

# Package ‘arctgisbinding’

June 26, 2023

**Version** 1.0.1.306

**Date** 2023-06-23

**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 (>= 4.0.2)

**Imports** methods

**Suggests** sp,

sf,

raster,

rgdal,

dplyr,

repr,

htmltools,

leaflet,

bit64

**License** file LICENSE

**URL** <http://esri.com>

**BugReports** <https://github.com/R-ArcGIS/r-bridge/issues>

**OS\_type** windows

**Encoding** UTF-8

**Archs** x64

## R topics documented:

arctgisbinding-package . . . . .	2
arc.check_product . . . . .	3
arc.data . . . . .	4
arc.dataset-class . . . . .	5
arc.datasetraster-class . . . . .	6
arc.datasetrastermosaic-class . . . . .	7

arc.delete . . . . .	8
arc.env . . . . .	9
arc.feature-class . . . . .	10
arc.fromP4ToWkt . . . . .	11
arc.fromWktToP4 . . . . .	12
arc.open . . . . .	13
arc.raster . . . . .	14
arc.raster-class . . . . .	16
arc.select . . . . .	18
arc.shape . . . . .	19
arc.shape-class . . . . .	20
arc.shapeinfo . . . . .	21
arc.table-class . . . . .	22
arc.write . . . . .	23
as.raster . . . . .	25
Convert to (sp) SpatialDataFrame, (sf) Simple Feature . . . . .	26
Convert to sp::Spatial* - Spatial geometry, sf::sfc - Simple Feature geometry . . . . .	27
Enterprise and Online portals . . . . .	28
resample, pixel, compression types . . . . .	29
Working with progressor . . . . .	30
<b>Index</b>	<b>32</b>

---

arcgisbinding-package *Bindings for ArcGIS*

---

## 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](#)

---

arc.check_product	<i>ArcGIS product and license information</i>
-------------------	---

---

## 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

```
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

```
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.data`*Class "arc.data"*

---

## Description

`arc.data` class and methods

## Usage

```
## 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

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

## 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="arctgisbinding")))
d
## Not run:
geometry type   : Point
WKT             : PROJCS["USA_Contiguous_Albers_Equal_Area_Conic",GEOGCS["GCS_...
WKID            : 102003
FID LATITUDE LONGITUDE ELEVATION OZONE X Y text
1 0 39.1447 -123.2065 194 0.04650 -2298092 515557.4 Value_0
2 1 39.4030 -123.3491 420 0.04969 -2301588 546772.7 Value_1
3 2 37.7661 -122.3978 5 0.05000 -2273948 347691.4 Value_2
4 3 37.9508 -122.3569 23 0.05799 -2264847 366623.2 Value_3
5 4 36.6986 -121.6354 36 0.05860 -2241776 214412.1 Value_0
... ... ... ... ...
191 190 34.0598 -117.1462 0 0.16449 -1921585 -170440.0 Value_2
192 191 34.2412 -117.2756 1384 0.16470 -1928645 -148045.5 Value_3
193 192 34.1065 -117.2732 0 0.17360 -1931774 -162775.2 Value_0

## 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.dataset-class	<i>Class "arc.dataset"</i>
-------------------	----------------------------

---

### 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.dataseptraster-class](#), [arc.dataseptrastermosaic-class](#)

## Examples

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

---

```
arc.dataseptraster-class
```

```
Class "arc.dataseptraster"
```

---

## Description

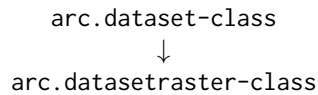
arc.dataseptraster S4 class. Dataset class for raster objects. Creates a dataset object with type = raster.

## 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.

**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**

**[arc.raster](#)** Create a `arc.raster` object

**`dim`** retrieves dimensions of a `arc.dataset` object

**`names`** return bands names

**[arc.write](#)** TODO

**References**

1. [ArcGIS Help: Raster dataset properties](#)

**See Also**

[arc.raster](#), [arc.write](#)

---

