Package ‘openxlsx’

March 23, 2017

Type Package
Title Read, Write and Edit XLSX Files
Version 4.0.17
Date 2017-03-09
URL https://github.com/awalker89/openxlsx
BugReports https://github.com/awalker89/openxlsx/issues
Maintainer Alexander Walker <Alexander.Walker189@gmail.com>
Description Simplifies the creation of Excel .xlsx files by providing a high level interface to writing, styling and editing worksheets. Through the use of 'Rcpp', read/write times are comparable to the 'xlsx' and 'XLConnect' packages with the added benefit of removing the dependency on Java.
License GPL-3
LinkingTo Rcpp
Depends R (>= 3.3.0)
Imports methods, Rcpp, grDevices, stats, utils
VignetteBuilder knitr
Suggests knitr, testthat
RoxygenNote 6.0.1.9000
Collate 'CommentClass.R' 'HyperlinkClass.R' 'RcppExports.R'
'classDefinitions.R' 'StyleClass.R' 'WorkbookClass.R'
'baseXML.R' 'borderFunctions.R' 'chartsheet_class.R'
'conditional_formatting.R' 'helperFunctions.R' 'loadWorkbook.R'
'openXL.R' 'openxlsx.R' 'openxlsxCoerce.R' 'readWorkbook.R'
'sheet_data_class.R' 'workbook_column_widths.R'
'workbook_read_workbook.R' 'workbook_write_data.R'
'worksheet_class.R' 'wrappers.R' 'writeData.R'
'writeDataTable.R' 'writexlsx.R'
NeedsCompilation yes
Author Alexander Walker [aut, cre],
Luca Braglia [ctb]
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>addFilter</td>
</tr>
<tr>
<td>addStyle</td>
</tr>
<tr>
<td>addWorksheet</td>
</tr>
<tr>
<td>all.equal</td>
</tr>
<tr>
<td>conditionalFormat</td>
</tr>
<tr>
<td>conditionalFormatting</td>
</tr>
<tr>
<td>convertFromExcelRef</td>
</tr>
<tr>
<td>convertToDate</td>
</tr>
<tr>
<td>convertToDateTime</td>
</tr>
<tr>
<td>copyWorkbook</td>
</tr>
<tr>
<td>createComment</td>
</tr>
<tr>
<td>createNamedRegion</td>
</tr>
<tr>
<td>createStyle</td>
</tr>
<tr>
<td>createWorkbook</td>
</tr>
<tr>
<td>dataValidation</td>
</tr>
<tr>
<td>deleteData</td>
</tr>
<tr>
<td>freezePane</td>
</tr>
<tr>
<td>getBaseFont</td>
</tr>
<tr>
<td>getCellRefs</td>
</tr>
<tr>
<td>getDateOrigin</td>
</tr>
<tr>
<td>getNamedRegions</td>
</tr>
<tr>
<td>getSheetNames</td>
</tr>
<tr>
<td>getStyles</td>
</tr>
<tr>
<td>getTables</td>
</tr>
<tr>
<td>insertImage</td>
</tr>
<tr>
<td>insertPlot</td>
</tr>
<tr>
<td>int2col</td>
</tr>
<tr>
<td>loadWorkbook</td>
</tr>
<tr>
<td>makeHyperlinkString</td>
</tr>
<tr>
<td>mergeCells</td>
</tr>
<tr>
<td>modifyBaseFont</td>
</tr>
<tr>
<td>names</td>
</tr>
<tr>
<td>openXL</td>
</tr>
<tr>
<td>openxlsx</td>
</tr>
<tr>
<td>pageBreak</td>
</tr>
<tr>
<td>pageSetup</td>
</tr>
<tr>
<td>read.xlsx</td>
</tr>
<tr>
<td>readWorkbook</td>
</tr>
<tr>
<td>removeCellMerge</td>
</tr>
<tr>
<td>removeColWidths</td>
</tr>
<tr>
<td>removeComment</td>
</tr>
<tr>
<td>removeFilter</td>
</tr>
</tbody>
</table>
Description

Add excel column filters to a worksheet

Usage

addFilter(wb, sheet, rows, cols)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wb</td>
<td>A workbook object</td>
</tr>
<tr>
<td>sheet</td>
<td>A name or index of a worksheet</td>
</tr>
<tr>
<td>rows</td>
<td>A row number.</td>
</tr>
<tr>
<td>cols</td>
<td>columns to add filter to.</td>
</tr>
</tbody>
</table>

Details

adds filters to worksheet columns, same as filter parameters in writeData. writeDataTable automatically adds filters to first row of a table. NOTE Can only have a single filter per worksheet unless using tables.
addStyle

Add a style to a set of cells

Description
Function adds a style to a specified set of cells.

Usage
addStyle(wb, sheet, style, rows, cols, gridExpand = FALSE, stack = FALSE)

Arguments
- wb: A Workbook object containing a worksheet.
- sheet: A worksheet to apply the style to.
- style: A style object returned from createStyle()
- rows: Rows to apply style to.
- cols: Columns to apply style to.
- gridExpand: If TRUE, style will be applied to all combinations of rows and cols.
- stack: If TRUE the new style is merged with any existing cell styles. If FALSE, any existing style is replaced by the new style.

Author(s)
Alexander Walker
addWorksheet

See Also

createStyle
expand.grid

Examples

## See package vignette for more examples.

## Create a new workbook
wb <- createWorkbook("My name here")

## Add a worksheets
addWorksheet(wb, "Expenditure", gridLines = FALSE)

## write data to worksheet 1
writeData(wb, sheet = 1, USPersonalExpenditure, rowNames = TRUE)

## create and add a style to the column headers
headerStyle <- createStyle(fontSize = 14, fontColour = "#FFFFFF", halign = "center",
                          fgFill = "#4F81BD", border="TopBottom", borderColour = "#4F81BD")
addStyle(wb, sheet = 1, headerStyle, rows = 1, cols = 1:6, gridExpand = TRUE)

## style for body
bodyStyle <- createStyle(border="TopBottom", borderColour = "#4F81BD")
addStyle(wb, sheet = 1, bodyStyle, rows = 2:6, cols = 1:6, gridExpand = TRUE)
setColWidths(wb, 1, cols=1, widths = 21) ## set column width for row names column

saveWorkbook(wb, "addStyleExample.xlsx", overwrite = TRUE)
Arguments

wb  A Workbook object to attach the new worksheet  
sheetName  A name for the new worksheet  
gridLines  A logical. If FALSE, the worksheet grid lines will be hidden.  
tabColour  Colour of the worksheet tab. A valid colour (belonging to colours()) or a valid hex colour beginning with "#"  
zoom  A numeric between 10 and 400. Worksheet zoom level as a percentage.  
header  document header. Character vector of length 3 corresponding to positions left, center, right. Use NA to skip a position.  
footer  document footer. Character vector of length 3 corresponding to positions left, center, right. Use NA to skip a position.  
evenHeader  document header for even pages.  
evenFooter  document footer for even pages.  
firstHeader  document header for first page only.  
firstFooter  document footer for first page only.  
visible  If FALSE, sheet is hidden else visible.  
paperSize  An integer corresponding to a paper size. See ?pageSetup for details.  
orientation  One of "portrait" or "landscape"  
vdpi  Vertical DPI. Can be set with options("openxlsx.dpi" = X) or options("openxlsx.vdpi" = X)  
hdpi  Horizontal DPI. Can be set with options("openxlsx.dpi" = X) or options("openxlsx.hdpi" = X)

Details

Headers and footers can contain special tags

- &[Page] Page number
- &[Pages] Number of pages
- &[Date] Current date
- &[Time] Current time
- &[Path] File path
- &[File] File name
- &[Tab] Worksheet name

