The foreign Package

April 24, 2007

Priority recommended

Version 0.8-20

Date 2007-04-06

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

Depends R (>= 2.5.0), stats

Imports methods, utils

Maintainer R-core <R-core@r-project.org>

Author R-core members, Saikat DebRoy <saikat@stat.wisc.edu>, Roger Bivand
<Roger.Bivand@nhh.no> and others:

Description Functions for reading and writing data stored by statistical packages such as Minitab, S,
SAS, SPSS, Stata, Systat, ..., and for reading and writing .dbf (dBase) files.

License GPL version 2 or later

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 http://ftp.sas.com/techsup/download/technote/ts140.html.

See Also

[read.xport](#)

Examples

```
## Not run:
lookup.xport("transport")
## 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

```
## Not run:
read.S(file.path("_Data", "myobj"))
data.restore("dumpdata", print = TRUE)
## 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 http://www.cs.waikato.ac.nz/~ml/weka/arff.html

See Also

write.arff

---

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.

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' (`http://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 are 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.

Author(s)

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

References

`http://shapelib.maptools.org/`.

The Borland file specification *via* `http://www.wotsit.org/search.asp?s=database`.

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 5-9 binary format into a data frame.

Usage

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

Arguments

`file`	a filename as a character string.
`convert.dates`	
	Convert Stata dates to `Date` class?
`tz`	Previously used to specify time zone. Now deprecated.
`convert.factors`	
	Use Stata value labels to create factors? (version 6.0 or later).
`missing.type`	For version 8,9 only, 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

The variables in the Stata data set become the columns of the data frame. Missing values are correctly handled. The data label, variable labels, and timestamp are stored as attributes of the data frame. Nothing is done with variable characteristics.

Optionally, Stata dates (%d 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 8.0 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 code"missing" attribute of the returned value.

The option to allow underscores in variable names may become the default in future versions now that R supports their use.

Value

a data frame

Author(s)

Thomas Lumley

References

Stata Users Manual (versions 5 & 6), Programming manual (version 7), or online help (version 8,9) describe the format of the files.

See Also

[write.dta](), [attributes]() [Date]() [factor]()

Examples

```
data(swiss)
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 below).
`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

Epi Info 2000, the current version, uses the Microsoft Access file format to store data. This may be readable with the `RODBC` or `RDCOM` packages.

References

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

See Also

DateTimeClasses

Examples

```
## Not run:
data<-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.

Author(s)

Douglas M. Bates

References

http://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 -ascii` 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

http://www.octave.org/

---

read.spss   *Read an SPSS Data File*

Description

`read.spss` reads a file stored by the SPSS `save` and `export` commands and returns a list.

Usage

```
read.spss(file, use.value.labels=TRUE, to.data.frame=FALSE,
          max.value.labels=Inf, trim.factor.names=FALSE)
```

Arguments

`file`   character variable with the name of the file to read.

`use.value.labels`
                       Convert variables with value labels into R factors with those levels?

`to.data.frame`
                       return a data frame?

`max.value.labels`
                       Only variables with at most this many unique values will be converted to factors

`trim.factor.names`
                       Trim trailing spaces from factor levels?

Details

This uses modified code from the PSPP project for reading the SPSS formats.

Occasionally in SPSS value labels will be added to some values of a continuous variable (eg to distinguish different types of missing data), and you will not want these variables converted to factors. By setting `max.val.labels` you can specify that variables with a large number of distinct values are not converted to factors even if they have value labels. In addition, variables will not be converted to factors if there are non-missing values that have no value label. The value labels are then returned in the `"value.labels"` attribute of the variable.

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. See the examples for `sub` for ways to remove trailing spaces in string data.

Value

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

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,...

You may see warnings about the file encoding: 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.

Author(s)

Saikat DebRoy and the R Core team

Examples

```
## Not run:
read.spss("datafile")
## don't convert value labels to factor levels
read.spss("datafile",use.value.labels=FALSE)
## convert value labels to factors for variables with at most
## ten distinct values.
read.spss("datafile",max.val.labels=10)
## End(Not run)
```

---

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 dataframe.

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.
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).

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.

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 http://ftp.sas.com/techsup/download/technote/
ts140.html.

See Also

lookup.xport

Examples

```
## Not run:
read.xport("transport")
## 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")
```

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).

References

Attribute-Relation File Format http://www.cs.waikato.ac.nz/~ml/weka/arff.html

See Also

read.arff

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/>

<http://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 version 6.0 or 7.0 binary format. Does not write matrix variables.

Usage

```
write.dta(dataframe, file, version = 6,
          convert.dates = TRUE, tz = "GMT",
          convert.factors = c("labels", "string", "numeric", "codes"))
```

Arguments

dataframe	a data frame.
file	character string giving filename.
version	Stata version: 6 and 7 are supported.
convert.dates	
	Convert POSIXt objects to Stata dates?

```
tz                 timezone for date conversion
convert.factors
                   how to handle factors
```

Details

The major differences between Stata versions is that 7.0 allows 32-character variable names. The `abbreviate` function is used to trim long variables 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.

The columns in the data frame become variables in the Stata data set. Missing values are correctly handled. Optionally, R date/time objects (`POSIXt` classes) are converted into the Stata format. This loses information – Stata dates are in days since 1960-1-1. `POSIXct` objects can be written without conversion but will not be understood as dates by Stata; `POSIXlt` objects cannot be written without conversion.

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. 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.

For Stata 8 or 9 use `version=7` – the only advantage of Stata 8 format is that it can represent multiple different missing value types, and R doesn't have them.

Value

```
NULL
```

Author(s)

Thomas Lumley

References

Stata 6.0 Users Manual, Stata 7.0 Programming manual, Stata 8.0, 9.0 online help 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) 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 and `validvarname` taking either `"V6"` or `"V7"`.

Value

None

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