Package ‘arcgisbinding’

July 12, 2018

Version 1.0.1.232
Date 2018-07-12
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 (>= 3.3.2)
Imports methods
Suggests sp, sf, raster, rgdal, dplyr
License file LICENSE
URL http://esri.com
BugReports https://github.com/R-ArcGIS/r-bridge/issues
OS_type windows
Encoding UTF-8
Archs i386, x64

R topics documented:

  arcgisbinding-package .............................................. 2
  arc.check_product .................................................. 3
  arc.data .......................................................... 4
  arc.data2sf ......................................................... 5
  arc.data2sp ........................................................ 6
  arc.dataset-class .................................................. 7
  arc.datasetraster-class .......................................... 8
  arc.datasetrastermosaic-class ................................... 9
  arc.delete .......................................................... 10
  arc.env ............................................................ 10
Description

Collection of classes and functions 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](#) (Classes and Methods for Spatial Data)
- [sf package](#) (Simple Features for R)
- [raster package](#) (Geographic Data Analysis and Modeling)
- [CRAN Task View: Analysis of Spatial Data](#)
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. Failure to run this function successfully implies a problem with ArcGIS installation or environment variables for ArcGIS.

Usage

```r
arc.check_product()
```

Value

A named list is returned with the following components:

- `app` Product: ArcGIS Desktop (i.e. ArcMap), or ArcGIS Pro. The name of the product connected.
- `license` 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.
- `version` Build number: The build number of the release being used. Useful in debugging and when creating error reports.
- `dll` 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

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

arc.data class and methods

Usage

```r
## S3 method for class 'arc.data'
x[i, j, drop]

## dplyr methods:
## S3 method for class 'arc.data'
filter(.data, ..., .dots)
## S3 method for class 'arc.data'
arrange(.data, ..., .dots)
## S3 method for class 'arc.data'
mutate(.data, ..., .dots)
## S3 method for class 'arc.data'
group_by(.data, ..., add)
## S3 method for class 'arc.data'
ungroup(x, ...)
```

Arguments

- `i, j, ...` indices specifying elements to subset
- `drop` if TRUE coerce the result to the lowest possible dimension and remove the geometry attribute
- `x` A arc.data object
- `.data` A arc.data object
- `.dots` other arguments (see package dplyr)
- `add` To add to the existing groups, use add = TRUE

Details

TODO arc.data object is data.frame with geometry attribute. To access geometry use `arc.shape`.

Extends

Class data.frame, directly.

dplyr methods

- filter: Return rows with matching conditions
- arrange: Arrange rows by variables
- mutate, transmute: Add new variables
- select: Select/rename variables by name
• group_by: Group by one or more variables
• slice: Select rows by position
• distinct: Select distinct/unique rows

Note
You can display the arc.data object. Geometry information, first 5 and last 3 row will be showed.

See Also
arc.shape, arc.open, arc.select

Examples

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

d
## Not run:
geometry type : Point
WKIG : PROJCS["USA_Coniguous_Albers_Equal_Area_Conic",GEOGCS["GCS_... 
WKID : 102003

<table>
<thead>
<tr>
<th>FID</th>
<th>LATITUDE</th>
<th>LONGITUDE</th>
<th>ELEVATION</th>
<th>OZONE</th>
<th>X</th>
<th>Y</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>39.1447</td>
<td>-123.2065</td>
<td>194</td>
<td>0.04650</td>
<td>-2298082</td>
<td>515557.4</td>
<td>Value_0</td>
</tr>
<tr>
<td>2</td>
<td>39.4030</td>
<td>-123.3491</td>
<td>420</td>
<td>0.04969</td>
<td>-2301588</td>
<td>546772.7</td>
<td>Value_1</td>
</tr>
<tr>
<td>3</td>
<td>37.7661</td>
<td>-122.3978</td>
<td>5</td>
<td>0.05000</td>
<td>-2273948</td>
<td>347691.4</td>
<td>Value_2</td>
</tr>
<tr>
<td>4</td>
<td>37.9508</td>
<td>-122.3569</td>
<td>23</td>
<td>0.05799</td>
<td>-2264847</td>
<td>366623.2</td>
<td>Value_3</td>
</tr>
<tr>
<td>5</td>
<td>36.6986</td>
<td>-121.6354</td>
<td>36</td>
<td>0.05860</td>
<td>-2241776</td>
<td>214412.1</td>
<td>Value_0</td>
</tr>
</tbody>
</table>