Value

XML tree

Author(s)

Alexander Walker
## Examples

```r
## Create a new workbook
wb <- createWorkbook("Fred")

## Add 3 worksheets
addWorksheet(wb, "Sheet 1")
addWorksheet(wb, "Sheet 2", gridLines = FALSE)
addWorksheet(wb, "Sheet 3", tabColour = "red")
addWorksheet(wb, "Sheet 4", gridLines = FALSE, tabColour = "#4F81BD")

## Headers and Footers
addWorksheet(wb, "Sheet 5",
  header = c("ODD HEAD LEFT", "ODD HEAD CENTER", "ODD HEAD RIGHT"),
  footer = c("ODD FOOT RIGHT", "ODD FOOT CENTER", "ODD FOOT RIGHT"),
  evenHeader = c("EVEN HEAD LEFT", "EVEN HEAD CENTER", "EVEN HEAD RIGHT"),
  evenFooter = c("EVEN FOOT RIGHT", "EVEN FOOT CENTER", "EVEN FOOT RIGHT"),
  firstHeader = c("TOP", "OF FIRST", "PAGE"),
  firstFooter = c("BOTTOM", "OF FIRST", "PAGE"))

addWorksheet(wb, "Sheet 6",
  header = c("[Date]", "ALL HEAD CENTER 2", "[Page] / [Pages]")
  footer = c("[Date] [Path][File]", NA, "[Tab]")
  firstHeader = c(NA, "Center Header of First Page", NA)
  firstFooter = c(NA, "Center Footer of First Page", NA))

addWorksheet(wb, "Sheet 7",
  header = c("ALL HEAD LEFT 2", "ALL HEAD CENTER 2", "ALL HEAD RIGHT 2")
  footer = c("ALL FOOT RIGHT 2", "ALL FOOT CENTER 2", "ALL FOOT RIGHT 2")
)

addWorksheet(wb, "Sheet 8",
  firstHeader = c("FIRST ONLY L", NA, "FIRST ONLY R")
  firstFooter = c("FIRST ONLY L", NA, "FIRST ONLY R")
)

## Need data on worksheet to see all headers and footers
writeData(wb, sheet = 5, 1:400)
writeData(wb, sheet = 6, 1:400)
writeData(wb, sheet = 7, 1:400)
writeData(wb, sheet = 8, 1:400)

## Save workbook
saveWorkbook(wb, "addWorksheetExample.xlsx", overwrite = TRUE)
```

## Description

Check equality of workbooks
Usage

```r
## S3 method for class 'Workbook'
all.equal(target, current, ...)
```

Arguments

- `target`: A Workbook object
- `current`: A Workbook object
- `...`: ignored

---

**conditionalFormat**  Add conditional formatting to cells

Description

DEPRECATED! USE `conditionalformatting`

Usage

```r
conditionalFormat(wb, sheet, cols, rows, rule = NULL, style = NULL,
                 type = "expression")
```

Arguments

- `wb`: A workbook object
- `sheet`: A name or index of a worksheet
- `cols`: Columns to apply conditional formatting to
- `rows`: Rows to apply conditional formatting to
- `rule`: The condition under which to apply the formatting or a vector of colours. See examples.
- `style`: A style to apply to those cells that satisfy the rule. A Style object returned from `createStyle()`
- `type`: Either 'expression', 'colorscale' or 'databar'. If 'expression' the formatting is determined by a formula. If colorScale cells are coloured based on cell value. See examples.

Details

DEPRECATED! USE `conditionalformatting`

Valid operators are "<", "<=", ">", ">=", "==", "!=". See Examples. Default style given by: `createStyle(fontColour = "#9C0006", bgFill = "#FFC7CE")`

Author(s)

Alexander Walker
conditionalFormatting

See Also
createStyle

conditionalFormatting  Add conditional formatting to cells

Description
Add conditional formatting to cells

Usage
conditionalFormatting(wb, sheet, cols, rows, rule = NULL, style = NULL, type = "expression", ...)

Arguments
wb  A workbook object
sheet  A name or index of a worksheet
cols  Columns to apply conditional formatting to
rows  Rows to apply conditional formatting to
rule  The condition under which to apply the formatting. See examples.
style  A style to apply to those cells that satisfy the rule. Default is createStyle(fontColour = "#9C0006", bgFill = "#FFC7CE")
type  Either 'expression', 'colorscale', 'databar', 'duplicates' or 'contains' (case insensitive).
...  See below

Details
See Examples.
If type == "expression"
  • style is a Style object. See createStyle
  • rule is an expression. Valid operators are "<", "<=", ">", ">=", "==", "!=".
If type == "colourScale"
  • style is a vector of colours with length 2 or 3
  • rule can be NULL or a vector of colours of equal length to styles
If type == "databar"
  • style is a vector of colours with length 2 or 3
• rule is a numeric vector specifying the range of the databar colours. Must be equal length to style
• ...
  – showvalue If FALSE the cell value is hidden. Default TRUE.
  – gradient If FALSE colour gradient is removed. Default TRUE.
  – border If FALSE the border around the database is hidden. Default TRUE.

If type == "duplicates"

• style is a Style object. See createStyle
• rule is ignored.

If type == "contains"

• style is a Style object. See createStyle
• rule is the text to look for within cells

If type == "between"

• style is a Style object. See createStyle
• rule is a numeric vector of length 2 specifying lower and upper bound (Inclusive)

Author(s)

Alexander Walker

See Also

createStyle

Examples

wb <- createWorkbook()
addWorksheet(wb, "cellIs")
addWorksheet(wb, "Moving Row")
addWorksheet(wb, "Moving Col")
addWorksheet(wb, "Dependent on 1")
addWorksheet(wb, "Duplicates")
addWorksheet(wb, "containsText")
addWorksheet(wb, "colourScale", zoom = 30)
addWorksheet(wb, "databar")
addWorksheet(wb, "between")
addWorksheet(wb, "logical operators")

negStyle <- createStyle(fontColour = "#9C0006", bgFill = "#FFC7CE")
posStyle <- createStyle(fontColour = "#006100", bgFill = "#C6EFCE")

## rule applies to all each cell in range
writeData(wb, "cellIs", -5:5)
writeData(wb, "cellIs", LETTERS[1:11], startCol=2)
conditionalFormatting(wb, "cellIs", cols=1, rows=1:11, rule="!=\0", style = negStyle)
conditionalFormatting(wb, "cellIs", cols=1, rows=1:11, rule="==0", style = posStyle)

## highlight row dependent on first cell in row
writeData(wb, "Moving Row", -5:5)
writeData(wb, "Moving Row", LETTERS[1:11], startCol=2)
conditionalFormatting(wb, "Moving Row", cols=1:2, rows=1:11, rule="$A1<0", style = negStyle)
conditionalFormatting(wb, "Moving Row", cols=1:2, rows=1:11, rule="$A1>0", style = posStyle)

## highlight column dependent on first cell in column
writeData(wb, "Moving Col", -5:5)
writeData(wb, "Moving Col", LETTERS[1:11], startCol=2)
conditionalFormatting(wb, "Moving Col", cols=1:2, rows=1:11, rule="A$1<0", style = negStyle)
conditionalFormatting(wb, "Moving Col", cols=1:2, rows=1:11, rule="A$1>0", style = posStyle)

## highlight entire range cols X rows dependent only on cell A1
writeData(wb, "Dependent on 1", -5:5)
writeData(wb, "Dependent on 1", LETTERS[1:11], startCol=2)
conditionalFormatting(wb, "Dependent on 1", cols=1:2, rows=1:11, rule="$A1<0", style = negStyle)
conditionalFormatting(wb, "Dependent on 1", cols=1:2, rows=1:11, rule="$A$1>0", style = posStyle)

