Package 'foreign'

January 13, 2022

Priority recommended

Version 0.8-82

Date 2022-01-13

Title Read Data Stored by 'Minitab', 'S', 'SAS', 'SPSS', 'Stata', 'Systat', 'Weka', 'dBase', ...

Depends R (>= 4.0.0)

Imports methods, utils, stats

Contact see 'MailingList'

Copyright see file COPYRIGHTS

Description Reading and writing data stored by some versions of 'Epi Info', 'Minitab', 'S', 'SAS', 'SPSS', 'Stata', 'Systat', 'Weka', and for reading and writing some 'dBase' files.

ByteCompile yes

Biarch yes

License GPL (>= 2)

BugReports https://bugs.r-project.org

MailingList R-help@r-project.org

URL https://svn.r-project.org/R-packages/trunk/foreign/

NeedsCompilation yes

Author R Core Team [aut, cph, cre],
Roger Bivand [ctb, cph],
Vincent J. Carey [ctb, cph],
Saikat DebRoy [ctb, cph],
Stephen Eglen [ctb, cph],
Rajarshi Guha [ctb, cph],
Swetlana Herbrandt [ctb],
Nicholas Lewin-Koh [ctb, cph],
Mark Myatt [ctb, cph],
Michael Nelson [ctb],
Ben Pfaff [ctb],

Brian Quistorff [ctb],
Frank Warmerdam [ctb, cph],
Stephen Weigand [ctb, cph],
Free Software Foundation, Inc. [cph]

Maintainer R Core Team <R-core@R-project.org>

Repository CRAN

Date/Publication 2022-01-13 12:09:16 UTC

R topics documented:

`lookup.xport` *Lookup Information on a SAS XPORT Format Library*

Description

Scans a file as a SAS XPORT format library and returns a list containing information about the SAS library.

Usage

```
lookup.xport(file)
```

Arguments

`file`	character variable with the name of the file to read. The file must be in SAS XPORT format.

Value

A list with one component for each dataset in the XPORT format library.

Author(s)

Saikat DebRoy

References

SAS Technical Support document TS-140: "The Record Layout of a Data Set in SAS Transport (XPORT) Format" available as https://support.sas.com/content/dam/SAS/support/en/technical-papers/record-layout-of-a-sas-version-5-or-6-data-set-in-sas-transport-xport-format.pdf

See Also

read.xport

Examples

```
## Not run: ## no XPORT file is installed.
lookup.xport("test.xpt")

## End(Not run)
```

---

read.arff *Read Data from ARFF Files*

Description

Reads data from Weka Attribute-Relation File Format (ARFF) files.

Usage

```
read.arff(file)
```

Arguments

file	a character string with the name of the ARFF file to read from, or a connection which will be opened if necessary, and if so closed at the end of the function call.

Value

A data frame containing the data from the ARFF file.

References

Attribute-Relation File Format https://waikato.github.io/weka-wiki/formats_and_processing/arff/.

See Also

write.arff; functions write.arff and read.arff in package **RWeka** which provide some support for logicals via conversion to or from factors.

---

read.dbf *Read a DBF File*

Description

The function reads a DBF file into a data frame, converting character fields to factors, and trying to respect NULL fields.

The DBF format is documented but not much adhered to. There is is no guarantee this will read all DBF files.

Usage

```
read.dbf(file, as.is = FALSE)
```

Arguments

file	name of input file
as.is	should character vectors not be converted to factors?

Details

DBF is the extension used for files written for the 'XBASE' family of database languages, 'covering the dBase, Clipper, FoxPro, and their Windows equivalents Visual dBase, Visual Objects, and Visual FoxPro, plus some older products' (https://www.clicketyclick.dk/databases/xbase/format/). Most of these follow the file structure used by Ashton-Tate's dBase II, III or 4 (later owned by Borland).

read.dbf is based on C code from http://shapelib.maptools.org/ which implements the 'XBASE' specification. It can convert fields of type "L" (logical), "N" and "F" (numeric and float) and "D" (dates): all other field types are read as-is as character vectors. A numeric field is read as an R integer vector if it is encoded to have no decimals, otherwise as a numeric vector. However, if the numbers are too large to fit into an integer vector, it is changed to numeric. Note that is possible to read integers that cannot be represented exactly even as doubles: this sometimes occurs if IDs are incorrectly coded as numeric.

Value

A data frame of data from the DBF file; note that the field names are adjusted to use in R using make.names(unique=TRUE).

There is an attribute "data_type" giving the single-character dBase types for each field.

Note

Not to be able to read a particular 'DBF' file is not a bug: this is a convenience function especially for shapefiles.