## End(Not run)

# subset rows 1,3 and 5 with corresponding features
d135 <- d[c(1,3,5),]

# dplyr support
require("dplyr")
filter(d, elevation > 1800)

#add new elevation column in meters
mutate(d, elevm = elevation * 0.3048)

arc.data2sf

Convert an 'arc.data' object to an 'sf' Simple Feature object

Description
Convert an ArcGIS arc.data to the equivalent sf object type. The output types that can be generated: POINT, MULTIPoint, POLYGON, MULTIPOLYGON, LINestring, MULTILINESTRING.
**Usage**

```r
carc.data2sf(x)
```

**Arguments**

- `x`  
  - *arc.data* object, result of *arc.select*.

**See Also**

- *arc.open*, *arc.select*

**Examples**

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

---

**arc.data2sp**

Convert an *arc.data* or *arc.raster* object to an sp 'SpatialDataFrame' object

**Description**

Convert an ArcGIS *arc.data* 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**

```r
carc.data2sp(x)
```

**Arguments**

- `x`  
  - *arc.data* object, result of *arc.select* or *arc.raster*.

**See Also**

- *arc.open*, *arc.select*

**Examples**

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

- `.info` internal
- `path` file path or layer name
- `dataset_type` dataset type

Methods

- `arc.delete`
- `arc.metadata`

References

1. ArcGIS Help: Dataset properties – dataset type

See Also

- `arc.open`, `arc.table-class`, `arc.feature-class`, `arc.datasetraster-class`, `arc.datasetrastermosaic-class`

Examples

```r
ozone.file <- system.file("extdata", "ca_ozone_pts.shp", package="arcgisbinding")
d <- arc.open(ozone.file)
d # print dataset info
```
arc.datasetraster-class

Class "arc.datasetraster"

Description

arc.datasetraster S4 class. Dataset class for raster objects. Creates a dataset object with type = raster.

Details

A raster dataset is any valid raster format organized into one or more bands. Each band consists of an array of pixels (cells), and each pixel has a value. A raster dataset has at least one band. Raster data is a discrete data representation in which space is divided into uniform cells, or pixels.

Extends

Class arc.dataset-class, directly.

\[
\text{arc.dataset-class} \downarrow \text{arc.datasetraster-class}
\]

Slots

sr Spatial reference.
extent Spatial extent of the dataset. The Extent describes the rectangle (boundary) containing all the raster dataset’s data.
pixel_type The pixel type of the referenced raster dataset.
compression_type The compression type.
nrow The number of rows.
ncol The number of columns.
bands raster dataset bands information.

Methods

\textbf{arc.raster} Create a \texttt{arc.raster} object
\textbf{dim} retrieves dimensions of a \texttt{arc.dataset} object
\textbf{names} return bands names
\textbf{arc.write} TODO

References

1. ArcGIS Help: Raster dataset properties

See Also

\texttt{arc.raster}, \texttt{arc.write}
Description

arc.datasetrastermosaic S4 class. Dataset class for mosaic objects.

Details

Mosaic datasets are made up of a collection of rasters. Mosaic structure efficiently stores and manages multiple rasters for visualization and analysis. Detailed information about mosaic datasets can be found in ArcGIS reference for mosaic datasets.

R-ArcGIS bridge handles mosaic data I/O using the arc.open() function. The mosaic dataset opened using arc.open can be processed on the fly by converting it to a raster object within R using the arc.raster function. Properties of a mosaic dataset such as extent, pixel_type, nrow, ncol and mosaicking rules. Mosaicking rules determine how a series of potentially intercepting rasters are displayed as a single raster. Mosaicking rules go beyond only visualization and can be used to stitch together different rasters making up a mosaic.

