

Package ‘xlsReadWrite’

January 2, 2012

Version 1.5.4

Title Read and write Excel files (.xls)

Date 2011-01-25

Author Hans-Peter Suter

Maintainer Hans-Peter Suter <support@swissr.org>

Depends R (>= 2.9.0)

Suggests tools, RUnit

Description Natively read and write Excel files (.xls/BIFF8)

License GPL-2

LicenseDetails Exception to allow linking of proprietary Flexcel code

OS_type windows

SystemRequirements Win 32-bit, dl regular shlib with 'xls.getshlib()' or manually (~420 KB)

URL <http://www.swissr.org>

BugReports <http://dev.swissr.org/projects/xlsreadwrite>

Repository CRAN

Date/Publication 2011-01-25 16:56:41

R topics documented:

xlsReadWrite-package	2
read.xls	4
write.xls	7
xls.datetime	8
xls.getshlib	11

Index	13
--------------	-----------

xlsReadWrite-package *Read and write Excel files*

Description

1. Overview
2. CRAN/pro versions
3. RUnit tests
4. Future plans
5. Download and compilation
6. Links and contact

1. Overview

xlsReadWrite is a packages which allows to read and write Excel files. Currently supported is the .xls (BIFF8) file format.

Getting started

- check out [read.xls](#), [write.xls](#) and [xls.getshlib](#)
- type: `help(package = "xlsReadWrite")` for more information

Why xlsReadWrite?

- it is fast
- well documented and (RUnit) tested
- the package does *not* have any external dependencies
- it has a nice simple interface and *just works fine* since ~4 years

On the less rosy side: xlsReadWrite still is windows only and uses proprietary 3rd party code (which means only our own code is, and can be, free). The package is written in Pascal, a very nice but rather obscure language in the R world.

Other solutions

xlsReadWrite is not the only game in town, we know of the following packages (brackets indicate dependencies): *RODBC* (drivers), *gdata* (Perl), *RExcel* or *COM* (Excel), *dataframes2xls* (Python), *xlsx* (Apache POI, Java). Last but not least, you may use plain old .csv files (none) or consider our own *xlsReadWritePro* (none) shareware package.

2. CRAN/pro versions

Besides the regular xlsReadWrite package there is a special CRAN version.
Reason (as copied from the startup message):

BACKGROUND: Our own xlsReadWrite code is free, but we also use proprietary code (Flexcel, tmssoftware.com) which can only be distributed legally in precompiled, i.e. binary form. As CRAN 'generally does not accept submissions of precompiled binaries due to security reasons' we only provide a placeholder and you can download the binary shlib separately.

The xlsReadWritePro version is our second attempt and goes further than the free version:

- append data to existing files
- work with in-memory Excel 'files' (called xls-obj)

- manage sheets (select, insert, copy, rename, delete, hide, info, list)
- support images (add, delete, count, list, export, info)
- address confined data areas (rows, cols, from, to, (named) ranges, cells)

- read and write formula values
- get file and sheet related info
- more date time helper functions

- formal support

By purchasing the xlsReadWritePro shareware version you help support our development and maintenance effort. We are grateful that quite some people and companies did and both, the free and the pro version shall benefit. The trial (<http://www.swissr.org>) is free and the two versions do coexist just fine. So, why not check out xlsReadWritePro now and see for yourself?

3. RUnit tests

There are more than 50 unit tests which should cover every aspect of the package and ensure and maintain code quality. The tests also serve as an extended reference going beyond the examples.

You can easily run the tests on your system, the RUnit test files are located at:

- `R_HOME/library/xlsReadWrite/unitTest/runit*.R`
- `R_HOME/library/xlsReadWrite/unitTest/_manual_execution.R`

4. Future plans

xlsReadWrite is mature and no big changes are planned. Certainly some internal refactoring would be beneficial and some features could be backported from the pro version.

In 2011 Delphi should be released for Mac, Linux and Win64, and we plan to support these platforms eventually. The underlying library is being developed for .xlsx and we are looking forward to it (probably as a separate package).