Author(s)

Nicholas Lewin-Koh and Roger Bivand; shapelib by Frank Warmerdam

References

http://shapelib.maptools.org/.

See Also

write.dbf

Examples

```
x <- read.dbf(system.file("files/sids.dbf", package="foreign")[1])
str(x)
summary(x)
```

`read.dta` *Read Stata Binary Files*

Description

Reads a file in Stata version 5–12 binary format into a data frame.

Frozen: will not support Stata formats after 12.

Usage

```
read.dta(file, convert.dates = TRUE, convert.factors = TRUE,
         missing.type = FALSE,
         convert.underscore = FALSE, warn.missing.labels = TRUE)
```

Arguments

`file`	a filename or URL as a character string.
`convert.dates`	Convert Stata dates to `Date` class, and date-times to `POSIXct` class?
`convert.factors`	Use Stata value labels to create factors? (Version 6.0 or later).
`missing.type`	For version 8 or later, store information about different types of missing data?
`convert.underscore`	Convert `"_"` in Stata variable names to `"."` in R names?
`warn.missing.labels`	Warn if a variable is specified with value labels and those value labels are not present in the file.

Details

If the filename appears to be a URL (of schemes 'http:', 'ftp:' or 'https:') the URL is first downloaded to a temporary file and then read. ('https:' is only supported on some platforms.)

The variables in the Stata data set become the columns of the data frame. Missing values are correctly handled. The data label, variable labels, timestamp, and variable/dataset characteristics are stored as attributes of the data frame.

By default Stata dates (%d and %td formats) are converted to R's `Date` class, and variables with Stata value labels are converted to factors. Ordinarily, `read.dta` will not convert a variable to a factor unless a label is present for every level. Use `convert.factors = NA` to override this. In any case the value label and format information is stored as attributes on the returned data frame. Stata's date formats are sketchily documented: if necessary use `convert.dates = FALSE` and examine the attributes to work out how to post-process the dates.

Stata 8 introduced a system of 27 different missing data values. If `missing.type` is `TRUE` a separate list is created with the same variable names as the loaded data. For string variables the list value is `NULL`. For other variables the value is `NA` where the observation is not missing and 0–26 when the observation is missing. This is attached as the `"missing"` attribute of the returned value.

The default file format for Stata 13, `format-115`, is substantially different from those for Stata 5–12.

Value

A data frame with attributes. These will include `"datalabel"`, `"time.stamp"`, `"formats"`, `"types"`, `"val.labels"`, `"var.labels"` and `"version"` and may include `"label.table"` and `"expansion.table"`. Possible versions are `5`, `6`, `7`, `-7` (Stata 7SE, 'format-111'), `8` (Stata 8 and 9, 'format-113'), `10` (Stata 10 and 11, 'format-114'). and `12` (Stata 12, 'format-115').

The value labels in attribute `"val.labels"` name a table for each variable, or are an empty string. The tables are elements of the named list attribute `"label.table"`: each is an integer vector with names.

Author(s)

Thomas Lumley and R-core members: support for value labels by Brian Quistorff.

References

Stata Users Manual (versions 5 & 6), Programming manual (version 7), or online help (version 8 and later) describe the format of the files. Or directly at https://www.stata.com/help.cgi?dta_114 and https://www.stata.com/help.cgi?dta_113, but note that these have been changed since first published.

See Also

Different approaches are available in package **memisc** (see its help for `Stata.file`), function `read_dta` in package **haven** and package **readstata13**.

`write.dta`, `attributes`, `Date`, `factor`

Examples

```
write.dta(swiss,swissfile <- tempfile())
read.dta(swissfile)
```

`read.epiinfo` *Read Epi Info Data Files*

Description

Reads data files in the `.REC` format used by Epi Info versions 6 and earlier and by EpiData. Epi Info is a public domain database and statistics package produced by the US Centers for Disease Control and EpiData is a freely available data entry and validation system.

Usage

```
read.epiinfo(file, read.deleted = FALSE, guess.broken.dates = FALSE,
             thisyear = NULL, lower.case.names = FALSE)
```

Arguments

`file`	A filename, URL, or connection.
`read.deleted`	Deleted records are read if `TRUE`, omitted if `FALSE` or replaced with `NA` if `NA`.
`guess.broken.dates`	Attempt to convert dates with 0 or 2 digit year information (see 'Details').
`thisyear`	A 4-digit year to use for dates with no year. Defaults to the current year.
`lower.case.names`	Convert variable names to lowercase?