Mosaicking rules define how intersections between different rasters within the mosaic dataset are handled and are made up of method and operator. Simply put, method defines which raster will be placed on top of the other for visualization in cases where they overlap and operator defines how the intersection between overlapping rasters in the mosaic dataset will be handled. The information on mosaicking rules can be found under ArcGIS reference for mosaicking rules.

Extends

Class arc.feature-class, arc.datasetraster-class directly and arc.table-class by class "arc.feature-class", arc.dataset-class by class "arc.table-class".

References

1. ArcGIS Help: What is a mosaic dataset?

See Also

arc.open, arc.raster, arc.select
**Description**

delete dataset

**Usage**

```r
arc.delete(x, ...)  
## S4 method for signature 'arc.dataset'  
arc.delete(x, ...)
```

**Arguments**

- `x` string full path or arc.dataset object
- `...` reserved

**Value**

logical, TRUE on success.

**Examples**

```r
table_path <- file.path(tempdir(), "data.gdb", "mytable")  
arc.write(table_path, data=list('f1'=c(23,45), 'f2'=c('hello', 'bob')))

# delete table  
arc.delete(table_path)

# delete database  
arc.delete(dirname(table_path))
```

---

**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**

```r
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.

Value

return environment list

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

```R
## Not run:
tool_exec <- function(in_params, out_params)
{
  env = arc.env()
  wksPath <- env$workspace
  ...
  return(out_params)
}
## End(Not run)
```

---

arc.feature-class  

Class "arc.feature"

Description

arc.feature S4 class.

Details

Container for shape information pertaining to extent and shape from a table class.
Extends

Class `arc.table-class`, directly and `arc.dataset-class` by class "arc.table".

```
arc.dataset-class
   ↓
arc.table-class
   ↓
arc.feature-class
```

Slots

- `shapeinfo` geometry information (see `arc.shapeinfo`)
- `extent` spatial extent of the dataset

Methods

- `arc.select` TODO
- `names` return names of columns
- `arc.shapeinfo` return geometry information

See Also

`arc.open, arc.dataset-class, arc.table-class, arc.datasetraster-class, arc.datasetrastermosaic-class`

Examples

```r
do 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.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 strings 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.

Usage

```
arcc.fromP4ToWkt(proj4)
```
Convert a Well-known Text Coordinate Reference System into a PROJ.4 string.

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.

arc.fromWktToP4(wkt)
arc.open

Arguments

wkt     WKT projection string, or a WKID integer

Value

return PROJ.4 string

References

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

See Also

arc.fromWktToP4

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.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.

Usage

arc.open(path)

Arguments

path     file path (character) or layer name (character)

Value

An arc.dataset-class 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 queried 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**: Rasters represent continuous geographic data in cells, or pixels, of equal size (square or rectangular). Spatial data represented on this rasters are also known as grided data. In contrast to spatial data structures represented in feature classes, rasters contain information on spatially continuous data.

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.select`, `arc.raster`, `arc.write`

Examples

```r
## open feature
filename <- system.file("extdata", "ca_ozone_pts.shp",
    package="arcgisbinding")
```
arc.progress_label

```r
#open raster

d <- arc.open(filename)
cat('all fields:', names(d@fields), fill = TRUE) # print all fields

## show raster dimension

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

---

**arc.progress_label**  
*Set progress 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**

```r
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**

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

## End(Not run)
```
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 progressor 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

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