5. Download and compilation

Current version: <http://www.swissr.org/download>
All resources: <http://dl.dropbox.com/u/2602516/swissrpkg/index.html>
Github repo: <http://github.com/swissr/xlsReadWrite>
CRAN: <http://cran.r-project.org/web/packages/xlsReadWrite/index.html>

Compilation

Prerequisites for compiling: Delphi 2007 and Flexcel (<http://www.tmssoftware.com>, the core library doing the Excel-work). Then follow these steps:

1. Clone the github repo and initialize the RPascal submodule
2. Adapt paths in 'include.mk'
3. Regarding Flexcel code:
 - in FLXCOMPILER.INC: disable the FLEXCEL define'
 - 'src/pas/vFlexcelCommits.txt' has the currently used Flexcel version
4. Run make <target> in cmd.exe (system path will be modified temporarily)
5. Alternatively compile/run project in the Delphi IDE.

6. Links and contact

General info: <http://www.swissr.org>
Bugtracker: <http://dev.swissr.org/projects/xlsReadWrite>
Forum: <http://dev.swissr.org/projects/xlsReadWrite/boards>
Email: support at swissr.org

January 25, 2011 / Hans-Peter

read.xls

Read Excel files

Description

Reads an Excel file into a data.frame or matrix. Supported is the .xls (BIFF8) file format.

Usage

```
read.xls(file,  
         colNames = TRUE,  
         sheet = 1,  
         type = "data.frame",  
         from = 1,  
         rowNames = NA, colClasses = NA, checkNames = TRUE,
```

```

dateTime = "numeric",
naStrings = NA,
stringsAsFactors = default.stringsAsFactors()

```

Arguments

file	name of an Excel file. Path may be absolute or relative to the current working directory.
colNames	with TRUE the first row of the sheet or the 'from'-starting area will be used for the colnames. Provide a character vector to use custom colnames. The above colname values may possibly get modified depending on the checkNames argument. With FALSE and for missing values, defaults will be used, i.e. a 'V' followed by the column number.
sheet	case sensitive character string or one-based number indicating the Excel sheet to read from.
type	a character string indicating the type of the returned object. Either data.frame or else double, integer, logical or character to get a matrix.
from	row to start reading from.
rowNames	with TRUE the first column of the sheet will be used for the rownames. Provide a character vector to use custom rownames. The above rowname values may possibly get modified depending on the checkNames argument. With FALSE defaults will be used, i.e. numbers starting from one. With NA the first sheet column will be considered to contain rownames under the following conditions: 1) colNames is TRUE or contains a character vector, 2) there are at least two columns and the column name for the potential rowname-column is empty, 3) the first data cell contains a string which 4) is not "1".
colClasses	specify the column types of the resulting data.frame. Possible entries are: double, numeric, integer, logical, character, factor, NA, isodate, isotime and isodatetime. Either supply an entry for each column or else a scalar which will be recycled. With NA suitable column types will be determined based on the <i>first</i> non-empty cell of the 16 rows following the starting (from) row. An integer will be recognised as numeric. NA can be used together with other entries, e.g. c(NA, "double", NA, "isodate"). If nothing can be determined, all values become NA (logical) and a warning message will be printed.
checkNames	logical. With TRUE col- and rownames are guaranteed to be syntactically valid variable names, e.g. by prepending an "X" or translating invalid characters to "." (make.names is being used).
dateTime	scalar character controlling how Excel date values will be interpreted when no explicit colClasses have been given: isodatetime or numeric. 'isodatetime' will possibly be shortened to 'isodate' or 'isotime' and 'numeric' to 'integer'.
naStrings	a character vector of strings which are to be interpreted as NA-values. NA disables this interpretation, i.e. an Excel 'NA' string becomes a character or factor.
stringsAsFactors	logical, the default follows the global option default.stringsAsFactors .

Details

The character string arguments for `colNames` and `colClasses` can optionally include an entry for the column used for the rownames (entry will be discarded). For `rowNames`, the character vector length must be the same as the number of data rows.

Value

A `data.frame` or a matrix of the specified type. `NULL` if the sheet is empty.

Pro version

The pro version is our second attempt and goes further than the free version: + append data to existing files, + work with in-memory objects, e.g. to compose Excel reports with multiple sheets, + address confined data areas: rows, cols and from, to; (named) ranges and picking from individual cells, + manage sheets (select, insert, copy, rename, delete, hide), + read/write formula values, + support images (add, delete, count, list and export), + additional `oleDateTime` helper functions, + last but not least: formal support contact.