## highlight duplicates using default style
writeData(wb, "Duplicates", sample(LETTERS[1:15]), size = 10, replace = TRUE)
conditionalFormatting(wb, "Duplicates", cols = 1, rows = 1:10, type = "duplicates")

## cells containing text
fn <- function(x) paste(sample(LETTERS, 10), collapse = "-")
writeData(wb, "containsText", sapply(1:10, fn))
conditionalFormatting(wb, "containsText", cols = 1, rows = 1:10, type = "contains", rule = "A")

## coloaur scale colours cells based on cell value
df <- read.xlsx(system.file("readTest.xlsx", package = "openxlsx"), sheet = 4)
writeData(wb, "colourScale", df, colNames=FALSE)  ## write data.frame

## rule is a vector or colours of length 2 or 3 (any hex colour or any of colours())
## If rule is NULL, min and max of cells is used. Rule must be the same length as style or NULL.
conditionalFormatting(wb, "colourScale", cols=1:ncol(df), rows=1:nrow(df),
style = c("black", "white"),
rule = c(0, 255),
type = "colourScale")
setColWidths(wb, "colourScale", cols = 1:ncol(df), widths = 1.07)
setRowHeights(wb, "colourScale", rows = 1:nrow(df), heights = 7.5)

## Databars
writeData(wb, "databar", -5:5)
conditionalFormatting(wb, "databar", cols = 1, rows = 1:11, type = "databar")  ## Default colours

## Between
## Highlight cells in interval [-2, 2]
writeData(wb, "between", -5:5)
conditionalFormatting(wb, "between", cols = 1, rows = 1:11, type = "between", rule = c(-2,2))

## Logical Operators
# You can use Excel's logical operators
writeData(wb, "logical operators", 1:10)
conditionalFormatting(wb, "logical operators", cols = 1, rows = 1:10,
rule = "OR($A1=1,$A1=3,$A1=5,$A1=7)"
)
saveWorkbook(wb, "conditionalFormattingExample.xlsx", TRUE)

####################################################################
## Databar Example
wb <- createWorkbook()
addWorksheet(wb, "databar")

## Databars
writeData(wb, "databar", -5:5, startCol = 1)
conditionalFormatting(wb, "databar", cols = 1, rows = 1:11, type = "databar") ## Defaults
writeData(wb, "databar", -5:5, startCol = 3)
conditionalFormatting(wb, "databar", cols = 3, rows = 1:11, type = "databar", border = FALSE)
writeData(wb, "databar", -5:5, startCol = 5)
conditionalFormatting(wb, "databar", cols = 5, rows = 1:11,
type = "databar", style = c("#a6a6a6"), showValue = FALSE)
writeData(wb, "databar", -5:5, startCol = 7)
conditionalFormatting(wb, "databar", cols = 7, rows = 1:11,
type = "databar", style = c("#a6a6a6"), showValue = FALSE, gradient = FALSE)
writeData(wb, "databar", -5:5, startCol = 9)
conditionalFormatting(wb, "databar", cols = 9, rows = 1:11,
type = "databar", style = c("#a6a6a6", ",a6a6a6"), showValue = FALSE, gradient = FALSE)
saveWorkbook(wb, file = "databarExample.xlsx", overwrite = TRUE)

convertFromExcelRef Convert excel column name to integer index

Description
Convert excel column name to integer index e.g. "J" to 10

Usage
convertFromExcelRef(col)

Arguments

| col | An excel column reference |
convertToDate

Examples

convertFromExcelRef("DOG")
convertFromExcelRef("COW")

## numbers will be removed
convertFromExcelRef("R22")

convertToDate(x, origin = "1900-01-01", ...)

Arguments

x A vector of integers
origin date. Default value is for Windows Excel 2010
... additional parameters passed to as.Date()

Details

Excel stores dates as number of days from some origin day

See Also

writeData

Examples

##2014 April 21st to 25th
convertToDate(c(41750, 41751, 41752, 41753, 41754, NA) )
convertToDate(c(41750.2, 41751.99, NA, 41753 ))
**convertToDateTIme**  
*Convert from excel time number to R POSIXct type.*

**Description**  
Convert from excel time number to R POSIXct type.

**Usage**  
```
convertToDateTIme(x, origin = "1900-01-01", ...)
```

**Arguments**

- `x`: A numeric vector
- `origin`: date. Default value is for Windows Excel 2010
- `...`: Additional parameters passed to `as.POSIXct`

**Details**  
Excel stores dates as number of days from some origin date

**Examples**
```
x <- c(41821.8127314815, 41820.8127314815, NA, 41819, NaN)
convertToDateTIme(x)
convertToDateTIme(x, tx = "Australia/Perth")
```

---

**copyWorkbook**  
*Copy a Workbook object.*

**Description**  
Just a wrapper of `wb$copy()`

**Usage**  
```
copyWorkbook(wb)
```

**Arguments**

- `wb`: A workbook object

**Value**  
Workbook
createComment

Examples

```r
wb <- createWorkbook()
wb2 <- wb # does not create a copy
wb3 <- copyWorkbook(wb) # wrapper for wb$copy()

addWorksheet(wb, "Sheet1") # adds worksheet to both wb and wb2 but not wb3

names(wb)
names(wb2)
names(wb3)
```

creatComment | create a Comment object

Description
Create a cell Comment object to pass to writeComment()

Usage
createComment(comment, author = Sys.getenv("USERNAME"), style = NULL,
visible = TRUE, width = 2, height = 4)

Arguments
- **comment**: Comment text. Character vector.
- **author**: Author of comment. Character vector of length 1
- **style**: A Style object or list of style objects the same length as comment vector. See createStyle.
- **visible**: TRUE or FALSE. Is comment visible.
- **width**: Textbox integer width in number of cells
- **height**: Textbox integer height in number of cells

See Also
writeComment

Examples
```r
wb <- createWorkbook()
addWorksheet(wb, "Sheet 1")

c1 <- createComment(comment = "this is comment")
writeComment(wb, 1, col = "B", row = 10, comment = c1)
```
createNamedRegion

Description

Create a named region.

Usage

createNamedRegion(wb, sheet, cols, rows, name)

Arguments

- **wb**: A workbook object
- **sheet**: A name or index of a worksheet
- **cols**: Numeric vector specifying columns to include in region
- **rows**: Numeric vector specifying rows to include in region
- **name**: Name for region. A character vector of length 1. Note region names must be case-insensitive unique.