Details

Epi Info allows dates to be specified with no year or with a 2 or 4 digits. Dates with four-digit years are always converted to `Date` class. With the `guess.broken.dates` option the function will attempt to convert two-digit years using the operating system's default method (see Date) and will use the current year or the `thisyear` argument for dates with no year information.

If `read.deleted` is `TRUE` the `"deleted"` attribute of the data frame indicates the deleted records.

Value

A data frame.

Note

Some later versions of Epi Info use the Microsoft Access file format to store data. That may be readable with the **RODBC** package.

References

https://www.cdc.gov/epiinfo/, http://www.epidata.dk

See Also

DateTimeClasses

Examples

```
## Not run: ## That file is not available
read.epiinfo("oswego.rec", guess.broken.dates = TRUE, thisyear = "1972")

## End(Not run)
```

---

read.mtp *Read a Minitab Portable Worksheet*

Description

Return a list with the data stored in a file as a Minitab Portable Worksheet.

Usage

```
read.mtp(file)
```

Arguments

file	character variable with the name of the file to read. The file must be in Minitab Portable Worksheet format.

Value

A list with one component for each column, matrix, or constant stored in the Minitab worksheet.

Note

This function was written around 1990 for the format current then. Later versions of Minitab appear to have added to the format.

Author(s)

Douglas M. Bates

References

https://www.minitab.com/

Examples

```
## Not run:
read.mtp("ex1-10.mtp")

## End(Not run)
```

`read.octave` *Read Octave Text Data Files*

Description

Read a file in Octave text data format into a list.

Usage

```
read.octave(file)
```

Arguments

`file`	a character string with the name of the file to read.

Details

This function is used to read in files in Octave text data format, as created by `save -text` in Octave. It knows about most of the common types of variables, including the standard atomic (real and complex scalars, matrices, and N-d arrays, strings, ranges, and boolean scalars and matrices) and recursive (structs, cells, and lists) ones, but has no guarantee to read all types. If a type is not recognized, a warning indicating the unknown type is issued, it is attempted to skip the unknown entry, and `NULL` is used as its value. Note that this will give incorrect results, and maybe even errors, in the case of unknown recursive data types.

As Octave can read MATLAB binary files, one can make the contents of such files available to R by using Octave's load and save (as text) facilities as an intermediary step.

Value

A list with one named component for each variable in the file.

Author(s)

Stephen Eglen <stephen@gnu.org> and Kurt Hornik

References

https://www.gnu.org/software/octave/

`read.spss` *Read an SPSS Data File*

Description

`read.spss` reads a file stored by the SPSS `save` or `export` commands.

This was orignally written in 2000 and has limited support for changes in SPSS formats since (which have not been many).

Usage

```
read.spss(file, use.value.labels = TRUE, to.data.frame = FALSE,
          max.value.labels = Inf, trim.factor.names = FALSE,
          trim_values = TRUE, reencode = NA, use.missings = to.data.frame,
          sub = ".", add.undeclared.levels = c("sort", "append", "no"),
          duplicated.value.labels = c("append", "condense"),
          duplicated.value.labels.infix = "_duplicated_", ...)
```

Arguments

`file`	character string: the name of the file or URL to read.
`use.value.labels`	logical: convert variables with value labels into R factors with those levels? This is only done if there are at least as many labels as values of the variable (when values without a matching label are returned as `NA`).
`to.data.frame`	logical: return a data frame?
`max.value.labels`	logical: only variables with value labels and at most this many unique values will be converted to factors if `TRUE`.
`trim.factor.names`	logical: trim trailing spaces from factor levels?
`trim_values`	logical: should values and value labels have trailing spaces ignored when matching for `use.value.labels = TRUE`?
`reencode`	logical: should character strings be re-encoded to the current locale. The default, `NA`, means to do so in UTF-8 or latin-1 locales, only. Alternatively a character string specifying an encoding to assume for the file.
`use.missings`	logical: should information on user-defined missing values be used to set the corresponding values to `NA`?
`sub`	character string: If not `NA` it is used by `iconv` to replace any non-convertible bytes in character/factor input. Default is `"."`. For back compatibility with **foreign** versions <= 0.8-68 use `sub=NA`.
`add.undeclared.levels`	character: specify how to handle variables with at least one value label and further non-missing values that have no value label (like a factor levels in R).