```python
# End(Not run)
```
arc.raster object

Load or create "arc.raster" object

Description

Methods to create an arc.raster object from scratch, extent, arc.open object or a raster file (inside or outside of a file geodatabase).

Usage

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

# S4 method for signature 'arc.datasetrastermosaic'
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`: codearc.datasetraster-class object.
- `bands`: optional, integer. List of bands to read (default: all bands).
- `...`: optional additional arguments such as nrow, ncol, extent, pixel_type, resample_type to be passed to the method.
- `path`: file path (character) or layer name (character).
- `dim`: optional. List for number of rows and columns of the raster.
- `nrow`: optional, integer > 0. Number of rows for the raster or mosaic dataset. The default is `object@nrow`.
- `ncol`: optional, integer > 0. Number of columns for the raster or mosaic dataset. The default is `object@ncol`.
- `nband`: integer > 0. Number of bands to create.
- `extent`: optional, list. Extent of raster to be read. The default is `object@extent`.
- `origin_x`: optional. Minimum x coordinate.
- `origin_y`: optional. Minimum y coordinate.
- `cellsizex_x`: optional. Size of pixel in x-axis.
- `cellsizex_y`: optional. Size of pixel in y-axis.
- `pixel_type`: optional. Type of raster pixels. For details about different pixel types see pixel_type. See also ArcGIS Help: Pixel Types. The default is `object@pixel_type`.
- `nodata`: numeric, value for no data values.
- `sr`: optional transform raster to spatial reference. The default is `object@sr`.

Value

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

1. ArcGIS Help: Raster Introduction
2. ArcGIS Help: Pixel Types
3. ArcGIS Help: Mosaic Introductions
4. ArcGIS Help: Mosaicking Rules

See Also

arc.open, arc.write, arc.raster-class

Examples

## resample raster

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

## Not run:
> r
type : Raster
pixel_type : U8 (8bit)
nrow : 200
ncol : 200
resample_type : CubicConvolution
cellsize : 154.256892046808, 154.557002731725
nodata : NA
extent : xmin=-28493.17, ymin=4224973, xmax=2358.212, ymax=4255885
WKT : PROJCS[North_American_1927_Cylindrical_Equal_Area", GEOGCS["
band : Band_1
## End(Not run)

## create an empty raster

r = arc.raster(NULL, path=tempfile("new_raster", fileext=".img"), extent=c(0, 0, 100, 100), nrow=100, ncol=100)
stopifnot(all(dim(r) == c(100, 100, 5)))

## Not run:
> dim(r)
nrow ncol nband
100 100 5
## End(Not run)
```

arc.raster-class reference class "arc.raster"

Description

A raster dataset is any valid raster format organized into one or more bands. Each band consists of an array of pixels (cells), and each pixel has a value. A raster dataset has at least one band. Raster data is a discrete data representation in which space is divided into uniform cells, or pixels.
Fields

sr  Get or set spacial reference
extent  Get or set extent. Use it to read a portion of the raster.
nrow  Get or set number of rows.
ncol  Get or set number of columns.
cellsize  Get pixel size.
pixel_type  Get or set pixel type. For details see ArcGIS help on pixel types.
pixel_depth  Get pixel depth. Pixel depth/Bit depth (1, 2, 4, 8, 16, 32, 64). For details see ArcGIS help on pixel types.
nodata  Get or set nodata value
resample_type  Get or set resampling type. For details see ArcGIS help on rasampling.
colormap  Get or set color map table. Return is a vector of 256 colors in the RGB format.
bands  Get list of raster bands
band  Get a single raster band

Methods

names  return bands names
dim  retrieves dimensions
$show()  show object
$pixel_block(ul_x, ul_y, nrow, ncol, bands)  Read pixel values.
   ul_x, ul_y - optional, upper left corner in pixels nrow, ncol - optional, size in pixels bands - optional, select band(s).
   The values returned are always a matrix, with the rows representing cells, and the columns representing band(s), c(nrow*ncol, length(bands)) (see Example #1)
$write_pixel_block(values, ul_x, ul_y, ncol, nrow)  Write pixel values. (see Example #2)
   ul_x, ul_y - optional, upper left corner in pixels nrow, ncol - optional, size in pixels
$has_colormap()  logical, return TRUE if raster has colormap
$attribute_table()  Query raster attribute table. Return data.frame object.
   Raster datasets that contain attribute tables typically have cell values that represent or define a class, group, category, or membership.
$save_as(path, opt)  TODO (see Example #3)
$commit(opt)  End writing. (see Example #2.3)
   opt - additional parameter(s): (default: "build-stats"),("build-pyramid")
arc.write  Write to an ArcGIS raster dataset