Details

Region is given by: min(cols):max(cols) X min(rows):max(rows)

Author(s)

Alexander Walker

See Also

getNamedRegions
createStyle

Examples

```r
## create named regions
wb <- createWorkbook()
addWorksheet(wb, "Sheet 1")

## specify region
writeData(wb, sheet = 1, x = iris, startCol = 1, startRow = 1)
createNamedRegion(wb = wb,
    sheet = 1,
    name = "iris",
    rows = 1:(nrow(iris)+1),
    cols = 1:ncol(iris))

## using writeData 'name' argument
writeData(wb, sheet = 1, x = iris, name = "iris2", startCol = 10)

out_file <- tempfile(fileext = ".xlsx")
saveWorkbook(wb, out_file, overwrite = TRUE)

## see named regions
getNamedRegions(wb) ## From Workbook object
getNamedRegions(out_file) ## From xlsx file

## read named regions
df <- read.xlsx(wb, namedRegion = "iris")
head(df)

df <- read.xlsx(out_file, namedRegion = "iris2")
head(df)
```

createStyle  Create a cell style

Description

Create a new style to apply to worksheet cells

Usage

```r
createStyle(fontName = NULL, fontSize = NULL, fontColour = NULL, numFmt = "GENERAL", border = NULL, borderColour =getOption("openxlsx.borderColour", "black"),
    borderStyle = getOption("openxlsx.borderStyle", "thin"), bgFill = NULL, fgFill = NULL, halign = NULL, valign = NULL, textDecoration = NULL, wrapText = FALSE, textRotation = NULL, indent = NULL)
```
Arguments

fontName  A name of a font. Note the font name is not validated. If fontName is NULL, the workbook base font is used. (Defaults to Calibri)

fontSize  Font size. A numeric greater than 0. If fontSize is NULL, the workbook base font size is used. (Defaults to 11)

fontColour  Colour of text in cell. A valid hex colour beginning with "#" or one of colours(). If fontColour is NULL, the workbook base font colours is used. (Defaults to black)

numFmt  Cell formatting

• GENERAL
• NUMBER
• CURRENCY
• ACCOUNTING
• DATE
• LONGDATE
• TIME
• PERCENTAGE
• FRACTION
• SCIENTIFIC
• TEXT
• COMMA for comma seperated thousands
• For date/datetime styling a combination of d, m, y and punctuation marks
• For numeric rouding use "0.00" with the preferred number of decimal places

border  Cell border. A vector of "top", "bottom", "left", "right" or a single string).

• "top" Top border
• bottom Bottom border
• left Left border
• right Right border
• TopBottom or c("top", "bottom") Top and bottom border
• LeftRight or c("left", "right") Left and right border
• TopLeftRight or c("top", "left", "right") Top, Left and right border
• TopBottomLeftRight or c("top", "bottom", "left", "right") All borders

borderColour  Colour of cell border vector the same length as the number of sides specified in "border" A valid colour (belonging to colours()) or a valid hex colour beginning with "#"

borderStyle  Border line style vector the same length as the number of sides specified in "border"

• none No Border
• thin thin border
• medium medium border
• dashed dashed border
• **dotted** dotted border
• **thick** thick border
• **double** double line border
• **hair** Hairline border
• **mediumDashed** medium weight dashed border
• **dashDot** dash-dot border
• **mediumDashDot** medium weight dash-dot border
• **dashDotDot** dash-dot-dot border
• **mediumDashDotDot** medium weight dash-dot-dot border
• **slantDashDot** slanted dash-dot border

**bgFill** Cell background fill colour. A valid colour (belonging to colours()) or a valid hex colour beginning with "#". – **Use for conditional formatting styles only.**

**fgFill** Cell foreground fill colour. A valid colour (belonging to colours()) or a valid hex colour beginning with "#"

**halign** Horizontal alignment of cell contents
• **left** Left horizontal align cell contents
• **right** Right horizontal align cell contents
• **center** Center horizontal align cell contents

**valign** A name Vertical alignment of cell contents
• **top** Top vertical align cell contents
• **center** Center vertical align cell contents
• **bottom** Bottom vertical align cell contents

**textDecoration** Text styling.
• **bold** Bold cell contents
• **strikeout** Strikeout cell contents
• **italic** Italicise cell contents
• **underline** Underline cell contents
• **underline2** Double underline cell contents

**wrapText** Logical. If TRUE cell contents will wrap to fit in column.

**textRotation** Rotation of text in degrees. 255 for vertical text.

**indent** Horizontal indentation of cell contents.

**Value**
A style object

**Author(s)**
Alexander Walker

**See Also**
addStyle
createWorkbook

Create a new Workbook object

Description
Create a new Workbook object

Usage
createWorkbook(creator = Sys.getenv("USERNAME"))

Arguments
creator Creator of the workbook (your name). Defaults to login username

Value
Workbook object

Author(s)
Alexander Walker
dataValidation

See Also

loadWorkbook
saveWorkbook

Examples

```r
## Create a new workbook
wb <- createWorkbook()

## Save workbook to working directory
saveWorkbook(wb, file = "createWorkbookExample.xlsx", overwrite = TRUE)
```

---

dataValidation  Add data validation to cells

Description

Add Excel data validation to cells

Usage

```r
dataValidation(wb, sheet, cols, rows, type, operator, value,
allowBlank = TRUE, showInputMsg = TRUE, showErrorMsg = TRUE)
```

Arguments

- `wb`: A workbook object
- `sheet`: A name or index of a worksheet
- `cols`: Columns to apply conditional formatting to
- `rows`: Rows to apply conditional formatting to
- `type`: One of `‘whole’, ‘decimal’, ‘date’, ‘time’, ‘textLength’, ‘list’` (see examples)
- `value`: a vector of length 1 or 2 depending on operator (see examples)
- `allowBlank`: logical
- `showInputMsg`: logical
- `showErrorMsg`: logical
Examples

```r
wb <- createWorkbook()
addWorksheet(wb, "Sheet 1")
addWorksheet(wb, "Sheet 2")

writeDataTable(wb, 1, x = iris[1:30,])

dataValidation(wb, 1, col = c(1:3), rows = 2:31, type = "whole",
               operator = "between", value = c(1, 9))

dataValidation(wb, 1, col = 5, rows = 2:31, type = "textLength",
               operator = "between", value = c(4, 6))

## Date and Time cell validation
df <- data.frame(d = as.Date("2016-01-01") + 5:5,
                 "t" = as.POSIXct("2016-01-01") - 5:5*10000)

writeData(wb, 2, x = df)
dataValidation(wb, 2, col = c(1, 2), rows = 2:12, type = "date",
               operator = "greaterThanOrEqual", value = as.Date("2016-01-01"))
dataValidation(wb, 2, col = 2, rows = 2:12, type = "time",
               operator = "between", value = df$[c(4 , 8) ,])

saveWorkbook(wb, "dataValidationExample.xlsx", overwrite = TRUE)
```

```r
# If type == 'list'
# operator argument is ignored.
wb <- createWorkbook()
addWorksheet(wb, "Sheet 1")
addWorksheet(wb, "Sheet 2")

writeDataTable(wb, sheet = 1, x = iris[1:30,])
writeData(wb, sheet = 2, x = sample(iris$Sepal.Length, 10))
dataValidation(wb, 1, col = 1, rows = 2:31, type = "list", value = "'Sheet 2'!A$1:$A$10")

# openXL(wb)
```

deleteData Delete cell data

Description

Delete contents and styling from a cell.
freezePane

Usage

deedata(wb, sheet, cols, rows, gridExpand = FALSE)

Arguments

wb A workbook object
sheet A name or index of a worksheet
cols columns to delete data from.
rows Rows to delete data from.
gridExpand If TRUE, all data in rectangle min(rows):max(rows) X min(cols):max(cols) will be removed.