```
arc.dataseptrastermosaic-class
```

```
Class "arc.dataseptrastermosaic"
```

---

**Description**

`arc.dataseptrastermosaic` S4 class. Dataset class for mosaic objects.

Details

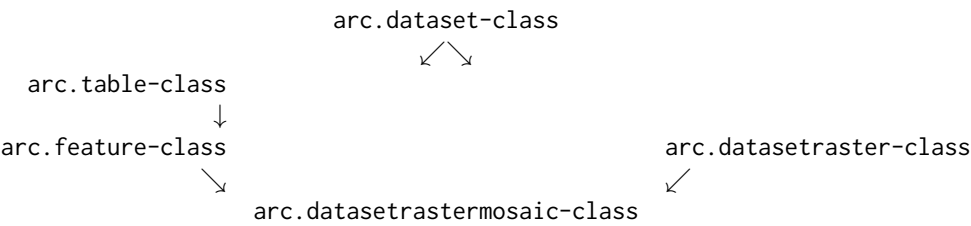
Mosaic datasets are made up of a collection of rasters. Mosaic structure efficiently stores and manages multiple rasters for visualization and analysis. Detailed information about mosaic datasets can be found in [ArcGIS reference for mosaic datasets](#).

[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](#)

---

arc.delete	Delete dataset
------------	----------------

---

Description

delete dataset

Usage

```
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**

```
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))
```

---

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 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.

**Value**

return enviroment 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**

```
## Not run:
tool_exec <- function(in_para, 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.dataseatraster-class](#), [arc.dataseatrastermosaic-class](#)

**Examples**

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

---

arc.fromP4ToWkt	<i>Convert PROJ.4 Coordinate Reference System string to Well-known Text.</i>
-----------------	--

---

**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**

```
arc.fromP4ToWkt(proj4)
```

**Arguments**

proj4	PROJ.4 projection string
-------	--------------------------

**Details**

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

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

**Value**

return WKT string

**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") # initialize based on EPSG code
```

---

arc.fromWktToP4	<i>Convert a Well-known Text Coordinate Reference System into a PROJ.4 string.</i>
-----------------	--

---

## 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.

## Usage

```
arc.fromWktToP4(wkt)
```

## 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.fromP4ToWkt](#)

## Examples

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

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

---

arc.open	<i>Open dataset, table, or layer</i>
----------	--------------------------------------

---

## 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 these rasters are also known as gridded 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

```
## 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.raster

*Load or create "arc.raster" object*

---

## Description

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

## Usage

```
## 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	code <a href="#">arc.datasetraster-class</a> 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. Use overwrite=TRUE to overwrite existing dataset.
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.
cellsize_x	optional. Size of pixel in x-axis.
cellsize_y	optional. Size of pixel in y-axis.
pixel_type	optional. Type of raster pixels. For details about different pixel types see <a href="#">pixel_type</a> . See also <a href="#">ArcGIS Help: Pixel Types</a> . 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.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.dataseatraster-class](#)

## Examples

```
## 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

stopifnot(length(v) == 25)

## 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)
  r2$write_pixel_block(v * 1.5, ul_y = i - 1L, nrow = 1L, ncol = r$ncol)
}
## 2.3 stop all writings and crete 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	<i>Load dataset to "data.frame"</i>
------------	-------------------------------------

---

## 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	<a href="#">arc.dataset-class</a> 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)
...	optional additional arguments such as transformation - datum transformation string

## 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="arctools")
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)

## datum transformation, from NAD_1983 to WGS_1984
## prepare dataset
x <- c(1455535.149968639, 1446183.62438038, 1447950.6349539757)
y <- c(478067.64943164587, 484500.4190463871, 479746.6336064786)
data_path <- file.path(tempdir(), "data.gdb", "test_datum")
## save as NAD_1983
arc.write(data_path, coords=cbind(x, y), shape_info=list(type="Point", WKID=2893))
## read and transform to WGS_1984
df <- arc.select(arc.open(data_path), sr=4326, transformation='NAD_1983_HARN_To_WGS_1984_2')
x <- arc.shape(df)$x
stopifnot(sprintf('%.8f', x[1]) == '-76.49626388')
```

---

`arc.shape`

*Get "arc.shape" geometry object*

---

**Description**

Get geometry object of [arc.shape-class](#) from [arc.data](#) object.