For "sort" (the default) it adds undeclared factor levels to the already declared levels (and labels) and sort them according to level, for "append" it appends undeclared factor levels to declared levels (and labels) without sorting, and for "no" this does not convert to factor in case of numeric SPSS levels (not labels), and still converts to factor if the SPSS levels are characters and to.data.frame=TRUE. For back compatibility with **foreign** versions <= 0.8-68 use add.undeclared.levels="no" (not recommended as this may convert some values with missing corresponding value labels to NA).

duplicated.value.labels
: character: what to do with duplicated value labels for different levels. For "append" (the default), the first original value label is kept while further duplicated labels are renamed to paste0(label,duplicated.value.labels.infix,level), for "condense", all levels with identical labels are condensed into exactly the first of these levels in R. Back compatibility with **foreign** versions <= 0.8-68 is not given as R versions >= 3.4.0 no longer support duplicated factor labels.

duplicated.value.labels.infix
: character: the infix used for labels of factor levels with duplicated value labels in SPSS (default "_duplicated_") if duplicated.value.labels="append".

...
: passed to as.data.frame if to.data.frame = TRUE.

Details

This uses modified code from the PSPP project (http://www.gnu.org/software/pspp/ for reading the SPSS formats.

If the filename appears to be a URL (of schemes 'http:', 'ftp:' or 'https:') the URL is first downloaded to a temporary file and then read. ('https:' is supported where supported by download.file with its current default method.)

Occasionally in SPSS, value labels will be added to some values of a continuous variable (e.g. to distinguish different types of missing data), and you will not want these variables converted to factors. By setting max.value.labels you can specify that variables with a large number of distinct values are not converted to factors even if they have value labels.

If SPSS variable labels are present, they are returned as the "variable.labels" attribute of the answer.

Fixed length strings (including value labels) are padded on the right with spaces by SPSS, and so are read that way by R. The default argument trim_values=TRUE causes trailing spaces to be ignored when matching to value labels, as examples have been seen where the strings and the value labels had different amounts of padding. See the examples for sub for ways to remove trailing spaces in character data.

URL https://docs.microsoft.com/en-us/windows/win32/intl/code-page-identifiers provides a list of translations from Windows codepage numbers to encoding names that iconv is likely to know about and so suitable values for reencode. Automatic re-encoding is attempted for apparent codepages of 200 or more in a UTF-8 or latin-1 locale: some other high-numbered codepages can be re-encoded on most systems, but the encoding names are platform-dependent (see iconvlist).

Value

A list (or optionally a data frame) with one component for each variable in the saved data set.

If what looks like a Windows codepage was recorded in the SPSS file, it is attached (as a number) as attribute `"codepage"` to the result.

There may be attributes `"label.table"` and `"variable.labels"`. Attribute `"label.table"` is a named list of value labels with one element per variable, either `NULL` or a named character vector. Attribute `"variable.labels"` is a named character vector with names the short variable names and elements the long names.

If there are user-defined missing values, there will be a attribute `"Missings"`. This is a named list with one list element per variable. Each element has an element `type`, a length-one character vector giving the type of missingness, and may also have an element `value` with the values corresponding to missingness. This is a complex subject (where the R and C source code for `read.spss` is the main documentation), but the simplest cases are types `"one"`, `"two"` and `"three"` with a corresponding number of (real or string) values whose labels can be found from the `"label.table"` attribute. Other possibilities are a finite or semi-infinite range, possibly plus a single value. See also http://www.gnu.org/software/pspp/manual/html_node/Missing-Observations.html#Missing-Observations.

Note

If SPSS value labels are converted to factors the underlying numerical codes will not in general be the same as the SPSS numerical values, since the numerical codes in R are always $1, 2, 3, \ldots$.

You may see warnings about the file encoding for SPSS `save` files: it is possible such files contain non-ASCII character data which need re-encoding. The most common occurrence is Windows codepage 1252, a superset of Latin-1. The encoding is recorded (as an integer) in attribute `"codepage"` of the result if it looks like a Windows codepage. Automatic re-encoding is done only in UTF-8 and latin-1 locales: see argument `reencode`.

Author(s)

Saikat DebRoy and the R-core team

See Also

A different interface also based on the PSPP codebase is available in package **memisc**: see its help for `spss.system.file`.