By purchasing the `xlsReadWritePro` shareware version you help support our development and maintenance effort. We are grateful that quite some people and companies did and both, the free and the pro version shall benefit. The trial (<http://www.swissr.org>) is free and the two versions do coexist just fine. So, why not check out `xlsReadWritePro` now and see for yourself?

See Also

[write.xls](#), [read.table](#), [matrix](#), [data.frame](#)

Examples

```
# --- only run for regular, i.e. non-cran version
shlib <- system.file("libs", if (nzchar(arch <- .Platform$r_arch)) arch else "",
  paste("xlsReadWrite", .Platform$dynlib.ext, sep = ""), package = "xlsReadWrite")
if (file.exists(shlib) && (file.info(shlib)$size > 20000)) {
# -----

  # path to Excel file
  rfile <- system.file("unitTests/data/origData.xls", package = "xlsReadWrite")

  # read as data.frame and as double

  read.xls(rfile)
  read.xls(rfile, type = "double")

  # read as integer with custom col-/rownames

  (rdata <- read.xls(rfile, from = 3, type = "integer",
    colNames = c("one", "one", "three", "four"),
    rowNames = paste("r", 1:12, sep = "")))
  # 2nd 'one' ends as 'one.1' due to 'checkNames'
  stopifnot(colnames(rdata)[2] == "one.1")
}
```

 write.xls

Write Excel files

Description

Saves a data.frame, matrix or vector as an Excel file. Currently supported is the .xls (BIFF8) file format.

Usage

```
write.xls(x, file,
         colNames = TRUE,
         sheet = 1,
         from = 1,
         rowNames = NA,
         naStrings = "")
```

Arguments

x	data to be written. A data.frame or else a matrix or vector of the type double, integer, logical or character. Vectors will be written in columns.
file	name of an Excel file. Path may be absolute or relative to the current working directory.
colNames	with TRUE the data colnames will be written into the first row of the sheet or 'from'-starting area. Provide a character vector to write custom colnames. With FALSE no colnames will be written.
sheet	case sensitive character string or one-based number indicating the Excel sheet to write to.
from	row to start writing from.
rowNames	with TRUE the data rownames will be written into the first column of the sheet. Provide a character vector to write custom rownames. With FALSE no rownames will be written. With NA the first column will be considered to receive the rownames under the following conditions: 1) colNames is TRUE or contains a character vector, 2) there are character rownames (supplied or in the data) and 3) the first entry thereof is not "1".
naStrings	the string to be used for NA values. An empty string clears (blanks) the cell, with NA the cell value will not be changed (only relevant for an eventual future append capability).

Details

New files are based on the template TemplateNew.xls. It can be modified and is located at R_HOME/library/xlsReadWrite/template (internal search is relative to the shlib located at R_HOME/library/xlsReadWrite/)

The character string arguments for `colNames` can optionally include an entry for the column used for the rownames (entry will be discarded). For `rowNames`, the character vector length must be the same as the number of data rows.

Pro version

See section in [read.xls](#).

See Also

[read.xls](#), [write.table](#)

Examples

```
# --- only run for regular, i.e. non-cran version
shlib <- system.file("libs", if (nzchar(arch <- .Platform$r_arch)) arch else "",
  paste("xlsReadWrite", .Platform$dynlib.ext, sep = ""), package = "xlsReadWrite")
if (file.exists(shlib) && (file.info(shlib)$size > 20000)) {
# -----

myval <- data.frame(
  Fertility = c(80.2, 83.1, 92.5),
  Agriculture = c(17, 45.1, 39.7),
  Testlogical = c(TRUE, TRUE, FALSE),
  Education = as.integer(c(12, 9, 5)),
  Catholic = c(9.96, 84.84, 93.4),
  Infant.Mortality = c(22.2, 22.2, 20.2),
  Testcharacter = c("Co", "De", "Fr"), stringsAsFactors = FALSE)

# write the data.frame...

write.xls(myval, "mytest.xls")

# ..then read and check (colClasses because we want logical and integer (for
# double) and character (for factor; could also have given stringsAsFactors))

mycls <- c("double", "double", "logical", "integer", "double", "double", "character")
wdata <- read.xls("mytest.xls", colClasses = mycls)
stopifnot(identical(wdata, myval))
}
```

xls.datetime

Date/Time conversion

Description

Several functions to convert Excel datetime values to and from strings.