See Also

arc.raster, arc.write, arc.datasetraster-class
Examples

```r
## Example #1. 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)
dim(v) == c(25, 1)
#(1) TRUE TRUE

## Example #2. process big raster
## 2.1 create new arc.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)

## 2.2 loop by rows, process pixels
for (i in 1L:r$nrow)
{
  v <- r$pixel_block(ul_y = i - 1L, nrow = 1L)
  r2write_pixel_block(v * 1.5, ul_y = i - 1L, nrow = 1L, ncol = r$ncol)
}
## 2.3 stop all writings and create raster file
r2$commit()

## Example #3. 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"))

## Example #4. 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**  
```
## 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 arguments.

Value

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

Note

If object is arc.feature-class, the "shape" of class arc.shape-class will be attached to the resulting arc.data object.

See Also

arc.data, arc.open, arc.write

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" geometry object

Description

Get geometry object of arc.shape-class from arc.data object.

Usage

arc.shape(x)
Arguments

\texttt{x} \\
\texttt{a \texttt{data.frame} object of type \texttt{arc.data}}

Value

returns \texttt{arc.shape-class}

See Also

\texttt{arc.shapeinfo, arc.select, arc.data}

Examples

\begin{verbatim}
d <- \texttt{arc.open(system.file("extdata", "ca_ozone_pts.shp", package=\"arcgisbinding\"))}
df <- \texttt{arc.select(d, \"ozone\")}

shp <- \texttt{arc.shape(df)}
\texttt{stopifnot(length(shp) == nrow(df))}

shp
\texttt{## Not run:}
\texttt{geometry type : Point}
\texttt{WKID : PROJCS[\"USA_Contiguous_Alpers_Equal_Area_Conic\",GEOGCS[\"GCS_...}
\texttt{WKID : 102003}
\texttt{length : 193}
\texttt{## End(Not run)}
\end{verbatim}
Methods

\[ \text{signature(x = "arc.shape", i=numeric) select geometry subset} \]

\texttt{arc.shapeinfo} return geometry information

\texttt{length} length of collection

See Also

\texttt{arc.shape, arc.shapeinfo}

Examples

d <- \texttt{arc.select} (\texttt{arc.open} (\texttt{system.file} ("extdata", "ca_ozone_pts.shp", package="arcgisbinding"), "FID"))

\texttt{shape} <- \texttt{arc.shape} (d)
shape

## Not run:
geometry type : Point
\texttt{WKID} : PROJCS["USA Contiguous Albers Equal Area Conic",GEOGCS["GCS_...]
\texttt{WKID} : 102003
length : 193

## End(Not run)

# access X and Y values
\texttt{xy} <- \texttt{list} (X=shape$x, Y=shape$y)

---

\texttt{arc.shape2sf} \hspace{1cm} \textit{Convert 'arc.shape' geometry to 'sfc' simple feature geometry}

Description

Convert \texttt{arc.shape-class} to sfc simple feature geometry: POINT, MULTIPPOINT, POLYGON, MULTIPOLYGON, LINestring, MULTILINESTRING. Similar to \texttt{arc.data2sf}.

Usage

\texttt{arc.shape2sf} (\texttt{shape})

Arguments

\texttt{shape} \hspace{1cm} \texttt{arc.shape-class}

See Also

\texttt{arc.shape, arc.data2sf}
Examples

```r
if (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 `arc.shape` geometry to SP spatial geometry.

**Description**

Convert `arc.shape-class` to SP spatial geometry: SpatialPoints, SpatialLines, or SpatialPolygons. Similar to `arc.data2sp`.

**Usage**

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

**Arguments**

- `shape` *arc.shape-class*
- `wkt` optional, WKT spatial reference

**See Also**

`arc.shape`, `arc.data2sp`

**Examples**