Examples

```
(sav <- system.file("files", "electric.sav", package = "foreign"))
dat <- read.spss(file=sav)
str(dat)   # list structure with attributes

dat <- read.spss(file=sav, to.data.frame=TRUE)
str(dat)   # now a data.frame


### Now we use an example file that is not very well structured and
### hence may need some special treatment with appropriate argument settings.
```

```
### Expect lots of warnings as value labels (corresponding to R factor labels) are uncomplete,
### and an unsupported long string variable is present in the data
(sav <- system.file("files", "testdata.sav", package = "foreign"))

### Examples for add.undeclared.levels:
## add.undeclared.levels = "sort" (default):
x.sort <- read.spss(file=sav, to.data.frame = TRUE)
## add.undeclared.levels = "append":
x.append <- read.spss(file=sav, to.data.frame = TRUE,
    add.undeclared.levels = "append")
## add.undeclared.levels = "no":
x.no <- read.spss(file=sav, to.data.frame = TRUE,
    add.undeclared.levels = "no")

levels(x.sort$factor_n_undeclared)
levels(x.append$factor_n_undeclared)
str(x.no$factor_n_undeclared)


### Examples for duplicated.value.labels:
## duplicated.value.labels = "append" (default)
x.append <- read.spss(file=sav, to.data.frame=TRUE)
## duplicated.value.labels = "condense"
x.condense <- read.spss(file=sav, to.data.frame=TRUE,
    duplicated.value.labels = "condense")

levels(x.append$factor_n_duplicated)
levels(x.condense$factor_n_duplicated)

as.numeric(x.append$factor_n_duplicated)
as.numeric(x.condense$factor_n_duplicated)


## Long Strings (>255 chars) are imported in consecutive separate variables
## (see warning about subtype 14):
x <- read.spss(file=sav, to.data.frame=TRUE, stringsAsFactors=FALSE)

cat.long.string <- function(x, w=70) cat(paste(strwrap(x, width=w), "\n"))

## first part: x$string_500:
cat.long.string(x$string_500)
## second part: x$STRIN0:
cat.long.string(x$STRIN0)
## complete long string:
long.string <- apply(x[,c("string_500", "STRIN0")], 1, paste, collapse="")
cat.long.string(long.string)
```

`read.ssd` *Obtain a Data Frame from a SAS Permanent Dataset, via read.xport*

Description

Generates a SAS program to convert the ssd contents to SAS transport format and then uses `read.xport` to obtain a data frame.

Usage

```
read.ssd(libname, sectionnames,
   tmpXport=tempfile(), tmpProgLoc=tempfile(), sascmd="sas")
```

Arguments

`libname`	character string defining the SAS library (usually a directory reference)
`sectionnames`	character vector giving member names. These are files in the `libname` directory. They will usually have a `.ssd0x` or `.sas7bdat` extension, which should be omitted. Use of ASCII names of at most 8 characters is strongly recommended.
`tmpXport`	character string: location where temporary xport format archive should reside – defaults to a randomly named file in the session temporary directory, which will be removed.
`tmpProgLoc`	character string: location where temporary conversion SAS program should reside – defaults to a randomly named file in session temporary directory, which will be removed on successful operation.
`sascmd`	character string giving full path to SAS executable.

Details

Creates a SAS program and runs it.

Error handling is primitive.

Value

A data frame if all goes well, or `NULL` with warnings and some enduring side effects (log file for auditing)

Note

This requires SAS to be available. If you have a SAS dataset without access to SAS you will need another product to convert it to a format such as `.csv`, for example 'Stat/Transfer' or 'DBMS/Copy' or the 'SAS System Viewer' (Windows only).

SAS requires section names to be no more than 8 characters. This is worked by the use of symbolic links: these are barely supported on Windows.

Author(s)

For Unix: VJ Carey <stvjc@channing.harvard.edu>

See Also

`read.xport`

Examples

```
## if there were some files on the web we could get a real
## runnable example
## Not run:
R> list.files("trialdata")
 [1] "baseline.sas7bdat" "form11.sas7bdat"   "form12.sas7bdat"
 [4] "form13.sas7bdat"   "form22.sas7bdat"   "form23.sas7bdat"
 [7] "form3.sas7bdat"    "form4.sas7bdat"    "form48.sas7bdat"
[10] "form50.sas7bdat"   "form51.sas7bdat"   "form71.sas7bdat"
[13] "form72.sas7bdat"   "form8.sas7bdat"    "form9.sas7bdat"
[16] "form90.sas7bdat"   "form91.sas7bdat"
R> baseline <- read.ssd("trialdata", "baseline")
R> form90 <- read.ssd("trialdata", "form90")

## Or for a Windows example
sashome <- "/Program Files/SAS/SAS 9.1"
read.ssd(file.path(sashome, "core", "sashelp"), "retail",
         sascmd = file.path(sashome, "sas.exe"))

## End(Not run)
```

`read.systat` *Obtain a Data Frame from a Systat File*

Description

`read.systat` reads a rectangular data file stored by the Systat `SAVE` command as (legacy) `*.sys` or more recently `*.syd` files.

Usage

```
read.systat(file, to.data.frame = TRUE)
```

Arguments

`file`	character variable with the name of the file to read
`to.data.frame`	return a data frame (otherwise a list)

Details

The function only reads those Systat files that are rectangular data files (`mtype = 1`), and warns when files have non-standard variable name codings. The files tested were produced on MS-DOS and Windows: files for the Mac version of Systat have a completely different format.

