")
d <- arc.open(filename)
cat('all fields:', names(d@fields), fill = TRUE) # print all fields

## open raster
filename <- system.file("pictures", "logo.jpg", package="rgdal")
d <- arc.open(filename)
dim(d) # show raster dimension
```

---

arc.progress_label *Set progressor label for Geoprocessing dialog box*

Description

Geoprocessing tools have a progressor, which includes both a progress label and a progress bar. The default progressor continuously moves back and forth to indicate the script is running. Using arc.progress_label and arc.progress_pos allows fine control over the script progress. Updating the progressor isn't necessary, but is useful in situations where solely outputting messages to the dialog is insufficient to communicate script progress.

Usage

```
arc.progress_label(label)
```

Arguments

label	Progress Label

Details

Using arc.progress_label allows control over the label that is displayed at the top of the running script. For example, it might be used to display the current step of the analysis taking place.

References

Understanding the progressor in script tools

Note

- Currently only functions in ArcGIS Pro, and has no effect in ArcGIS Desktop.
- This function is only available from within an ArcGIS session, and has no effect when run from the command line or in background geoprocessing.

See Also

arc.progress_pos, "Progress Messages" example Geoprocessing script

Examples

```
## Not run:
arc.progress_label("Calculating bootstrap samples...")

## End(Not run)
```

---

arc.progress_pos *Set progressor position for Geoprocessing dialog box*

Description

Geoprocessing tools have a progressor, which includes both a progress label and a progress bar. The default progressor continuously moves back and forth to indicate the script is running. Using arc.progress_label and arc.progress_pos allow fine control over the script progress. Updating the progressor isn't necessary, but is useful in situations where solely outputting messages to the dialog is insufficient to communicate script progress.

Usage

```
arc.progress_pos(pos = -1)
```

Arguments

pos	Progress position (in percent)

Details

Using arc.progress_pos allows control over the progrssor position displayed at the top of the running script. The position is an integer percentage, 0 to 100, that the progress bar should be set to, with 100 indicating the script has completed (100%).

Setting the position to -1 resets the progressor to the default progressor, which continuously moves to indicate the script is running.

References

Understanding the progressor in script tools

Note

- Currently only functions in ArcGIS Pro, and has no effect in ArcGIS Desktop.
- This function is only available from within an ArcGIS session, and has no effect when run from the command line or in background geoprocessing.

See Also

arc.progress_label, "Progress Messages" example Geoprocessing script

Examples

```
## Not run:
arc.progress_pos(55)

## End(Not run)
```

---

arc.raster *Create arc.raster object*

Description

Create arc.raster object

Usage

```
arc.raster(object, bands, ...)

## S4 method for signature 'arc.datasetraster'
arc.raster(object, bands, ...)

## S4 method for signature '`NULL`'
arc.raster(object, path, dim, nrow, ncol, nband, extent,
  origin_x, origin_y, cellsize_x, cellsize_y, pixel_type, nodata, sr, ...)
```

Arguments

object	arc.datasetraster-class object.
bands	integer vector of bands (default: all bands).
...	optional additional arguments such as nrow, ncol, extent, pixel_type, resample_type to be passed to the method.

Value

arc.raster returns a raster object (type of arc.raster-class).

Examples

```
## create and resample raster
r.file <- system.file("pictures", "cea.tif", package="rgdal")
r <- arc.raster(arc.open(r.file), nrow=200, ncol=200, resample_type="CubicConvolution")
r
stopifnot(r$nrow == 200)
```

`arc.raster-class` *Reference Class "arc.raster"*

Description

TODO

Fields

`sr` Get or set Spacial Reference

`extent` Get or set extent. Uset it to read portion of raster.

`nrow` Get or set number of row.

`ncol` Get or set number of column.

`cellsize` Get pixel size.

`pixel_type` Get or set pixel type.

`pixel_depth` Get pixel depth

`nodata` Get or set nodata value

`resample_type` Get or set resampling type.

`colormap` Get or set color map table. Return is a vector of 256 colors in the RGB format.

`bands` Get raster band information

Methods

names

dim

arc.write

`$save_as(path, opt)`

`$commit(opt)` End writing. `opt` - aditional parameter (default: build-stats)

`$pixel_block(ul_x, ul_y, nrow, ncol, bands)` Read pixel values. `ul_x, ul_y` - upper left corner in pixels `nrow, ncol` - size in pixels

`$write_pixel_block(values, ul_x, ul_y, ncol, nrow)` Write pixel values.

`$attribute_table()` Query raster attribute table

Examples