Usage

```

dateTimeToStr(odate, format = "")
strToDateTime(sdate)
dateTimeToIsoStr(odate, isoformat = "YYYY-MM-DD hh:mm:ss")
isoStrToDateTime(sdate)

```

Arguments

odate	a numeric (double) datetime value from Excel.
format	formatting string, see list in details. With an empty string the system defaults settings for shortDateFormat and longTimeFormat will be used.
isoformat	one of the following character strings: YYYYMMDD (basic date), YYYY-MM-DD (extended date), YYYYMMDDhhmmss (basic date/time), YYYY-MM-DD hh:mm:ss (extended date/time) or YYYY-MM-DD hh:mm:ss.f (extended date/time including fractions (with 1, 2 or 3 decimal places))
sdate	a date as a string.

Details

dateTimeToStr converts a given double value to a string representation optionally using the formatting string.

strToDateTime converts a given character string to a double value. The string must contain a valid date and/or time value (in respect to the active locale). Names (e.g. days and months) are not supported.

isoStrToDateTime and dateTimeToIsoStr follow the ISO-8601 standard.

The following table lists the supported formatting strings (adapted from Delphi help):

c	ShortDateFormat followed by LongTimeFormat. Time is not displayed if midnight precisely.
d	day as a number without a leading zero (1-31).
dd	day as a number with a leading zero (01-31).
ddd	day as an abbreviation (Sun-Sat) using the ShortDayNames global variable.
dddd	day as a full name (Sunday-Saturday) using LongDayNames.
dddddd	date using ShortDateFormat variable.
dddddd	date using LongDateFormat.
e	year in the current period/era as a number without a leading zero.
ee	year in the current period/era as a number with a leading zero.
g	period/era as an abbreviation.
gg	period/era as a full name.
m	month as a number without a leading zero (1-12). If m immediately follows an h or hh, the minute rather than the month is displayed.
mm	month as a number with a leading zero (01-12).
mmm	month as an abbreviation (Jan-Dec) using the ShortMonthNames global variable.
mmmm	month as a full name (January-December) using LongMonthNames.
yy	year as a two-digit number (00-99).
yyyy	year as a four-digit number (0000-9999).
h	hour without a leading zero (0-23).

hh	hour with a leading zero (00-23).
n	minute without a leading zero (0-59).
nn	minute with a leading zero (00-59).
s	second without a leading zero (0-59).
ss	second with a leading zero (00-59).
z	millisecond without a leading zero (0-999).
zzz	millisecond with a leading zero (000-999).
t	time using the format given by ShortTimeFormat.
tt	time using the format given by LongTimeFormat.
am/pm	uses the 12-hour clock for the preceding h or hh specifier, and displays 'am' for any hour before noon, and 'pm' for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	uses the 12-hour clock for the preceding h or hh specifier, and displays 'a' for any hour before noon, and 'p' for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	uses the 12-hour clock for the preceding h or hh specifier, and displays the contents of the TimeAMString global variable for any hour before noon, and the contents of the TimePMString global variable for any hour after noon.
/	separator character given by the DateSeparator variable.
:	time separator character given by TimeSeparator.
'xx'/'xx'	characters enclosed in single or double quotes are displayed as-is, and do not affect formatting.

Pro version

See section in [read.xls](#).

See Also

[read.xls](#), [write.xls](#)

Examples

```
# --- only run for regular, i.e. non-cran version
shlib <- system.file("libs", if (nzchar(arch <- .Platform$r_arch)) arch else "",
  paste("xlsReadWrite", .Platform$dynlib.ext, sep = ""), package = "xlsReadWrite")
if (file.exists(shlib) && (file.info(shlib)$size > 20000)) {
# -----

  # convert iso datetime character to numeric datetime
  (idt <- isoStrToDateTime("2010-08-14 09:23:13"))

  # convert numeric datetime to string
  dateTimeToStr(idt)
  (sdt <- dateTimeToStr(idt, format = "c"))      # same as above (default)

  dateTimeToStr(idt, format = "dddddd")        # long date format
  (sd <- dateTimeToStr(idt, format = "dddddd")) # short date format
  (st <- dateTimeToStr(idt, format = "t"))      # short time format

  # convert character datetime to numeric
  (dt <- strToDateTime(sdt))
}
```

```
(dd <- strToDateTime(sd))
(tt <- strToDateTime(st))

stopifnot(isTRUE(all.equal(dt, (dd + tt))))
}
```

xls.getshlib *Download regular shlib (dll/so)*

Description

Downloads the regular shlib from our swissr dropbox account and replaces the currently used placeholder shlib with it. The size is about ~420 KB.