The C code was originally written for an add-on module for Systat described in Bivand (1992 paper). Variable names retain the trailing dollar in the list returned when `to.data.frame` is `FALSE`, and in that case character variables are returned as is and filled up to 12 characters with blanks on the right. The original function was limited to reading Systat files with up to 256 variables (a Systat limitation); it will now read up to 8192 variables.

If there is a user comment in the header this is returned as attribute `"comment"`. Such comments are always a multiple of 72 characters (with a maximum of 720 chars returned), normally padded with trailing spaces.

Value

A data frame (or list) with one component for each variable in the saved data set.

Author(s)

Roger Bivand

References

Systat Manual, 1987, 1989

Bivand, R. S. (1992) SYSTAT-compatible software for modelling spatial dependence among observations. *Computers and Geosciences* **18**, 951–963.

Examples

```
summary(iris)
iris.s <- read.systat(system.file("files/Iris.syd", package="foreign")[1])
str(iris.s)
summary(iris.s)
```

`read.xport` *Read a SAS XPORT Format Library*

Description

Reads a file as a SAS XPORT format library and returns a list of data.frames.

Usage

```
read.xport(file, ...)
```

Arguments

`file`	character variable with the name of the file to read. The file must be in SAS XPORT format.
`...`	passed to `as.data.frame` when creating the data frames.

Value

If there is a more than one dataset in the XPORT format library, a named list of data frames, otherwise a data frame. The columns of the data frames will be either numeric (corresponding to numeric in SAS) or factor (corresponding to character in SAS). All SAS numeric missing values (including special missing values represented by `._`, `.A` to `.Z` by SAS) are mapped to R `NA`.

Trailing blanks are removed from character columns before conversion to a factor. Some sources claim that character missing values in SAS are represented by `' '` or `''`: these are not treated as R missing values.

Author(s)

Saikat DebRoy <saikat@stat.wisc.edu>

References

SAS Technical Support document TS-140: "The Record Layout of a Data Set in SAS Transport (XPORT) Format" available at https://support.sas.com/content/dam/SAS/support/en/technical-papers/record-layout-of-a-sas-version-5-or-6-data-set-in-sas-transport-xport-format.pdf

See Also

`lookup.xport`

Examples

```
## Not run: ## no XPORT file is installed
read.xport("test.xpt")

## End(Not run)
```

S3 read functions *Read an S3 Binary or data.dump File*

Description

Reads binary data files or `data.dump` files that were produced in S version 3.

Usage

```
data.restore(file, print = FALSE, verbose = FALSE, env = .GlobalEnv)
read.S(file)
```

Arguments

`file`	the filename of the S-PLUS `data.dump` or binary file.
`print`	whether to print the name of each object as read from the file.
`verbose`	whether to print the name of every subitem within each object.
`env`	environment within which to create the restored object(s).

Details

`read.S` can read the binary files produced in some older versions of S-PLUS on either Windows (versions 3.x, 4.x, 2000) or Unix (version 3.x with 4 byte integers). It automatically detects whether the file was produced on a big- or little-endian machine and adapts itself accordingly.

`data.restore` can read a similar range of files produced by `data.dump` and for newer versions of S-PLUS, those from `data.dump(.....,oldStyle=TRUE)`.

Not all S3 objects can be handled in the current version. The most frequently encountered exceptions are functions and expressions; you will also have trouble with objects that contain model formulas. In particular, comments will be lost from function bodies, and the argument lists of functions will often be changed.

Value

For `read.S`, an R version of the S3 object.

For `data.restore`, the name of the file.

Author(s)

Duncan Murdoch

Examples

```
## if you have an S-PLUS _Data file containing 'myobj'
## Not run: read.S(file.path("_Data", "myobj"))
data.restore("dumpdata", print = TRUE)

## End(Not run)
```

---

write.arff *Write Data into ARFF Files*

Description

Writes data into Weka Attribute-Relation File Format (ARFF) files.

Usage

```
write.arff(x, file, eol = "\n", relation = deparse(substitute(x)))
```

Arguments

`x`	the data to be written, preferably a matrix or data frame. If not, coercion to a data frame is attempted.
`file`	either a character string naming a file, or a connection. `""` indicates output to the standard output connection.
`eol`	the character(s) to print at the end of each line (row).
`relation`	The name of the relation to be written in the file.

Details

relation will be passed through make.names before writing to the file, in an attempt to it them acceptable to Weka, and column names what do not start with an alphabetic character will have X prepended.