```
## read 5x5 pixel block with 10,10 offset
r.file <- system.file("pictures", "cea.tif", package="rgdal")
r <- arc.raster(arc.open(r.file))
v <- r$pixel_block(ul_x = 10L, ul_y = 10L, nrow = 5L, ncol= 5L)
stopifnot(length(v) == 25)

## process big raster
r2 = arc.raster(NULL, path=tempfile("r2", fileext=".img"),
                dim=dim(r), pixel_type=r$pixel_type, nodata=r$nodata,
                extent=r$extent,sr=r$sr)
for (i in 1L:r$nrow)
{
```

```
  v <- r$pixel_block(ul_y = i - 1L, nrow = 1L)
  r2$write_pixel_block(v * 1.5, ul_y = i - 1L, nrow = 1L, ncol = r$ncol)
}
r2$commit()

## resample raster
r <- arc.raster(arc.open(r.file), nrow=200L, ncol=200L, resample_type="BilinearGaussBlur")

## save to a different format
r$save_as(tempfile("new_raster", fileext=".img"))

## get and compare all pixel values
r.file <- system.file("pictures", "logo.jpg", package="rgdal")
rx <- raster::brick(r.file)
r <- arc.raster(arc.open(r.file))
stopifnot(all(raster::values(rx) == r$pixel_block()))
```

`arc.select` *Load dataset to data.frame*

Description

Load dataset to a standard data frame.

Usage

```
arc.select(object, fields, where_clause, selected, sr, ...)

## S4 method for signature 'arc.table'
arc.select(object, fields, where_clause, selected, sr,
  ...)
```

Arguments

`object`	arc.dataset-class object
`fields`	string, or list of strings, containing fields to include (default: all)
`where_clause`	SQL where clause
`selected`	use only selected records (if any) when dataset is a layer or standalone table
`sr`	transform geometry to Spatial Reference (default: object@sr)
`...`	Additional arguements (currently ignored)

Value

`arc.select` returns a `data.frame` object (type of `arc.data`).

Note

If dataset includes the `arc.feature` attribute, the "shape" of class `arc.shape-class` will be attached to the resulting `data.frame` object.

See Also

arc.open

Examples

```
## read all fields
ozone.file <- system.file("extdata", "ca_ozone_pts.shp",
                          package="arcgisbinding")
d <- arc.open(ozone.file)
df <- arc.select(d, names(d@fields))
head(df, n=3)

## read 'name', 'fid' and geometry
df <- arc.select(d, c('fid', 'ozone'), where_clause="fid < 5")
nrow(df)

## transform points to "+proj=eqc"
df <- arc.select(d,"fid", where_clause="fid<5", sr="+proj=eqc")
arc.shape(df)
```

---

arc.shape *Get arc.shape object*

Description

Get arc.shape-class from arc.data

Arguments

df	arc.dataframe

See Also

arc.select

Examples

```
d <- arc.open(system.file("extdata", "ca_ozone_pts.shp",
                          package="arcgisbinding"))

df <- arc.select(d, 'ozone')
shp <- arc.shape(df)
length(shp$x)
```

arc.shape-class *Class "arc.shape"*

Description

arc.shape is geometry collection

Note

arc.shape is attached to an ArcGIS data.frame as the attribute "shape". Each element corresponds to one record in the input data frame. Points are presented as an array of lists, with each list containing (x, y, Z, M), where

Examples

```
d <- arc.open(system.file("extdata", "ca_ozone_pts.shp",
                          package="arcgisbinding"))
arc.df <- arc.select(d, "FID")

shape <- arc.shape(arc.df)
# create data.frame with X and Y columns
df <- data.frame(arc.df, X=shape$x, Y=shape$y)
# print out row #42
df[42,]
```

arc.shape2sf *Convert Esri shape to sfc simple feature geometry*

Description

Convert arc.shape-class to sfc simple feature geometry: POINT, MULTIPOINT, POLYGON, MULTIPOLYGON, LINESTRING, MULTILINESTRING.

Usage

```
arc.shape2sf(shape)
```

Arguments

shape	arc.shape-class

See Also

arc.shape

Examples