**Usage**

```
arc.shape(x)
```

**Arguments**

`x` a `data.frame` object of type `arc.data`

Value

returns [arc.shape-class](#)

See Also

[arc.shapeinfo](#), [arc.select](#), [arc.data](#)

Examples

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

shp <- arc.shape(df)
stopifnot(length(shp) == nrow(df))

shp
## Not run:
geometry type      : Point
WKT                 : PROJCS["USA_Contiguous_Albers_Equal_Area_Conic",GEOGCS["GCS_...
WKID                : 102003
length             : 193

## End(Not run)
```

---

arc.shape-class	<i>Class "arc.shape"</i>
-----------------	--------------------------

---

Description

arc.shape S4 class. Object arc.shape is a geometry collection.

Details

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

Extends

Class list, directly.

Slots

.Data internal  
shapeinfo geometry information, for mode details see [arc.shapeinfo](#)

Methods

[ signature(x = "arc.shape", i=numeric) select geometry subset  
[arc.shapeinfo](#) return geometry information  
**length** length of collection

**See Also**

[arc.shape](#), [arc.shapeinfo](#)

**Examples**

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

shape <- arc.shape(d)
shape
## Not run:
geometry type   : Point
WKT             : PROJCS["USA_Contiguous_Albers_Equal_Area_Conic",GEOGCS["GCS_...
WKID            : 102003
length         : 193

## End(Not run)

# access X and Y values
xy <- list(X=shape$x, Y=shape$y)
```

---

arc.shapeinfo	<i>Get geometry information</i>
---------------	---------------------------------

---

**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**

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

**Arguments**

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

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

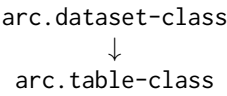
arc.table S4 class

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.

Extends

Class arc.dataset-class, directly.



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

```
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	<i>Write dataset, raster, feature, table or layer</i>
-----------	---

---

## 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

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

## Arguments

path	full output path
data	Accepts input source objects (see ‘Details’ for the types of objects allowed).
...	Additional arguments: <ul style="list-style-type: none"> <li>• coords list containing geometry. Accepts Spatial objects. If data is data.frame coords can be list of field names (see <i>Example #2</i>).</li> <li>• shape_info list. Required argument if data has no spatial attribute (see <i>Example #2</i>).</li> <li>• validate logical. Default FALSE. If TRUE makes the geometries topologically correct.</li> </ul>
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="arctgisbinding"))
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")
## 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'))
data(meuse.riv, package="sp")
riv <- sp::SpatialPolygons(list(sp::Polygons(list(sp::Polygon(meuse.riv)), "meuse.riv")))

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

## 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
))

## 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'))

## 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=sp::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

```

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

```

**Arguments**

x	<a href="#">arc.raster-class</a> object
kind	internal parameter
...	.

**Value**

return RasterLayer for single band source 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)
```

---

Convert to (sp) SpatialDataFrame, (sf) Simple Feature

*Convert 'arc.data' or 'arc.raster' object to 'sp' - SpatialDataFrame object or 'sf' - Simple Feature 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.

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

**Usage**

```
arc.data2sp(x, ...)
arc.data2sf(x, ...)
```

**Arguments**

x	<a href="#">arc.data</a> object, result of <a href="#">arc.select</a> or <a href="#">arc.raster</a> .
...	optional additional argument such wkt WKT spatial reference or crs coordinate reference string to assign to return object

**Value**

`sp::Spatial*DataFrame` object.

`sf::sfc` object.

**See Also**

[arc.open](#), [arc.select](#) [arc.raster](#)

## Examples

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

require("sp")
df.sp <- arc.data2sp(d)
## Not run: spplot(df.sp)
```

```
require("sf")
df.sf <- arc.data2sf(d)
## Not run: plot(df.sf)
```

---

Convert to `sp::Spatial*` - Spatial geometry, `sf::sfc` - Simple Feature geometry  
 Convert 'arc.shape' geometry object to `sp::Spatial*` - Spatial geometry or `sf::sfc` - simple feature geometry

---

## Description

Convert [arc.shape-class](#) to sp spatial geometry: `SpatialPoints`, `SpatialLines`, or `SpatialPolygons`. Similar to [arc.data2sp](#).

Convert [arc.shape-class](#) to `sfc` simple feature geometry: `POINT`, `MULTIPOINT`, `POLYGON`, `MULTIPOLYGON`, `LINestring`, `MULTILINESTRING`. Similar to [arc.data2sf](#).

## Usage

