

Package ‘WriteXLS’

May 26, 2019

Version 5.0.0

Date 2019-05-24

Title Cross-Platform Perl Based R Function to Create Excel 2003 (XLS)
and Excel 2007 (XLSX) Files

Description

Cross-platform Perl based R function to create Excel 2003 (XLS) and Excel 2007 (XLSX) files from one or more data frames. Each data frame will be written to a separate named worksheet in the Excel spreadsheet. The worksheet name will be the name of the data frame it contains or can be specified by the user.

Author Marc Schwartz <marc_schwartz@me.com> and various authors for Perl modules listed in each .pm file.

Copyright The copyright holders of the Perl files are listed in each .pm file under the Perl directory.

Maintainer Marc Schwartz <marc_schwartz@me.com>

License GPL (>= 2)

SystemRequirements Perl

Imports utils

URL <https://github.com/marcschwartz/WriteXLS>

BugReports <https://github.com/marcschwartz/WriteXLS/issues>

NeedsCompilation no

Repository CRAN

Date/Publication 2019-05-25 22:50:13 UTC

R topics documented:

testPerl	2
WriteXLS	3
Index	8

testPerl

Test Perl installation and required modules for WriteXLS()

Description

Test Perl installation and required modules for WriteXLS()

Usage

```
testPerl(perl = "perl", verbose = TRUE)
```

Arguments

perl	Name of the perl executable to be called.
verbose	Output test result messages.

Details

This function will test your current system to be sure that Perl is installed and if so, whether or not all of the Perl modules required for WriteXLS() are present.

Success and/or error messages as appropriate will be output.

Value

A boolean value (TRUE or FALSE). TRUE if Perl and ALL required modules are found

Note

Please be sure to read the included INSTALL file (in the main package installation directory) for additional details on meeting the requirements for Perl and the additional Perl modules that are necessary for WriteXLS to work properly. The file includes platform specific recommendations for common scenarios. The path to the package installation directory can be located using `system.file(package = "WriteXLS")`.

A working installed version of Perl must be present in the current system searchpath or the exact path of the perl executable must be provided via the perl argument. Perl modules required for this function that may not be part of a default Perl installation are included with this package. These modules include:

Archive::Zip, OLE::Storage_Lite, Parse::RecDescent, Spreadsheet::WriteExcel, Excel::Writer::XLSX and Text::CSV_PP

Author(s)

Marc Schwartz <marc_schwartz@me.com>

Many thanks to Prof. Brian Ripley for his assistance in the testing of this package.

See Also[WriteXLS](#)

WriteXLS	<i>Cross-platform Perl based R function to create Excel 2003 (XLS) or Excel 2007 (XLSX) files</i>
----------	---------------------------------------------------------------------------------------------------

Description

Writes one or more R data frames to an Excel 2003 or Excel 2007 file

Usage

```
WriteXLS(x, ExcelFileName = "R.xls", SheetNames = NULL, perl = "perl",
         verbose = FALSE, Encoding = c("UTF-8", "latin1", "cp1252"),
         row.names = FALSE, col.names = TRUE,
         AdjWidth = FALSE, AutoFilter = FALSE, BoldHeaderRow = FALSE,
         na = "",
         FreezeRow = 0, FreezeCol = 0,
         envir = parent.frame())
```

Arguments

x	A character vector or factor containing the names of one or more R data frames; A character vector or factor containing the name of a single list which contains one or more R data frames; a single list object of one or more data frames; a single data frame object.
ExcelFileName	The name of the Excel file to be created. If the file extension is <i>.XLS</i> , an Excel 2003 file will be created. If the file extension is <i>.XLSX</i> , an Excel 2007 file will be created. Must be a valid Excel filename. May include an existing path. <code>normalizePath</code> is used to support tilde expansion, etc.
SheetNames	A character vector containing the names of each worksheet to be created. If NULL (the default), the names of the dataframes will be used instead. Worksheet names may be up to 31 characters in length and must be unique. If specified, <code>length(SheetNames)</code> must be the same as <code>length(x)</code> . NOTE: The order of the names here must match the order of the data frames as listed in x.
perl	Name of the perl executable to be called.
verbose	Output step-by-step status messages during the creation of the Excel file. Default is FALSE.
Encoding	Define the character encoding to be used for the exported data frames. Defaults to UTF-8.
row.names	If TRUE, the row names of the data frames are included in the Excel file worksheets.
col.names	If TRUE, the column names of the data frames are included in the Excel file worksheets.