```r
if (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**

Get geometry 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.
Usage

```r
## S4 method for signature 'arc.shape'
arc.shapeinfo(object)
## S4 method for signature 'arc.feature'
arc.shapeinfo(object)
```

Arguments

```r
object  arc.feature-class or arc.shape-class object
```

Value

returns named list of :

- **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.open, arc.shape

Examples

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

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

---

**arc.table-class**  
*Class "arc.table"

Description

arc.table S4 class

Details

The fields slot includes the details of the ArcGIS data types of the relevant fields, which include data types not directly representable in R.
**arc.write**

**Extends**

Class `arc.dataset-class`, directly.

```
arc.dataset-class
↓
arc.table-class
```

**Slots**

- `fields` named list of field types.

**Methods**

- `arc.select` return data.frame. TODO
- `names` return names of columns

**See Also**

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

**Examples**

```r
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.write**

*Write dataset, raster, feature, table or layer*

**Description**

Export a data 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. If data is raster-like object, this function writes a raster dataset. See ‘Details’ section for more information.

**Usage**

```r
arc.write(path, data, ..., overwrite = FALSE)
```
Arguments

path       full output path

Arguments

path       full output path

data       Accepts input source objects (see 'Details' for the types of objects allowed).

...       Optional parameters.

• coordslist containing geometry. Accepts Spatial objects. Put field names
  if data is data.frame and consists coordinates (see Example #2).
• shape_info required argument if data has no spatial attribute.

overwrite  overwrite existing dataset. default = FALSE.

Details

Export to a new table dataset when data type is:

• named list of vectors (see Example #4)
• data.frame

Export to a new feature dataset when data type is:

• arc.data result of arc.select
• named list of vectors, parameters coords and shape_info are required (see Example #5)
• data.frame, parameters coords and shape_info are required (see Example #2)
• SpatialPointsDataFrame in package sp
• SpatialLinesDataFrame in package sp
• SpatialPolygonsDataFrame in package sp
• sf, sfc in package sf

Export to a new raster dataset when data type is:

• arc.raster result of arc.raster
• SpatialPixels, SpatialPixelsDataFrame in package sp (see Example #6)
• SpatialGrid in package sp
• RasterLayer in package raster (see Example #7)
• RasterBrick in package raster

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?

Note

To write Date column type corresponding data column must have POSIXct type (see Example #4).

See Also

arc.open, arc.select, arc.raster

Examples

### Example #1. write 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)

```r
# create and write to a new file geodatabase
fgdb_path <- file.path(tempdir(), "data.gdb")
data(meuse, package="sp")
```

```r
# Example #2. create feature dataset 'meuse'
arc.write(file.path(fgdb_path, "meuse\pts"), data=meuse, coords=c("x", "y", "elev"), shape_info=list(type="point"))
riv <- sp::SpatialPolygons(list(sp::Polygons(list(sp::Polygon(meuse.riv)),"meuse.riv")))
```

```r
# Example #3. write only geometry
arc.write(file.path(fgdb_path, "meuse\riv"), coords=riv)
```

```r
# Example #4. write a table
t <- Sys.time() # now
arc.write(file.path(fgdb_path, "tlb"), data=list(
  'f_double'=c(23,45),
  'f_string'=c('hello', 'bob'),
  'f_datetime'=as.POSIXct(c(t, t - 3600)) # now and an hour ago
))
```

```r
# Example #5. 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'))
```

```r
# Example #6. write Raster
# make SpatialPixelsDataFrame
data(meuse.grid, package="sp")
sp::coordinates(meuse.grid) = c("x", "y")
sp::gridded(meuse.grid) <- TRUE
meuse.grid@proj4string <- CRS(arc.fromWktToP4(28992))
```
arc.write(file.path(fgdb_path, "meuse_grid"), meuse.grid)

## Example #7. 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)

---

**as.raster**

*Create RasterLayer or RasterBrick (raster package)*

### Description

Create Raster* object from arc.raster TODO

### Usage