```
arc.shape2sp(shape, ...)
arc.shape2sf(shape, ...)
```

## Arguments

<code>shape</code>	<a href="#">arc.shape-class</a>
<code>...</code>	optional wkt WKT spatial reference or crs spatial reference string to assign to return object

## Value

an object of class `sp::Spatial*`.

an object of class `sf::sfc`, which is a classed list-column with simple feature geometries.

## See Also

[arc.shape](#), [arc.data2sp](#) [arc.data2sf](#)

**Examples**

```
d <- arc.select(arc.open(system.file("extdata", "ca_ozone_pts.shp", package="arctgisbinding")), 'ozone')
x <- arc.shape(d)
```

```
geom <- arc.shape2sp(x)
## Not run: plot(geom)
```

```
geom <- arc.shape2sf(x)
## Not run: plot(geom)
```

---

Enterprise and Online portals

*ArcGIS Enterprise and Online portals*

---

**Description**

The `arc.portal_connect()` function to sign in to a portal. To check available portals call `arc.check_portal()`. Functions returns a list that contains active info and available portals.

**Usage**

```
arc.portal_connect(url, user, password)
arc.check_portal()
```

**Arguments**

<code>url</code>	The URL of the portal to be signed in to. (character)
<code>user</code>	The user name of the user signing in to the portal. (character)
<code>password</code>	The password of the user signing in to the portal. (character)

**Details**

If `url` already in active list of portals connections then `user` and `password` parameters are optional

**Value**

An named list of portal connections.

- `url` - The URL of the current portal.
- `user` - The user name.
- `version` - The portal version.
- `organization` - The organization name.
- `session` - TODO.
- `token` - This is the Enterprise token for built-in logins.
- `portals` - list of active portals.
- `offlines` - list of offline portals.

---

resample, pixel, compression types

*Raster resample, pixel, compression types*


---

## Description

The following table shows the pixel\_type value and the range of values stored for different bit depths:

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

The following table shows the resamp\_type value:

Resample type	Definition
"NearestNeighbor"	- Performs a nearest neighbor assignment and is the fastest of the interpolation methods
"BilinearInterpolation"	- Performs a bilinear interpolation and determines the new value of a cell based on a v
"CubicConvolution"	- Performs a cubic convolution and determines the new value of a cell based on fitting
"Majority"	- Performs a majority algorithm and determines the new value of the cell based on the
"BilinearInterpolationPlus"	TODO
"BilinearGaussBlur"	TODO
"BilinearGaussBlurPlus"	TODO
"Average"	TODO
"Minimum"	TODO
"Average"	TODO
"VectorAverage"	TODO

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

The following table shows the compression\_type value:

Compression type	Lossy or lossless	Notes
"LZ77"	Lossless	
"JPEG"	Lossy	Can define a compression quality
"JPEG 2000"	Lossy or lossless	Can define a compression quality
"PackBits"	Lossless	Applies to TIFF only
"LZW"	Lossless	
"RLE"	Lossless	

"CCITT GROUP 3"	Lossless	Applies to TIFF only
"CCITT GROUP 4"	Lossless	Applies to TIFF only
"CCITT (1D)"	Lossless	Applies to TIFF only
"None"	No data compression	

## References

1. [ArcGIS Help: Pixel Types](#)

## See Also

[arc.raster](#), [arc.raster-class](#)

---

Working with progressor

*Progressor for ArcGIS Geoprocessing dialog*

---

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

## Arguments

label	Progress Label
pos	Progress position (in percent)

## 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.

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\\_pos](#), "Progress Messages" example Geoprocessing script

### Examples

```
## Not run:  
arc.progress_label("Calculating bootstrap samples...")  
arc.progress_pos(55)  
  