AdjWidth	If TRUE, will adjust the worksheet column widths based upon the longest entry in each column. This is approximate.
AutoFilter	If TRUE, will add autofiltering to each column in each worksheet. Note that not all spreadsheet applications support this feature.
BoldHeaderRow	If TRUE, will apply a bold font to the header row for each worksheet.
na	The string to use for missing values in the data. Defaults to ""
FreezeRow	Rows including this row and above this row will be frozen and not scroll. The default value of 0 will scroll the entire sheet. Note that not all spreadsheet applications support this feature.
FreezeCol	Columns including this column and to the left of this column will be frozen and not scroll. The default value of 0 will scroll the entire sheet. Note that not all spreadsheet applications support this feature.
envir	The environment in which to look for the data frames named in x. This defaults to the environment in which WriteXLS was called.

Details

This function takes: a character vector or factor containing the names of one or more R data frames; A character vector or factor containing the name of a single list which contains one or more R data frames; a single list object containing one or more data frames; a single data frame object and exports them to an Excel 2003 or 2007 spreadsheet file. Each data frame will be written to a separate worksheet in the same Excel file.

The order of the worksheets created in the Excel file will match the order of the entries in x.

The actual creation of the Excel file is performed by Perl scripts called WriteXLS.pl (for XLS files) and WriteXLSX.pl (for XLSX files), which are included with this package.

Note that the named Excel file, if it already exists, will be overwritten and no warning is given. In addition, if the file exists and is open by another application (eg. Excel, OO.org, LibreOffice, etc.) you will likely get an error message regarding the inability to open the file and/or that the file is already in use by another application or user. Errors can also occur if the file has been marked as read-only or if your access rights do not allow you to overwrite the file or write to the folder you have indicated in the path to the file.

There is an intermediate step, where the R data frames are first written to CSV files using `wri telines` with argument `useBytes = TRUE` before being written to the Excel file by the relevant Perl script. `tempdir` is used to determine the current R session temporary directory and a new sub-directory called "WriteXLS" will be created there. The CSV files will be written to that directory and both the files and the directory will be deleted prior to the function terminating normally using `on.exit`. It is possible that these will remain in place if this function terminates abnormally or is aborted prior to completion.

Since `as.character` is used to coerce data frame column content to character vectors prior to export, data types supported by `as.character` will be exported to their character representation correctly. For other data types, it is recommended that you first coerce them to character columns, formatted as you require, and then use `WriteXLS` to create the Excel file.

All of the CSV files will be created prior to the creation of the Excel file as the Perl script will loop over them as part of the process. Thus, sufficient free disk space must be available for these files and the Excel file at the same time. Note, importantly, that in the course of creating XLSX files,

which are ZIP compressed XML files, Perl will require free disk space that is some multiple of the size of the resultant XLSX file, in the course of creating the XML files before compression into the XLSX file. Thus, additional free disk space, in addition to the CSV files noted above, will also be required temporarily.

A text file called "SheetNames.txt" will be created in the same temporary directory as the CSV files. This file will contain the sheet names, one per line and will be used by the Perl script to name the worksheets in the Excel file.

Each worksheet will be named using either the names in SheetNames, the names of the data frames in `x` or the names of the list elements if `x` is a list (up to the first 31 characters, which is an Excel limitation). If any the data frame names specified in `x` are longer than 31 characters, they will be truncated to 31 characters. SheetNames if specified, will be checked to make sure that all of the entries are less than or equal to 31 characters. If not, an error message will be displayed.

Note that the order of the names in SheetNames MUST match the order of the data frames named in `x`.

Note that the worksheets must have unique names. Thus, if SheetNames is NULL, the data frame names will be checked to be sure that they are unique up through the first 31 characters. If SheetNames is specified, the entries will be checked to be sure that they are unique. If not, an error message will be displayed.

Note that the following characters are not allowed for Excel worksheet names: `[]:*\?/\`

The data frame column names will be exported "as is" and will be the first row in the corresponding worksheet, if `col.names = TRUE`.

As of version 4.1.0, the function will preserve numbers with leading zeroes in the resultant Excel file, whereas prior to this, leading zeroes would be stripped prior to writing the cell content. If the column being written to the Excel file has a mix of numbers with and without leading zeroes, those with leading zeroes will be left justified in the column as text, while the others will be right justified as numeric.

UTF-8 encoded content in the data frame should be properly exported by default. If you are operating in a 'latin1' based locale (also known as iso-8859-1) or a Windows CP-1252 locale, set Encoding to 'latin1' or 'cp1252', respectively.

As of version 5.0.0, Unicode character based content is better supported if running on a Windows based operating system. This is achieved by generating the intermediate CSV files using `writeln(..., useBytes = TRUE)`, which should avoid the re-encoding of the content of the CSV files into the operating locale of the relevant computer. Set Encoding to 'UTF-8' in this case to preserve the Unicode content in the Excel file. Since Encoding applies to all data frames and columns therein being exported to the same Excel file, if mixed character encodings are present, this may cause errors in the character representations in the Excel file. If you note errors from Perl regarding characters not mapping to Unicode, try setting Encoding to 'latin1'. If mixed encodings are present, you may want to explore the use of `iconv` to standardize the data to a single encoding.