```r
## S4 method for signature 'arc.raster'
as.raster(x, kind, ...)
```

### Arguments

- `x` : `arc.raster-class` object
- `kind` : internal parameter
- `...` : 

### Value

return RasterLayer for single band source or RasterBrick

### Examples

```r
## 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)
```

---

**pixel_type**

*Pixel types*

### Description

Bit depth capacity for raster dataset cells.
The following table shows the **pixel_type** value and the range of values stored for different bit depths:
raster_compression_type

Pixel type | Bit depth | Range of values that each cell can contain
---|---|---
“U1” | 1 bit | 0 to 1
“U2” | 2 bits | 0 to 3
“U4” | 4 bits | 0 to 15
“U8” | Unsigned 8 bit integers | 0 to 255
“S8” | 8 bit integers | -128 to 128
“U16” | Unsigned 16 bit integers | 0 to 65535
“S16” | 16 bit integers | -32768 to 32767
“U32” | Unsigned 32 bit integers | 0 to 4294967295
“S32” | 32 bit integers | -2147483648 to 2147483647
“F32” | 32 bit Single precision floating point | -3.402823466e+38 to 3.402823466e+38
“F64” | 64 bit Double precision floating point | 0 to 18446744073709551616

References

1. ArcGIS Help: Pixel Types

See Also

resample_type, arc.raster, arc.raster-class

Description

Raster compression types

The following table shows the compression_type value:

<table>
<thead>
<tr>
<th>Compression type</th>
<th>Lossy or lossless</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;LZ77&quot;</td>
<td>Lossless</td>
<td>Can define a compression quality</td>
</tr>
<tr>
<td>&quot;JPEG&quot;</td>
<td>Lossy</td>
<td>Can define a compression quality</td>
</tr>
<tr>
<td>&quot;JPEG 2000&quot;</td>
<td>Lossy or lossless</td>
<td>Can define a compression quality</td>
</tr>
<tr>
<td>&quot;PackBits&quot;</td>
<td>Lossless</td>
<td>Applies to TIFF only</td>
</tr>
<tr>
<td>&quot;LZW&quot;</td>
<td>Lossless</td>
<td></td>
</tr>
<tr>
<td>&quot;RLE&quot;</td>
<td>Lossless</td>
<td>Applies to TIFF only</td>
</tr>
<tr>
<td>&quot;CCITT GROUP 3&quot;</td>
<td>Lossless</td>
<td>Applies to TIFF only</td>
</tr>
<tr>
<td>&quot;CCITT GROUP 4&quot;</td>
<td>Lossless</td>
<td>Applies to TIFF only</td>
</tr>
<tr>
<td>&quot;CCITT (1D)&quot;</td>
<td>Lossless</td>
<td>Applies to TIFF only</td>
</tr>
<tr>
<td>&quot;None&quot;</td>
<td>No data compression</td>
<td></td>
</tr>
</tbody>
</table>

See Also

arc.raster, resample_type, pixel_type, arc.datasetraster-class
**resample_type**

<table>
<thead>
<tr>
<th><strong>resample_type</strong></th>
<th><strong>Resample types</strong></th>
</tr>
</thead>
</table>

## Description

Resample types

## Supported

<table>
<thead>
<tr>
<th><strong>Resample type</strong></th>
<th><strong>Definition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;NearestNeighbor&quot;</td>
<td>- Performs a nearest neighbor assignment and is the fastest of the interpolation methods. This is the default.</td>
</tr>
<tr>
<td>&quot;BilinearInterpolation&quot;</td>
<td>- Performs a bilinear interpolation and determines the new value of a cell based on a weighted distance average of the four nearest input cell centers.</td>
</tr>
<tr>
<td>&quot;CubicConvolution&quot;</td>
<td>- Performs a cubic convolution and determines the new value of a cell based on fitting a smooth curve through the 16 nearest input cell centers.</td>
</tr>
<tr>
<td>&quot;Majority&quot;</td>