## End(Not run)
```

# Index

- \* **SpatialReference**
  - arc.shapeinfo, [21](#)
- \* **classes**
  - arc.data, [4](#)
  - arc.dataset-class, [5](#)
  - arc.datasetraster-class, [6](#)
  - arc.datasetrastermosaic-class, [7](#)
  - arc.feature-class, [10](#)
  - arc.raster-class, [16](#)
  - arc.shape-class, [20](#)
  - arc.table-class, [22](#)
- \* **datasets**
  - arc.select, [18](#)
  - arc.write, [23](#)
- \* **dataset**
  - arc.data, [4](#)
  - arc.dataset-class, [5](#)
  - arc.datasetraster-class, [6](#)
  - arc.datasetrastermosaic-class, [7](#)
  - arc.feature-class, [10](#)
  - arc.table-class, [22](#)
- \* **delete**
  - arc.delete, [8](#)
- \* **features**
  - arc.data, [4](#)
- \* **feature**
  - arc.feature-class, [10](#)
  - arc.open, [13](#)
  - arc.select, [18](#)
  - arc.write, [23](#)
  - Convert to (sp) SpatialDataFrame, (sf) Simple Feature, [26](#)
  - Convert to sp::Spatial\* - Spatial geometry, sf::sfc - Simple Feature geometry, [27](#)
- \* **geomery**
  - arc.shapeinfo, [21](#)
- \* **geometry**
  - arc.shape, [19](#)
  - arc.shape-class, [20](#)
  - arc.shapeinfo, [21](#)
  - Convert to (sp) SpatialDataFrame, (sf) Simple Feature, [26](#)
- Convert to sp::Spatial\* - Spatial geometry, sf::sfc - Simple Feature geometry, [27](#)
- \* **layer**
  - arc.open, [13](#)
- \* **methods**
  - arc.data, [4](#)
  - arcgisbinding-package, [2](#)
- \* **method**
  - arc.delete, [8](#)
  - arc.env, [9](#)
  - arc.fromP4ToWkt, [11](#)
  - arc.fromWktToP4, [12](#)
  - arc.open, [13](#)
  - arc.raster, [14](#)
  - arc.select, [18](#)
  - arc.shape, [19](#)
  - arc.write, [23](#)
  - as.raster, [25](#)
  - Convert to (sp) SpatialDataFrame, (sf) Simple Feature, [26](#)
  - Convert to sp::Spatial\* - Spatial geometry, sf::sfc - Simple Feature geometry, [27](#)
  - Enterprise and Online portals, [28](#)
  - Working with progressor, [30](#)
- \* **mosaic**
  - arc.datasetrastermosaic-class, [7](#)
- \* **open dataset**
  - arc.open, [13](#)
- \* **open feature**
  - arc.open, [13](#)
- \* **open layer**
  - arc.open, [13](#)
- \* **open raster**
  - arc.open, [13](#)
- \* **open shape**
  - arc.open, [13](#)
- \* **open table**
  - arc.open, [13](#)
- \* **open**
  - arc.datasetraster-class, [6](#)
  - arc.open, [13](#)



- arc.write, 23
- \* **package**
  - arcgisbinding-package, 2
- \* **portal**
  - Enterprise and Online portals, 28
- \* **projection**
  - arc.fromP4ToWkt, 11
  - arc.fromWktToP4, 12
- \* **rasterdataset**
  - arc.dataseptraster-class, 6
- \* **raster**
  - arc.dataseptraster-class, 6
  - arc.dataseptrastermosaic-class, 7
  - arc.open, 13
  - arc.raster, 14
  - arc.raster-class, 16
  - arc.write, 23
  - as.raster, 25
- \* **select**
  - arc.select, 18
- \* **shape**
  - arc.open, 13
  - arc.select, 18
  - arc.shape-class, 20
  - arc.shapeinfo, 21
  - Convert to sp::Spatial\* - Spatial geometry, sf::sfc - Simple Feature geometry, 27
- \* **spatial**
  - arc.raster-class, 16
  - arc.select, 18
  - Convert to (sp) SpatialDataFrame, (sf) Simple Feature, 26
  - Convert to sp::Spatial\* - Spatial geometry, sf::sfc - Simple Feature geometry, 27
- \* **subset**
  - arc.data, 4
- \* **table**
  - arc.open, 13
  - arc.select, 18
- \* **write**