If one or more of the data frame columns have been assigned a *comment* attribute using the `comment` function, these will be used to create a worksheet cell comment in this first row of the worksheet for each column with this attribute. These can serve to provide descriptive information for each column. This will work for both the Excel 2003 and 2007 file formats.

Note that arguments `AdjWidth`, `AutoFilter`, `BoldHeaderRow`, `FreezeRow` and `FreezeCol` will apply to ALL worksheets exported.

Value

TRUE if the Excel file was successfully created. FALSE if any errors occurred.

Note

Please be sure to read the included INSTALL file (in the main package installation directory) for additional details on meeting the requirements for Perl and the additional Perl modules that are necessary for this function to work properly. The file includes platform specific recommendations for common scenarios. The path to the package installation directory can be located using `system.file(package = "WriteXLS")`.

A working installed version of Perl must be present in the current system searchpath or the exact path of the perl executable must be provided via the `perl` argument. Perl modules required for this function that may not be part of a default Perl installation are included with this package. These modules include:

Archive::Zip, OLE::Storage_Lite, Parse::RecDescent, Spreadsheet::WriteExcel, Excel::Writer::XLSX and Text::CSV_PP

To test your Perl installation and verify that all required Perl modules are available, use the `testPerl` function provided in this package.

Author(s)

Marc Schwartz <marc_schwartz@me.com>

Many thanks to Prof. Brian Ripley for his assistance in the testing of this package.

References

Spreadsheet::WriteExcel Perl Module <http://search.cpan.org/dist/Spreadsheet-WriteExcel>

Excel::Writer::XLSX Perl Module <http://search.cpan.org/~jmcnamara/Excel-Writer-XLSX/>

Excel 2007 Specifications and Limitations <http://office.microsoft.com/en-us/excel-help/excel-specifications-and-limits-HP010073849.aspx>

For Perl Unicode Issues <http://www.ahinea.com/en/tech/perl-unicode-struggle.html> and/or <http://www.perl.com/pub/2012/04/perlunicook-standard-preamble.html>

See Also

[writeLines](#), [testPerl](#), [comment](#) and [iconv](#)

Examples

```
## Only run the examples if Perl and all modules are present
if (testPerl(verbose = FALSE))
{
  ## Examples using built-in data frames
  ## Create XLS (Excel 2003) files
  WriteXLS("iris", "iris.xls")

  WriteXLS(c("iris", "infert", "esoph"), "Example.xls")
}
```

```
iris.split <- split(iris, iris$Species)
WriteXLS("iris.split", "irissplit.xls")

## Example using comment()
## Commented cells will have a small red triangle in the
## upper right hand corner of the cell. Click on the cell
## or place the cursor over the cell to see the pop-up
## containing the comment text.
## Create an XLSX (Excel 2007) file
## Adjust the column widths
## Bold the header row
comment(iris$Sepal.Length) <- "Length of the sepals (cm)"
comment(iris$Sepal.Width) <- "Width of the sepals (cm)"
comment(iris$Petal.Length) <- "Length of the petals (cm)"
comment(iris$Petal.Width) <- "Width of the petals (cm)"
comment(iris$Species) <- "Species of the flowers"
WriteXLS("iris", "iriscomments.xlsx",
        AdjWidth = TRUE, BoldHeaderRow = TRUE)

## Add row names
WriteXLS("iris", "irisrownames.xlsx",
        AdjWidth = TRUE, BoldHeaderRow = TRUE, row.names = TRUE)

## Use latin1 Encoding
WriteXLS("iris", "irisLatin1.xls", Encoding = "latin1")

## Write a 0 row data frame
## Worksheet will contain header row only
DF0 <- data.frame("A" = numeric(), "B" = numeric(), "C" = numeric())
WriteXLS("DF0", "DF0.xls", AdjWidth = TRUE, BoldHeaderRow = TRUE)

## 'x' is a single data frame object
WriteXLS(iris, "irisDF.xls")

## 'x' is a list object containing data frames
WriteXLS(iris.split, "irisList.xls")

## Clean up and delete XLS files
rm(iris.split)
rm(DF0)
unlink("iris.xls")
unlink("Example.xls")
unlink("irissplit.xls")
unlink("iriscomments.xlsx")
unlink("irisrownames.xlsx")
unlink("irisLatin1.xls")
unlink("DF0.xls")
unlink("irisDF.xls")
unlink("irisList.xls")
}
```

Index

*Topic **file**

testPerl, [2](#)

WriteXLS, [3](#)

comment, [6](#)

iconv, [6](#)

testPerl, [2](#), [6](#)

writeLines, [6](#)

WriteXLS, [3](#), [3](#)