Author(s)

Alexander Walker

Examples

## write some data
wb <- createWorkbook()
addWorksheet(wb, "Worksheet 1")
x <- data.frame(matrix(runif(200), ncol = 10))
writeData(wb, sheet = 1, x = x, startCol = 2, startRow = 3, colNames = FALSE)

## delete some data
deletedata(wb, sheet = 1, cols = 3:5, rows = 5:7, gridExpand = TRUE)
deletedata(wb, sheet = 1, cols = 7:9, rows = 5:7, gridExpand = TRUE)
deletedata(wb, sheet = 1, cols = LETTERS, rows = 18, gridExpand = TRUE)

csaveWorkbook(wb, "deleteDataExample.xlsx", overwrite = TRUE)

freezePane Freeze a worksheet pane

Description

Freeze a worksheet pane

Usage

freezePane(wb, sheet, firstActiveRow = NULL, firstActiveCol = NULL,
firstRow = FALSE, firstCol = FALSE)
getBaseFont

Arguments

wb A workbook object
sheet A name or index of a worksheet
firstActiveRow Top row of active region
firstActiveCol Furthest left column of active region
firstRow If TRUE, freezes the first row (equivalent to firstActiveRow = 2)
firstCol If TRUE, freezes the first column (equivalent to firstActiveCol = 2)

Author(s)

Alexander Walker

Examples

## Create a new workbook
wb <- createWorkbook("Kenshin")

## Add some worksheets
addWorksheet(wb, "Sheet 1")
addWorksheet(wb, "Sheet 2")
addWorksheet(wb, "Sheet 3")
addWorksheet(wb, "Sheet 4")

## Freeze Panes
freezePane(wb, "Sheet 1", firstActiveRow = 5, firstActiveCol = 3)
freezePane(wb, "Sheet 2", firstCol = TRUE) ## shortcut to firstActiveCol = 2
freezePane(wb, 3, firstRow = TRUE) ## shortcut to firstActiveRow = 2
freezePane(wb, 4, firstActiveRow = 1, firstActiveCol = "D")

## Save workbook
saveWorkbook(wb, "freezePaneExample.xlsx", overwrite = TRUE)

getBaseFont Return the workbook default font

Description

Return the workbook default font

Returns the base font used in the workbook.

Usage

getBaseFont(wb)

Arguments

wb A workbook object
getCellRefs

Author(s)

Alexander Walker

Examples

```r
## create a workbook
wb <- createWorkbook()
getBaseFont(wb)

## modify base font to size 10 Arial Narrow in red
modifyBaseFont(wb, fontSize = 10, fontColour = "#FF0000", fontName = "Arial Narrow")
getBaseFont(wb)
```

getCellRefs  

*Return excel cell coordinates from (x,y) coordinates*

Description

Return excel cell coordinates from (x,y) coordinates

Usage

```r
getCellRefs(cellCoords)
```

Arguments

- `cellCoords` A data.frame with two columns coordinate pairs.

Value

Excel alphanumeric cell reference

Author(s)

Alexander Walker
get Date Origin

Get the date origin an xlsx file is using

Description
Return the date origin used internally by an xlsx or xlsm file

Usage
get Date Origin(xlsxFile)

Arguments
xlsxFile An xlsx or xlsm file.

Details
Excel stores dates as the number of days from either 1904-01-01 or 1900-01-01. This function checks the date origin being used in an Excel file and returns is so it can be used in convertToDate

Value
One of "1900-01-01" or "1904-01-01".

Author(s)
Alexander Walker

See Also
convertToDate

Examples

## create a file with some dates
write.xlsx(as.Date("2015-01-10") - (0:4), file = "getdateoriginExample.xlsx")
m <- read.xlsx("getdateoriginExample.xlsx")

## convert to dates
do <- getDateOrigin(system.file("readTest.xlsx", package = "openxlsx"))
convertToDate(m[[1]], do)
getNamedRegions

Get named regions

Description

Return a vector of named regions in a xlsx file or Workbook object

Usage

getNamedRegions(x)

Arguments

x
An xlsx file or Workbook object

See Also

createNamedRegion

Examples

```r
## create named regions
wb <- createWorkbook()
addWorksheet(wb, "Sheet 1")

## specify region
writeData(wb, sheet = 1, x = iris, startCol = 1, startRow = 1)
createNamedRegion(wb = wb, 
               sheet = 1,
               name = "iris",
               rows = 1:(nrow(iris)+1),
               cols = 1:ncol(iris))

## using writeData 'name' argument to create a named region
writeData(wb, sheet = 1, x = iris, name = "iris2", startCol = 10)

out_file <- tempfile(fileext = ".xlsx")
saveWorkbook(wb, out_file, overwrite = TRUE)

## see named regions
getNamedRegions(wb) ## From Workbook object
gUnnamedRegions(out_file) ## From xlsx file

## read named regions
df <- read.xlsx(wb, namedRegion = "iris")
head(df)

df <- read.xlsx(out_file, namedRegion = "iris2")
head(df)
```
**getSheetNames**  
*Get names of worksheets*

**Description**

Returns the worksheet names within an xlsx file.

**Usage**

gSheetNames(file)

**Arguments**

- **file**
  An xlsx or xlam file.

**Value**

Character vector of worksheet names.

**Author(s)**

Alexander Walker

**Examples**

gSheetNames(system.file("readTest.xlsx", package = "openxlsx"))

---

**getStyles**  
*Returns a list of all styles in the workbook*

**Description**

Returns list of style objects in the workbook.

**Usage**

gStyles(wb)

**Arguments**

- **wb**
  A workbook object

**See Also**

replaceStyle
getTables

Examples

```r
## load a workbook
wb <- loadWorkbook(file = system.file("loadExample.xlsx", package = "openxlsx"))
getStyles(wb)[1:3]
```

---

getTables  

*List Excel tables in a workbook*

Description

List Excel tables in a workbook

Usage

```r
getTables(wb, sheet)
```

Arguments

- `wb`  
  A workbook object
- `sheet`  
  A name or index of a worksheet

Value

character vector of table names on the specified sheet

Examples

```r
wb <- createWorkbook()
addWorksheet(wb, sheetName = "Sheet 1")
writeDataTable(wb, sheet = "Sheet 1", x = iris)
writeDataTable(wb, sheet = 1, x = mtcars, tableName = "mtcars", startCol = 10)

getTables(wb, sheet = "Sheet 1")
```
**insertImage**  
*Insert an image into a worksheet*

**Description**

Insert an image into a worksheet

**Usage**

```r
insertImage(wb, sheet, file, width = 6, height = 3, startRow = 1, startCol = 1, units = "in", dpi = 300)
```

**Arguments**

- `wb`: A workbook object
- `sheet`: A name or index of a worksheet
- `file`: An image file. Valid file types are: jpeg, png, bmp
- `width`: Width of figure.
- `height`: Height of figure.
- `startRow`: Row coordinate of upper left corner of the image
- `startCol`: Column coordinate of upper left corner of the image
- `units`: Units of width and height. Can be "in", "cm" or "px"
- `dpi`: Image resolution used for conversion between units.

**Author(s)**

Alexander Walker

**See Also**

`insertPlot`

**Examples**

```r
## Create a new workbook
wb <- createWorkbook("Ayanami")

## Add some worksheets
addWorksheet(wb, "Sheet 1")
addWorksheet(wb, "Sheet 2")
addWorksheet(wb, "Sheet 3")

## Insert images
img <- system.file("einstein.jpg", package = "openxlsx")
insertImage(wb, "Sheet 1", img, startRow = 5, startCol = 3, width = 6, height = 5)
insertImage(wb, 2, img, startRow = 2, startCol = 2)
```
**insertPlot**

```
insertImage(wb, 3, img, width = 15, height = 12, startRow = 3, startCol = "G", units = "cm")
```

## Description

The current plot is saved to a temporary image file using dev.copy. This file is then written to the workbook using insertImage.