<td>- Performs a majority algorithm and determines the new value of the cell based on the most popular values within the filter window.</td>
</tr>
<tr>
<td>&quot;BilinearInterpolationPlus&quot;</td>
<td>- Performs a bilinear interpolation and determines the new value of a cell based on a weighted distance average of the four nearest input cell centers.</td>
</tr>
<tr>
<td>&quot;BilinearGaussBlur&quot;</td>
<td>- Performs a bilinear interpolation and determines the new value of a cell based on a Gaussian blur.</td>
</tr>
<tr>
<td>&quot;BilinearGaussBlurPlus&quot;</td>
<td>- Performs a bilinear interpolation and determines the new value of a cell based on a Gaussian blur plus a weighted distance average of the four nearest input cell centers.</td>
</tr>
<tr>
<td>&quot;Average&quot;</td>
<td>- Performs an average of the values within the filter window.</td>
</tr>
<tr>
<td>&quot;Minimum&quot;</td>
<td>- Performs a minimum of the values within the filter window.</td>
</tr>
<tr>
<td>&quot;VectorAverage&quot;</td>
<td>- Performs a vector average of the values within the filter window.</td>
</tr>
<tr>
<td>TODO</td>
<td>- Performs a majority algorithm and determines the new value of the cell based on the most popular values within the filter window.</td>
</tr>
</tbody>
</table>

## References

1. [Cell size and resampling in analysis](#)

## Note

The Bilinear and Cubic options should not be used with categorical data, since the cell values may be altered.

## See Also

`pixel_type arc.raster arc.raster-class`
Index

*Topic SpatialReference
  arc.shapelineinfo, 25

*Topic classes
  arc.data, 4
  arc.dataset-class, 7
  arc.datasetraster-class, 8
  arc.datasetrastermosaic-class, 9
  arc.feature-class, 11
  arc.raster-class, 19
  arc.shape-class, 23
  arc.table-class, 26

*Topic datasets
  arc.select, 21
  arc.write, 27

*Topic dataset
  arc.data, 4
  arc.dataset-class, 7
  arc.datasetraster-class, 8
  arc.datasetrastermosaic-class, 9
  arc.feature-class, 11
  arc.shape-class, 23
  arc.table-class, 26

*Topic delete
  arc.delete, 10

*Topic features
  arc.data, 4

*Topic feature
  arc.data2sf, 5
  arc.data2sp, 6
  arc.feature-class, 11
  arc.select, 21
  arc.shape2sf, 24
  arc.shape2sp, 25
  arc.write, 27

*Topic geometry
  arc.data2sf, 5
  arc.data2sp, 6
  arc.shape, 22
  arc.shape-class, 23
  arc.shape2sf, 24
  arc.shape2sp, 25
  arc.shapeinfo, 25

*Topic methods
  arc.data, 4
  arcgisbinding-package, 2

*Topic method
  arc.data2sf, 5
  arc.data2sp, 6
  arc.delete, 10
  arc.env, 10
  arc.fromP4ToWkt, 12
  arc.fromWktToP4, 13
  arc.open, 14
  arc.progress_label, 16
  arc.progress_pos, 17
  arc.raster, 18
  arc.select, 21
  arc.shape, 22
  arc.shape2sf, 24
  arc.shape2sp, 25
  arc.write, 27
  as.raster, 30

*Topic mosaic
  arc.datasetrastermosaic-class, 9

*Topic open
  arc.datasetraster-class, 8
  arc.open, 14
  arc.write, 27

*Topic package
  arcgisbinding-package, 2

*Topic projection
  arc.fromP4ToWkt, 12
  arc.fromWktToP4, 13

*Topic rasterdataset
  arc.datasetraster-class, 8

*Topic raster
  arc.datasetraster-class, 8
  arc.datasetrastermosaic-class, 9
  arc.raster, 18
  arc.raster-class, 19
  arc.write, 27
  as.raster, 30

*Topic select
  arc.select, 21

*Topic shape
  arc.shape-class, 23
  arc.shapeinfo, 25
INDEX

SpatialPointsDataFrame, 28
SpatialPolygonsDataFrame, 28

ungroup.arc.data (arc.data), 4