  - arc.write, 23
- [, arc.shape (arc.shape-class), 20
- [.arc.data (arc.data), 4
- arc.check\_portal (Enterprise and Online portals), 28
- arc.check\_product, 3
- arc.data, 4, 19, 20, 26
- arc.data2sf, 27
- arc.data2sf (Convert to (sp) SpatialDataFrame, (sf) Simple Feature), 26
- arc.data2sp, 27
- arc.data2sp (Convert to (sp) SpatialDataFrame, (sf) Simple Feature), 26
- arc.dataset-class, 5, 18
- arc.dataseptraster-class, 6, 15
- arc.dataseptrastermosaic-class, 7
- arc.delete, 6, 8
- arc.delete, arc.dataset-method (arc.delete), 8
- arc.env, 9
- arc.feature-class, 10, 21
- arc.fromP4ToWkt, 11, 12
- arc.fromWktToP4, 12, 12
- arc.metadata, 6
- arc.open, 5, 6, 8, 11, 13, 16, 19, 22, 24, 26
- arc.portal\_connect (Enterprise and Online portals), 28
- arc.progress\_label, 30
- arc.progress\_label (Working with progressor), 30
- arc.progress\_pos, 30, 31
- arc.progress\_pos (Working with progressor), 30
- arc.raster, 7, 8, 14, 14, 17, 24, 26, 30
- arc.raster, arc.dataseptraster-method (arc.raster), 14
- arc.raster, arc.dataseptrastermosaic-method (arc.raster), 14
- arc.raster, NULL-method (arc.raster), 14
- arc.raster-class, 15, 16, 26
- arc.select, 5, 8, 10, 14, 18, 20, 22–24, 26
- arc.select, arc.table-method (arc.select), 18
- arc.shape, 4, 5, 19, 21, 22, 27
- arc.shape-class, 20, 21
- arc.shape2sf (Convert to sp::Spatial\* - Spatial geometry, sf::sfc - Simple Feature geometry), 27
- arc.shape2sp (Convert to sp::Spatial\* - Spatial geometry, sf::sfc - Simple Feature geometry), 27
- arc.shapeinfo, 10, 20, 21, 21
- arc.shapeinfo, arc.feature-method (arc.shapeinfo), 21
- arc.shapeinfo, arc.shape-method (arc.shapeinfo), 21
- arc.shapeinfo.arc.data (arc.shapeinfo), 21
- arc.table-class, 22
- arc.write, 7, 14, 16, 17, 19, 23

- arcgisbinding (arcgisbinding-package), 2
- arcgisbinding-package, 2
- arrange.arc.data (arc.data), 4
- as.raster, 25
- as.raster, arc.raster-method  
(as.raster), 25
- compression type, 7
- compression\_type (resample, pixel,  
compression types), 29
- Convert to (sp) SpatialDataFrame, (sf)  
Simple Feature, 26
- Convert to sp::Spatial\* - Spatial  
geometry, sf::sfc - Simple  
Feature geometry, 27
- dim, arc.dataseatraster-method  
(arc.dataseatraster-class), 6
- dim, arc.raster-method  
(arc.raster-class), 16
- dim<-, arc.raster-method  
(arc.raster-class), 16
- Enterprise and Online portals, 28
- filter.arc.data (arc.data), 4
- group\_by.arc.data (arc.data), 4
- length, arc.shape-method  
(arc.shape-class), 20
- mutate.arc.data (arc.data), 4
- names, arc.dataseatraster-method  
(arc.dataseatraster-class), 6
- names, arc.feature-method  
(arc.feature-class), 10
- names, arc.raster-method  
(arc.raster-class), 16
- names, arc.table-method  
(arc.table-class), 22
- pixel type, 7, 17
- pixel\_type, 15
- pixel\_type (resample, pixel,  
compression types), 29
- POSIXct, 24
- RasterBrick, 24
- RasterLayer, 24
- resample, pixel, compression types, 29
- resample\_type (resample, pixel,  
compression types), 29
- resampling type, 17
- sf, 24
- sfc, 24
- SpatialGrid, 24
- SpatialLinesDataFrame, 24
- SpatialPixels, 24
- SpatialPixelsDataFrame, 24
- SpatialPointsDataFrame, 24
- SpatialPolygonsDataFrame, 24
- ungroup.arc.data (arc.data), 4
- Working with progressor, 30