## Usage

```
insertPlot(wb, sheet, width = 6, height = 4, xy = NULL, startRow = 1,
startCol = 1, fileType = "png", units = "in", dpi = 300)
```

## Arguments

- **wb**  
  A workbook object
- **sheet**  
  A name or index of a worksheet
- **width**  
  Width of figure. Defaults to 6in.
- **height**  
  Height of figure. Defaults to 4in.
- **xy**  
  Alternate way to specify startRow and startCol. A vector of length 2 of form (startcol, startRow)
- **startRow**  
  Row coordinate of upper left corner of figure. xy[[2]] when xy is given.
- **startCol**  
  Column coordinate of upper left corner of figure. xy[[1]] when xy is given.
- **fileType**  
  File type of image
- **units**  
  Units of width and height. Can be "in", "cm" or "px"
- **dpi**  
  Image resolution

## Author(s)

Alexander Walker

## See Also

- **insertImage**
Examples

```r
## Not run:
## Create a new workbook
wb <- createWorkbook()

## Add a worksheet
addWorksheet(wb, "Sheet 1", gridLines = FALSE)

## create plot objects
require(ggplot2)
p1 <- qplot(mpg, data = mtcars, geom = "density",
fill = as.factor(gear), alpha = I(.5), main = "Distribution of Gas Mileage")
p2 <- qplot(age, circumference, data = Orange, geom = c("point", "line"), colour = Tree)

## Insert currently displayed plot to sheet 1, row 1, column 1
print(p1) # plot needs to be showing
insertPlot(wb, 1, width = 5, height = 3.5, fileType = "png", units = "in")

## Insert plot 2
print(p2)
insertPlot(wb, 1, xy = c("J", 2), width = 16, height = 10, fileype = "png", units = "cm")

## Save workbook
saveWorkbook(wb, "insertPlotExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

**int2col**

*Convert integer to Excel column*

**Description**

Converts an integer to an Excel column label.

**Usage**

```r
int2col(x)
```

**Arguments**

- `x` A numeric vector

**Examples**

```r
int2col(1:10)
```
loadWorkbook  

*Load an existing .xlsx file*

### Description

loadWorkbook returns a workbook object conserving styles and formatting of the original .xlsx file.

### Usage

```r
loadWorkbook(file, xlsxFile = NULL)
```

### Arguments

- **file**: A path to an existing .xlsx or .xlsm file
- **xlsxFile**: alias for file

### Value

Workbook object.

### Author(s)

Alexander Walker

### See Also

- `removeWorksheet`

### Examples

```r
## load existing workbook from package folder
wb <- loadWorkbook(file = system.file("loadExample.xlsx", package= "openxlsx"))
names(wb)  # list worksheets
wb ## view object
## Add a worksheet
addWorksheet(wb, "A new worksheet")

## Save workbook
saveWorkbook(wb, "loadExample.xlsx", overwrite = TRUE)
```
makeHyperlinkString  
create Excel hyperlink string

**Description**

Wrapper to create internal hyperlink string to pass to writeFormula()

**Usage**

```r
makeHyperlinkString(sheet, row = 1, col = 1, text = NULL, file = NULL)
```

**Arguments**

- **sheet**: Name of a worksheet
- **row**: integer row number for hyperlink to link to
- **col**: column number of letter for hyperlink to link to
- **text**: display text
- **file**: Excel file name to point to. If NULL hyperlink is internal.

**See Also**

- `writeFormula`

**Examples**

```r
## Writing internal hyperlinks
wb <- createWorkbook()
addWorksheet(wb, "Sheet1")
addWorksheet(wb, "Sheet2")
addWorksheet(wb, "Sheet 3")
writeData(wb, sheet = 3, x = iris)

## External Hyperlink
names(x) <- c("google", "google Aus")
class(x) <- "hyperlink"
writeData(wb, sheet = 1, x = x, startCol = 10)

## Internal Hyperlink - create hyperlink formula manually
writeFormula(wb, "Sheet1", x = '=HYPERLINK("#Sheet2!B3", "Text to Display - Link to Sheet2")', startCol = 3)

## Internal - No text to display using makeHyperlinkString() function
writeFormula(wb, "Sheet1", startRow = 1, x = makeHyperlinkString(sheet = "Sheet 3", row = 1, col = 2))
```
mergeCells

## Description
Merge cells within a worksheet

## Usage
mergeCells(wb, sheet, cols, rows)

## Arguments
- **wb**: A workbook object
- **sheet**: A name or index of a worksheet
- **cols**: Columns to merge
- **rows**: corresponding rows to merge

## Details
As merged region must be rectangular, only min and max of cols and rows are used.

## Author(s)
Alexander Walker

## See Also
removeCellMerge
modifyBaseFont

Modify the default font

Description

Modify the default font for this workbook

Usage

modifyBaseFont(wb, fontSize = 11, fontColour = "black",
              fontName = "Calibri")

Arguments

- wb: A workbook object
- fontSize: font size
- fontColour: font colour
- fontName: Name of a font
Details

The font name is not validated in any way. Excel replaces unknown font names with Arial. Base font is black, size 11, Calibri.

Author(s)

Alexander Walker

Examples

```r
## create a workbook
wb <- createWorkbook()
addWorksheet(wb, "S1")
## modify base font to size 10 Arial Narrow in red
modifyBaseFont(wb, fontSize = 10, fontColour = "#FF0000", fontName = "Arial Narrow")

writeData(wb, "S1", iris)
writeDataTable(wb, "S1", x = iris, startCol = 10) ## font colour does not affect tables
saveWorkbook(wb, "modifyBaseFontExample.xlsx", overwrite = TRUE)
```

get or set worksheet names

Description

get or set worksheet names

Usage

```r
## S3 method for class 'Workbook'
names(x)

## S3 replacement method for class 'Workbook'
names(x) <- value
```

Arguments

- `x`: A Workbook object
- `value`: a character vector the same length as `wb`

Examples

```r
wb <- createWorkbook()
addWorksheet(wb, "S1")
addWorksheet(wb, "S2")
addWorksheet(wb, "S3")
```
openXL

Open a Microsoft Excel file (xls/xlsx) or an openxlsx Workbook

Description

This function tries to open a Microsoft Excel (xls/xlsx) file or an openxlsx Workbook with the proper application, in a portable manner.

In Windows (c) and Mac (c), it uses system default handlers, given the file type.

In Linux it searches (via `which`) for available xls/xlsx reader applications (unless options('openxlsx.xlsxApp') is set to the app bin path), and if it finds anything, sets options('openxlsx.xlsxApp') to the program choosed by the user via a menu (if many are present, otherwise it will set the only available). Currently searched for apps are Libreoffice/Openoffice (soffice bin), Gnumeric (gnumeric) and Calligra Sheets (calligrasheets).

Usage

```r
openXL(file=NULL)
```

Arguments

- `file` path to the Excel (xls/xlsx) file or Workbook object.

Author(s)

Luca Braglia

Examples

```r
# file example
data <- NULL
example(writeData)
openXL("writeDataExample.xlsx")