However, the references say that ARFF files are ASCII files, and that encoding is not enforced.

References

Attribute-Relation File Format https://waikato.github.io/weka-wiki/formats_and_processing/arff/.

See Also

read.arff; functions write.arff and read.arff in package **RWeka** which provide some support for logicals via conversion to or from factors.

Examples

```
write.arff(iris, file = "")
```

---

write.dbf *Write a DBF File*

Description

The function tries to write a data frame to a DBF file.

Usage

```
write.dbf(dataframe, file, factor2char = TRUE, max_nchar = 254)
```

Arguments

dataframe	a data frame object.
file	a file name to be written to.
factor2char	logical, default TRUE, convert factor columns to character: otherwise they are written as the internal integer codes.
max_nchar	The maximum number of characters allowed in a character field. Strings which exceed this will be truncated with a warning. See Details.

Details

Dots in column names are replaced by underlines in the DBF file, and names are truncated to 11 characters.

Only vector columns of classes `"logical"`, `"numeric"`, `"integer"`, `"character"`, `"factor"` and `"Date"` can be written. Other columns should be converted to one of these.

Maximum precision (number of digits including minus sign and decimal sign) for numeric is 19 - scale (digits after the decimal sign) which is calculated internally based on the number of digits before the decimal sign.

The original DBASE format limited character fields to 254 bytes. It is said that Clipper and FoxPro can read up to 32K, and it is possible to write a reader that could accept up to 65535 bytes. (The documentation suggests that only ASCII characters can be assumed to be supported.) Readers expecting the older standard (which includes Excel 2003, Access 2003 and OpenOffice 2.0) will truncate the field to the maximum width modulo 256, so increase `max_nchar` only if you are sure the intended reader supports wider character fields.

Value

Invisible `NULL`.

Note

Other applications have varying abilities to read the data types used here. Microsoft Access reads `"numeric"`, `"integer"`, `"character"` and `"Date"` fields, including recognizing missing values, but not `"logical"` (read as `0,-1`). Microsoft Excel understood all possible types but did not interpret missing values in character fields correctly (showing them as character nuls).

Author(s)

Nicholas J. Lewin-Koh, modified by Roger Bivand and Brian Ripley; shapelib by Frank Warmerdam.

References

http://shapelib.maptools.org/

https://www.clicketyclick.dk/databases/xbase/format/data_types.html

See Also

`read.dbf`

Examples

```
str(warpbreaks)
try1 <- paste(tempfile(), ".dbf", sep = "")
write.dbf(warpbreaks, try1, factor2char = FALSE)
in1 <- read.dbf(try1)
str(in1)
try2 <- paste(tempfile(), ".dbf", sep = "")
```

```
write.dbf(warpbreaks, try2, factor2char = TRUE)
in2 <- read.dbf(try2)
str(in2)
unlink(c(try1, try2))
```

`write.dta` *Write Files in Stata Binary Format*

Description

Writes the data frame to file in the Stata binary format. Does not write array variables unless they can be `drop`-ed to a vector.

Frozen: will not support Stata formats after 10 (also used by Stata 11).

Usage

```
write.dta(dataframe, file, version = 7L,
          convert.dates = TRUE, tz = "GMT",
          convert.factors = c("labels", "string", "numeric", "codes"))
```

Arguments

`dataframe`	a data frame.
`file`	character string giving filename.
`version`	integer: Stata version: 6, 7, 8 and 10 are supported, and 9 is mapped to 8, 11 to 10.
`convert.dates`	logical: convert `Date` and `POSIXct` objects: see section 'Dates'.
`tz`	timezone for date conversion.
`convert.factors`	how to handle factors.

Details

The major difference between supported file formats in Stata versions is that version 7.0 and later allow 32-character variable names (5 and 6 were restricted to 8-character names). The `abbreviate` function is used to trim variable names to the permitted length. A warning is given if this is needed and it is an error for the abbreviated names not to be unique. Each version of Stata is claimed to be able to read all earlier formats.

The columns in the data frame become variables in the Stata data set. Missing values are handled correctly.

There are four options for handling factors. The default is to use Stata 'value labels' for the factor levels. With `convert.factors = "string"`, the factor levels are written as strings (the name of the value label is taken from the `"val.labels"` attribute if it exists or the variable name otherwise). With `convert.factors = "numeric"` the numeric values of the levels are written, or `NA` if they cannot be coerced to numeric. Finally, `convert.factors = "codes"` writes the underlying integer

codes of the factors. This last used to be the only available method and is provided largely for backwards compatibility.

If the "label.table" attribute contains value labels with names not already attached to a variable (not the variable name or name from "val.labels") then these will be written out as well.