```
require(sp)
d <- arc.select(arc.open(system.file("extdata", "ca_ozone_pts.shp", package="arcgisbinding")), 'ozone')
x <- arc.shape2sp(arc.shape(d))
## Not run: plot(x)
```

`arc.shape2sp` *Convert Esri shape to sp spatial geometry*

Description

Convert `arc.shape-class` to sp spatial geometry: `SpatialPoints`, `SpatialLines`, or `SpatialPolygons`.

Usage

```
arc.shape2sp(shape, wkt)
```

Arguments

shape	`arc.shape-class`
wkt	WKT spatial reference

See Also

`arc.shape`

Examples

```
require(sp)
d <- arc.select(arc.open(system.file("extdata", "ca_ozone_pts.shp", package="arcgisbinding")), 'ozone')
x <- arc.shape2sp(arc.shape(d))
## Not run: plot(x)
```

`arc.shapeinfo` *Shape Information*

Description

`arc.shapeinfo` provides details on what type of geometry is stored within the dataset, and the spatial reference of the geometry. The well-known text, `WKT`, allows interoperable transfer of the spatial reference system (CRS) between environments. The `WKID` is a numeric value that ArcGIS uses to precisely specify a projection.

Arguments

object	arc.dataset-class object

Slots

`type` geometry type: "Point", "Polyline", or "Polygon"

`hasZ` TRUE if geometry includes Z-values

`hasM` TRUE if geometry includes M-values

`WKT` well-known text representation of the shape's spatial reference

`WKID` well-known ID of the shape's spatial reference

References

1. ArcGIS REST API: Using spatial references
2. Spatial reference lookup

See Also

`arc.dataset-class` `arc.shape-class`

Examples

```
d <- arc.open(system.file("extdata", "ca_ozone_pts.shp",
                          package="arcgisbinding"))
# from arc.dataset
info <- arc.shapeinfo(d)
info$WKT   # print dataset spatial reference

# from arc.shape
df <- arc.select(d, 'ozone')
arc.shapeinfo(arc.shape(df))
```

---

arc.sp2data *deprecated Convert a sp SpatialDataFrame object to an arc.dataframe object*

Description

deprecated Convert sp `SpatialPointsDataFrame`, `SpatialPolygonsDataFrame`, and `SpatialLinesDataFrame` objects to an ArcGIS-compatible `data.frame`.

Arguments

`sp.df`	`SpatialPointsDataFrame`, `SpatialPolygonsDataFrame`, or `SpatialLinesDataFrame`

See Also

`arc.data2sp`

`arc.write` *Write dataset, raster, feature, table or layer*

Description

Export a `data.frame` object to an ArcGIS dataset. If the data frame includes a spatial attribute, this function writes a feature dataset. If no spatial attribute is found, a table is instead written.
Export a `arc.raster`, `raster::RasterLayer` or `raster::RasterBrick` object to an ArcGIS raster dataset.

Arguments

`path`	full output path
`data`	input source. Accepts `data.frame`, spatial `data.frame`, `SpatialPointsDataFrame`, `SpatialLinesDataFrame`, and `SpatialPolygonsDataFrame`, `arc.raster`, `raster::RasterLayer`, `raster::RasterBrick` objects.
`...`	Optional parameters • `coords` list containing geometry. Accepts `Spatial` objects. Put field names if `data` is data.frame and consists coordinates. • `shape_info` required argument if `data` has no spatial attribute • `overwrite` overwrite existing dataset. default = FALSE.

Details

Supports a variety of output formats. Below are pairs of example paths and the resulting data types:

- `C:/place.gdb/fc`: File Geodatabase Feature Class
- `C:/place.gdb/fdataset/fc`: File Geodatabase Feature Dataset
- `in_memory\logreg`: In-memory workspace (must be run in ArcGIS Session)
- `C:/place.shp`: Esri Shapefile
- `C:/place.dbf`: Table
- `C:/place.gdb/raster`: File Geodatabase Raster when `data` parameter is `arc.raster` or `Raster*` object
- `C:/image.img`: ERDAS Imaging
- `C:/image.tif`: Geo TIFF

References

- What is the difference between a shapefile and a layer file?
- ArcGIS Help: What is a layer?

See Also

`arc.dataset-class`, `arc.open`, `arc.raster`

Examples