# (not yet saved) Workbook example
wb <- createWorkbook()
x <- mtcars[1:6,]
addWorksheet(wb, "Cars")
writeData(wb, "Cars", x, startCol = 2, startRow = 3, rowNames = TRUE)
openXL(wb)
```
openxlsx  

**xlsx reading, writing and editing.**

**Description**

openxlsx simplifies the process of writing and styling Excel xlsx files from R and removes the dependency on Java.

**Details**

The openxlsx package uses global options to simplify formatting:

- `options("openxlsx.borderColour" = "black")`
- `options("openxlsx.borderStyle" = "thin")`
- `options("openxlsx.dateFormat" = "mm/dd/yyyy")`
- `options("openxlsx.datetimeFormat" = "yyyy-mm-dd hh:mm:ss")`
- `options("openxlsx.numFmt" = NULL)`
- `options("openxlsx.pageSize" = 9) # A4`
- `options("openxlsx.orientation" = "portrait") # page orientation`

See the Formatting vignette for examples.

**Additional options**

- `options("openxlsx.zipFlags" = "-9") # set max zip compression level default is "-1"

**See Also**

- `vignette("Introduction", package = "openxlsx")`
- `vignette("formatting", package = "openxlsx")`
- `writeData`
- `writeDataTable`
- `write.xlsx`
- `read.xlsx`

for examples
pageBreak

add a page break to a worksheet

Description

insert page breaks into a worksheet

Usage

pageBreak(wb, sheet, i, type = "row")

Arguments

wb A workbook object
sheet A name or index of a worksheet
i row or column number to insert page break.
type One of "row" or "column" for a row break or column break.

See Also

addWorksheet

Examples

wb <- createWorkbook()
addWorksheet(wb, "Sheet 1")
writeData(wb, sheet = 1, x = iris)

pageBreak(wb, sheet = 1, i = 10, type = "row")
pageBreak(wb, sheet = 1, i = 20, type = "row")
pageBreak(wb, sheet = 1, i = 2, type = "column")

saveWorkbook(wb, "pageBreakExample.xlsx", TRUE)
## In Excel: View tab -> Page Break Preview

pageSetup

Set page margins, orientation and print scaling

Description

Set page margins, orientation and print scaling
Usage

pageSetup(wb, sheet, orientation = NULL, scale = 100, left = 0.7, right = 0.7, top = 0.75, bottom = 0.75, header = 0.3, footer = 0.3, fitToWidth = FALSE, fitToHeight = FALSE, paperSize = NULL, printTitleRows = NULL, printTitleCols = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wb</td>
<td>A workbook object</td>
</tr>
<tr>
<td>sheet</td>
<td>A name or index of a worksheet</td>
</tr>
<tr>
<td>orientation</td>
<td>Page orientation. One of &quot;portrait&quot; or &quot;landscape&quot;</td>
</tr>
<tr>
<td>scale</td>
<td>Print scaling. Numeric value between 10 and 400</td>
</tr>
<tr>
<td>left</td>
<td>left page margin in inches</td>
</tr>
<tr>
<td>right</td>
<td>right page margin in inches</td>
</tr>
<tr>
<td>top</td>
<td>top page margin in inches</td>
</tr>
<tr>
<td>bottom</td>
<td>bottom page margin in inches</td>
</tr>
<tr>
<td>header</td>
<td>header margin in inches</td>
</tr>
<tr>
<td>footer</td>
<td>footer margin in inches</td>
</tr>
<tr>
<td>fitToWidth</td>
<td>If TRUE, worksheet is scaled to fit to page width on printing.</td>
</tr>
<tr>
<td>fitToHeight</td>
<td>If TRUE, worksheet is scaled to fit to page height on printing.</td>
</tr>
<tr>
<td>paperSize</td>
<td>See details. Default value is 9 (A4 paper).</td>
</tr>
<tr>
<td>printTitleRows</td>
<td>Rows to repeat at top of page when printing. Integer vector.</td>
</tr>
<tr>
<td>printTitleCols</td>
<td>Columns to repeat at left when printing. Integer vector.</td>
</tr>
</tbody>
</table>

Details

paperSize is an integer corresponding to:

- 1 Letter paper (8.5 in. by 11 in.)
- 2 Letter small paper (8.5 in. by 11 in.)
- 3 Tabloid paper (11 in. by 17 in.)
- 4 Ledger paper (17 in. by 11 in.)
- 5 Legal paper (8.5 in. by 14 in.)
- 6 Statement paper (5.5 in. by 8.5 in.)
- 7 Executive paper (7.25 in. by 10.5 in.)
- 8 A3 paper (297 mm by 420 mm)
- 9 A4 paper (210 mm by 297 mm)
- 10 A4 small paper (210 mm by 297 mm)
- 11 A5 paper (148 mm by 210 mm)
- 12 B4 paper (250 mm by 353 mm)
- 13 B5 paper (176 mm by 250 mm)
• 14 Folio paper (8.5 in. by 13 in.)
• 15 Quarto paper (215 mm by 275 mm)
• 16 Standard paper (10 in. by 14 in.)
• 17 Standard paper (11 in. by 17 in.)
• 18 Note paper (8.5 in. by 11 in.)
• 19 #9 envelope (3.875 in. by 8.875 in.)
• 20 #10 envelope (4.125 in. by 9.5 in.)
• 21 #11 envelope (4.5 in. by 10.375 in.)
• 22 #12 envelope (4.75 in. by 11 in.)
• 23 #14 envelope (5 in. by 11.5 in.)
• 24 C paper (17 in. by 22 in.)
• 25 D paper (22 in. by 34 in.)
• 26 E paper (34 in. by 44 in.)
• 27 DL envelope (110 mm by 220 mm)
• 28 C5 envelope (162 mm by 229 mm)
• 29 C3 envelope (324 mm by 458 mm)
• 30 C4 envelope (229 mm by 324 mm)
• 31 C6 envelope (114 mm by 162 mm)
• 32 C65 envelope (114 mm by 229 mm)
• 33 B4 envelope (250 mm by 353 mm)
• 34 B5 envelope (176 mm by 250 mm)
• 35 B6 envelope (176 mm by 125 mm)
• 36 Italy envelope (110 mm by 230 mm)
• 37 Monarch envelope (3.875 in. by 7.5 in.).
• 38 6 3/4 envelope (3.625 in. by 6.5 in.)
• 39 US standard fanfold (14.875 in. by 11 in.)
• 40 German standard fanfold (8.5 in. by 12 in.)
• 41 German legal fanfold (8.5 in. by 13 in.)
• 42 ISO B4 (250 mm by 353 mm)
• 43 Japanese double postcard (200 mm by 148 mm)
• 44 Standard paper (9 in. by 11 in.)
• 45 Standard paper (10 in. by 11 in.)
• 46 Standard paper (15 in. by 11 in.)
• 47 Invite envelope (220 mm by 220 mm)
• 48 Letter extra paper (9.275 in. by 12 in.)
• 51 Legal extra paper (9.275 in. by 15 in.)
• 52 Tabloid extra paper (11.69 in. by 18 in.)
• 53 A4 extra paper (236 mm by 322 mm)
• 54 Letter transverse paper (8.275 in. by 11 in.)
• 55 A4 transverse paper (210 mm by 297 mm)
• 56 Letter extra transverse paper (9.275 in. by 12 in.)
• 57 SuperA/SuperA/A4 paper (227 mm by 356 mm)
• 58 SuperB/SuperB/A3 paper (305 mm by 487 mm)
• 59 Letter plus paper (8.5 in. by 12.69 in.)
• 60 A4 plus paper (210 mm by 330 mm)
• 61 A5 transverse paper (148 mm by 210 mm)
• 62 JIS B5 transverse paper (182 mm by 257 mm)
• 63 A3 extra paper (322 mm by 445 mm)
• 64 A5 extra paper (174 mm by 235 mm)
• 65 ISO B5 extra paper (201 mm by 276 mm)
• 66 A2 paper (420 mm by 594 mm)
• 67 A3 transverse paper (297 mm by 420 mm)
• 68 A3 extra transverse paper (322 mm by 445 mm)

Author(s)
Alexander Walker

Examples

```r
wb <- createWorkbook()
addWorksheet(wb, "S1")
addWorksheet(wb, "S2")
writeDataTable(wb, 1, x = iris[1:30,])
writeDataTable(wb, 2, x = iris[1:30,], xy = c("C", 5))

## landscape page scaled to 50%
pageSetup(wb, sheet = 1, orientation = "landscape", scale = 50)

## portrait page scales to 300% with 0.5in left and right margins
pageSetup(wb, sheet = 2, orientation = "portrait", scale = 300, left= 0.5, right = 0.5)