Using the default command, `xls.getshlib()`, should work just fine. Every step is documented, i.e. will be printed on the console.

The file will be downloaded with the `download.file` R function and thus follows the option `timeout` which defaults to 60 seconds.

There are quite a lot of different versions (shlib, R, (in future) platforms). If the correct shlib cannot be found and/or if there are permission issues, `xls.getshlib` will stop. But you can always download the full regular package or shlib manually from the URLs indicated in the details section below. Other than being less convenient it is the same. In case of any issues we are happy to hear about them (bug tracker/forum/email), thank you.

Usage

```
xls.getshlib(pkgvers = NA, url = NA, md5 = TRUE,
             reload.shlib = TRUE, tmpdir = tempdir())
```

Arguments

<code>pkgvers</code>	optional character string to override the default (which is: <code>(packageDescription("xlsReadWrite")\$Version)</code>). ‘ <code>pkgvers</code> ’ is used to replace placeholder in dropbox download url.
<code>url</code>	optional character string to indicate a custom url to the zipped file containing the shlib. Local, i.e. <code>file://<my url></code> urls are possible. If an url string has been given, the ‘ <code>pkgvers</code> ’ argument will not be used.
<code>md5</code>	with <code>TRUE</code> an md5 value will be retrieved from a file (with ‘ <code>.md5.txt</code> ’ suffix) located at the url indicated. A character string is assumed to be the md5 value to be used for checking. <code>FALSE</code> disables the check.
<code>reload.shlib</code>	<code>TRUE</code> to replace the loaded (placeholder) library. With <code>FALSE</code> the shlib will only be downloaded (path will be displayed).
<code>tmpdir</code>	temporary folder to downloaded file.

Details

The shlibs and all regular, cran and source packages are available from our swissr dropbox account. The source code is at github and the most recent releases can be downloaded directly from our main site. Links:

- <http://dl.dropbox.com/u/2602516/swissrpkg/index.html>
- <http://github.com/swissr/xlsreadwrite>
- <http://www.swissr.org/download>

Why do you need to download the shlib from an external source? Two (three) reasons:

1. our own xlsReadWrite code is free (GPL-2), but we also use proprietary code (Flexcel, tmssoftware.com) which can only be distributed legally in precompiled, i.e. binary form.
2. as CRAN '*generally does not accept submissions of precompiled binaries due to security reasons*' we only provide a placeholder and you have to download the binary shlib separately.
3. xlsReadWrite is written in Pascal and CRAN most probably would not be able/interested to support suitable compilers, i.e. Delphi and/or FPC. (We did not ask about compilers though, because the first point already rules out a normal open source distribution of the package).

There have been thorough tests initially but we do not give ANY GUARANTEES AT ALL. Eventually inspect the xls.getshlib code and/or type ?xlsReadWrite to find instructions about how to compile the package for yourself (Delphi/Flexcel needed).

Pro version

See section in [read.xls](#).

See Also

[read.xls](#), [write.xls](#), [download.file](#) and [options](#) for the timeout.

Examples

```
## Not run:  
xls.getshlib()  
  
## End(Not run)
```

Index

- *Topic **chron**
 - xls.datetime, 8
- *Topic **file**
 - read.xls, 4
 - write.xls, 7
 - xls.getshlib, 11
- *Topic **package**
 - xlsReadWrite-package, 2

- data.frame, 6
- dateTimeToIsoStr (xls.datetime), 8
- dateTimeToStr (xls.datetime), 8
- default.stringsAsFactors, 5
- download.file, 12

- isoStrToDateTime (xls.datetime), 8

- make.names, 5
- matrix, 6

- options, 12

- read.table, 6
- read.xls, 2, 4, 8, 10, 12

- strToDateTime (xls.datetime), 8

- write.table, 8
- write.xls, 2, 6, 7, 10, 12

- xls.datetime, 8
- xls.getshlib, 2, 11
- xlsReadWrite (xlsReadWrite-package), 2
- xlsReadWrite-package, 2