```
## write as a shapefile
fc <- arc.open(system.file("extdata", "ca_ozone_pts.shp", package="arcgisbinding"))
d <- arc.select(fc, 'ozone')
d[1,] <- 0.6
arc.write(tempfile("ca_new", fileext=".shp"), d)

## create and write to a new file geodatabase
fgdb_path <- file.path(tempdir(), "data.gdb")

data(meuse, package="sp")
## create feature dataset 'meuse'
arc.write(file.path(fgdb_path, "meuse\\pts"), data=meuse, coords=c("x", "y", "elev"), shape_info=list(type='
data(meuse.riv, package="sp")
riv <- sp::SpatialPolygons(list(sp::Polygons(list(sp::Polygon(meuse.riv)),"meuse.riv")))
## write only geometry
arc.write(file.path(fgdb_path, "meuse\\riv"), coords=riv)

## write as table
arc.write(file.path(fgdb_path, "tlb"), data=list('f1'=c(23,45), 'f2'=c('hello', 'bob')))

## from scratch as feature class
arc.write(file.path(fgdb_path, "fc_pts"), data=list('data'=rnorm(100)),
          coords=list(x=runif(100,min=0,max=10),y=runif(100,min=0,max=10)),
          shape_info=list(type='Point'))

## write Raster
# make SpatialPixelsDataFrame
data(meuse.grid, package="sp")
sp::coordinates(meuse.grid) = c("x", "y")
sp::gridded(meuse.grid) <- TRUE
meuse.grid@proj4string=sp::CRS(arc.fromWktToP4(28992))

arc.write(file.path(fgdb_path, "meuse_grid"), meuse.grid)

## write using a RasterLayer object
r <- raster::raster(ncol=10, nrow=10)
raster::values(r) <- runif(raster::ncell(r))
arc.write(file.path(fgdb_path, "raster"), r)
```

`arcgisbinding` *Bindings for ArcGIS*

Description

This package provides classes for loading, converting and exporting ArcGIS datasets and layers in R.

Introduction

For a complete list of exported functions, use `library(help = "arcgisbinding")`.

References

- sp package
- CRAN Task View: Analysis of Spatial Data

`as.raster` *Create RasterLayer or RasterBrick (raster package)*

Description

Create Rraster* object from arc.raster TODO

Arguments

`x`	arc.raster-class object
`...`	TODO

Value

`RasterLayer` or `RasterBrick`

Examples

```
## convert arc.raster to Rasterlayer object
r.file <- system.file("pictures", "logo.jpg", package="rgdal")
r <- arc.raster(arc.open(r.file))
rx <- as.raster(r)
```

`dim` *Dimensions of an object*

Description

Get or set raster dimention

Arguments

`x`	object arc.raster-class or arc.datasetraster-class object

Value

integer vector of `c(nrow, ncol, nband)` dim,arc.raster-method

dplyr *dplyr support*

Description

dplyr methods for arc.data objects: filter, arrange, mutate, select, group_by

Usage

```
filter.arc.data(.data, ..., .dots)

arrange.arc.data(.data, ..., .dots)

mutate.arc.data(.data, ..., .dots)

group_by.arc.data(.data, ..., add)

ungroup.arc.data(x, ...)
```

names *Get names*

Description

Return names of columns when x is arc.table-class. Return band names when x is arc.raster-class or arc.datasetraster-class.

Arguments

x	object

Value

string vector

pixeltypes *Pixel types*

Description

The pixel type of the referenced raster dataset.

The types are

- "U1" 1 bit
- "U2" 2 bits
- "U4" 4 bits
- "U8" Unsigned 8 bit integers
- "S8" 8 bit integers
- "U16" Unsigned 16 bit integers
- "S16" 16 bit integers
- "U32" Unsigned 32 bit integers
- "S32" 32 bit integers
- "F32" Single precision floating point
- "F64" Double precision floating point

`resampletypes` *Resample types*

Description

Resample types

Supported

- "NearestNeighbor" Nearest neighbor assignment. This is the default
- "BilinearInterpolation" Bilinear interpolation
- "CubicConvolution" Cubic convolution
- "Majority" TODO
- "BilinearInterpolationPlus" TODO
- "BilinearGaussBlur" TODO
- "BilinearGaussBlurPlus" TODO
- "Average" TODO
- "Minimum" TODO
- "Average" TODO
- "VectorAverage" TODO

Index