If the "datalabel" attribute contains a string, it is written out as the dataset label otherwise the dataset label is "Written by R.".

If the "expansion.table" attribute exists expansion fields are written. This attribute should contain a list where each element is character vector of length three. The first vector element contains the name of a variable or "_dta" (meaning the dataset). The second element contains the characeristic name. The third contains the associated data.

If the "val.labels" attribute contains a character vector with a string label for each value then this is written as the value labels. Otherwise the variable names are used.

If the "var.labels" attribute contains a character vector with a string label for each variable then this is written as the variable labels. Otherwise the variable names are repeated as variable labels.

For Stata 8 or later use the default version = 7 – the only advantage of Stata 8 format over 7 is that it can represent multiple different missing value types, and R doesn't have them. Stata 10/11 allows longer format lists, but R does not make use of them.

Note that the Stata formats are documented to use ASCII strings – R does not enforce this, but use of non-ASCII character strings will not be portable as the encoding is not recorded. Up to 244 bytes are allowed in character data, and longer strings will be truncated with a warning.

Stata uses some large numerical values to represent missing values. This function does not currently check, and hence integers greater than 2147483620 and doubles greater than 8.988e+307 may be misinterpreted by Stata.

Value

NULL

Dates

Unless disabled by argument convert.dates = FALSE, R date and date-time objects (POSIXt classes) are converted into the Stata date format, the number of days since 1960-01-01. (For date-time objects this may lose information.) Stata can be told that these are dates by

```
format xdate %td;
```

It is possible to pass objects of class POSIXct to Stata to be treated as one of its versions of date-times. Stata uses the number of milliseconds since 1960-01-01, either excluding (format %tc) or counting (format %tC) leap seconds. So either an object of class POSICct can be passed to Stata with convert.dates = FALSE and converted in Stata, or 315619200 should be added and then multiplied by 1000 before passing to write.dta and assigning format %tc. Stata's comments on the first route are at https://www.stata.com/manuals13/ddatetime.pdf, but at the time of writing were wrong: R uses POSIX conventions and hence does not count leap seconds.

Author(s)

Thomas Lumley and R-core members: support for value labels by Brian Quistorff.

References

Stata 6.0 Users Manual, Stata 7.0 Programming manual, Stata online help (version 8 and later, also https://www.stata.com/help.cgi?dta_114 and https://www.stata.com/help.cgi?dta_113) describe the file formats.

See Also

read.dta, attributes, DateTimeClasses, abbreviate

Examples

```
write.dta(swiss, swissfile <- tempfile())
read.dta(swissfile)
```

`write.foreign` *Write Text Files and Code to Read Them*

Description

This function exports simple data frames to other statistical packages by writing the data as free-format text and writing a separate file of instructions for the other package to read the data.

Usage

```
write.foreign(df, datafile, codefile,
              package = c("SPSS", "Stata", "SAS"), ...)
```

Arguments

df	A data frame
datafile	Name of file for data output
codefile	Name of file for code output
package	Name of package
...	Other arguments for the individual writeForeign functions

Details

The work for this function is done by foreign:::writeForeignStata, foreign:::writeForeignSAS and foreign:::writeForeignSPSS. To add support for another package, eg Systat, create a function writeForeignSystat with the same first three arguments as write.foreign. This will be called from write.foreign when package="Systat".

Numeric variables and factors are supported for all packages: dates and times (Date, dates, date, and POSIXt classes) and logical vectors are also supported for SAS and characters are supported for SPSS.

For package="SAS" there are optional arguments dataname = "rdata" taking a string that will be the SAS data set name, validvarname taking either "V6" or "V7", and libpath = NULL taking a

string that will be the directory where the target SAS datset will be written when the generated SAS code been run.

For `package="SPSS"` there is an optional argument `maxchars = 32L` taking an integer that causes the variable names (not variable labels) to be abbreviated to not more than `maxchars` chars. For compatibility with SPSS version 12 and before, change this to `maxchars = 8L`. In single byte locales with SPSS versions 13 or later, this can be set to `maxchars = 64L`.

For `package="SPSS"`, as a side effect, the decimal indicator is always set by `SET DECIMAL=DOT.` which may override user settings of the indicator or its default derived from the current locale.

Value

Invisible `NULL`.

Author(s)

Thomas Lumley and Stephen Weigand

Examples

```
## Not run:
datafile <- tempfile()
codefile <- tempfile()
write.foreign(esoph, datafile, codefile, package="SPSS")
file.show(datafile)
file.show(codefile)
unlink(datafile)
unlink(codefile)

## End(Not run)
```

Index