## print titles
addWorksheet(wb, "print_title_rows")
addWorksheet(wb, "print_title_cols")
writeData(wb, "print_title_rows", rbind(iris, iris, iris, iris))
writeData(wb, "print_title_cols", x = rbind(mtcars, mtcars, mtcars), rowNames = TRUE)

pageSetup(wb, sheet = "print_title_rows", printTitleRows = 1) ## first row
pageSetup(wb, sheet = "print_title_cols", printTitleCols = 1, printTitleRows = 1)

saveWorkbook(wb, "pageSetupExample.xlsx", overwrite = TRUE)
```
**Description**
Read data from an Excel file or Workbook object into a data.frame

**Usage**
```
read.xlsx(xlsxFile, sheet = 1, startRow = 1, colNames = TRUE,
    rowNames = FALSE, detectDates = FALSE, skipEmptyRows = TRUE,
    skipEmptyCols = TRUE, rows = NULL, cols = NULL, check.names = FALSE,
    namedRegion = NULL, na.strings = "NA", fillMergedCells = FALSE)
```

**Arguments**
- `xlsxFile` An xlsx file, Workbook object or URL to xlsx file.
- `sheet` The name or index of the sheet to read data from.
- `startRow` first row to begin looking for data. Empty rows at the top of a file are always skipped, regardless of the value of startRow.
- `colNames` If `true`, the first row of data will be used as column names.
- `rowNames` If `true`, first column of data will be used as row names.
- `detectDates` If `true`, attempt to recognise dates and perform conversion.
- `skipEmptyRows` If `true`, empty rows are skipped else empty rows after the first row containing data will return a row of NAs.
- `skipEmptyCols` If `true`, empty columns are skipped.
- `rows` A numeric vector specifying which rows in the Excel file to read. If `NULL`, all rows are read.
- `cols` A numeric vector specifying which columns in the Excel file to read. If `NULL`, all columns are read.
- `check.names` logical. If `TRUE` then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names.
- `namedRegion` A named region in the Workbook. If not `NULL` startRow, rows and cols paramters are ignored.
- `na.strings` A character vector of strings which are to be interpreted as NA. Blank cells will be returned as NA.
- `fillMergedCells` If `true`, the value in a merged cell is given to all cells within the merge.

**Details**
Formulae written using `writeFormula` to a Workbook object will not get picked up by `read.xlsx()`. This is because only the formula is written and left to be evaluated when the file is opened in Excel. Opening, saving and closing the file with Excel will resolve this.
Value
data.frame

Author(s)
Alexander Walker

See Also
getNamedRegions

Examples

```r
xlsxFile <- system.file("readTest.xlsx", package = "openxlsx")
df1 <- read.xlsx(xlsxFile = xlsxFile, sheet = 1, skipEmptyRows = FALSE) sapply(df1, class)

df2 <- read.xlsx(xlsxFile = xlsxFile, sheet = 3, skipEmptyRows = TRUE) df2$Date <- convertToDate(df2$Date) sapply(df2, class) head(df2)

df2 <- read.xlsx(xlsxFile = xlsxFile, sheet = 3, skipEmptyRows = TRUE, detectDates = TRUE) sapply(df2, class) head(df2)

wb <- loadWorkbook(system.file("readTest.xlsx", package = "openxlsx"))
df3 <- read.xlsx(wb, sheet = 2, skipEmptyRows = FALSE, colName = TRUE) df4 <- read.xlsx(xlsxFile, sheet = 2, skipEmptyRows = FALSE, colName = TRUE) all.equal(df3, df4)

wb <- loadWorkbook(system.file("readTest.xlsx", package = "openxlsx")) df3 <- read.xlsx(wb, sheet = 2, skipEmptyRows = FALSE, cols = c(1, 4), rows = c(1, 3, 4))
```

## URL
##
xlsxFile <- "https://github.com/awalker89/openxlsx/raw/master/inst/readTest.xlsx"
#head(read.xlsx(xlsxFile))

---

**readWorkbook**

*Read from an Excel file or Workbook object*

**Description**

Read data from an Excel file or Workbook object into a data.frame
readWorkbook

Usage

readWorkbook(xlsxFile, sheet = 1, startRow = 1, colNames = TRUE,
rowNames = FALSE, detectDates = FALSE, skipEmptyRows = TRUE,
skipEmptyCols = TRUE, rows = NULL, cols = NULL, check.names = FALSE,
namedRegion = NULL, na.strings = "NA", fillMergedCells = FALSE)

Arguments

xlsxFile An xlsx file, Workbook object or URL to xlsx file.
sheet The name or index of the sheet to read data from.
startRow first row to begin looking for data. Empty rows at the top of a file are always
      skipped, regardless of the value of startRow.
colNames If TRUE, the first row of data will be used as column names.
rowNames If TRUE, first column of data will be used as row names.
detectDates If TRUE, attempt to recognise dates and perform conversion.
skipEmptyRows If TRUE, empty rows are skipped else empty rows after the first row containing
      data will return a row of NAs.
skipEmptyCols If TRUE, empty columns are skipped.
rows A numeric vector specifying which rows in the Excel file to read. If NULL, all
      rows are read.
cols A numeric vector specifying which columns in the Excel file to read. If NULL, all
      columns are read.
check.names logical. If TRUE then the names of the variables in the data frame are checked
to ensure that they are syntactically valid variable names
namedRegion A named region in the Workbook. If not NULL startRow, rows and cols paramters
      are ignored.
nan.strings A character vector of strings which are to be interpreted as NA. Blank cells will
      be returned as NA.
fillMergedCells If TRUE, the value in a merged cell is given to all cells within the merge.

Details

Creates a data.frame of all data in worksheet.

Value

data.frame

Author(s)

Alexander Walker
**removeCellMerge**

Create a new Workbook object

**Description**

Unmerges any merged cells that intersect with the region specified by, min(cols):max(cols) X min(rows):max(rows)

**Usage**

```r
removeCellMerge(wb, sheet, cols, rows)
```

**Arguments**

- `wb` A workbook object
- `sheet` A name or index of a worksheet
- `cols` vector of column indices
- `rows` vector of row indices

**Author(s)**

Alexander Walker

**See Also**

- `mergeCells`
- `getXSS`
removeColWidths  Remove column widths from a worksheet

Description
Remove column widths from a worksheet

Usage
removeColWidths(wb, sheet, cols)

Arguments
wb  A workbook object
    sheet  A name or index of a worksheet
    cols  Indices of columns to remove custom width (if any) from.

Author(s)
Alexander Walker

See Also
setColWidths

Examples
## Create a new workbook
wb <- loadWorkbook(file = system.file("loadExample.xlsx", package = "openxlsx"))

## remove column widths in columns 1 to 20
removeColWidths(wb, 1, cols = 1:20)
saveWorkbook(wb, "removeColWidthsExample.xlsx", overwrite = TRUE)

removeComment  Remove a comment from a cell

Description
Remove a cell comment from a worksheet

Usage
removeComment(wb, sheet, cols, rows, gridExpand = TRUE)
removeFilter

**Arguments**

- **wb**: A workbook object
- **sheet**: A vector of names or indices of worksheets
- **cols**: Columns to delete comments from
- **rows**: Rows to delete comments from
- **gridExpand**: If TRUE, all data in rectangle min(rows):max(rows) X min(cols):max(cols) will be removed.

**See Also**

- createComment
- writeComment

---

**Description**

Removes filters from addFilter() and writeData()

**Usage**

```r
removeFilter(wb, sheet)
```

**Arguments**

- **wb**: A workbook object
- **sheet**: A vector of names or indices of worksheets

**Examples**

```r
wb <- createWorkbook()
addWorksheet(wb, "Sheet 1")
addWorksheet(wb, "Sheet 2")
addWorksheet(wb, "Sheet 3")
writeData(wb, 1, iris)
addFilter(wb, 1, row = 1, cols = 1:ncol(iris))

## Equivalently
writeData(wb, 2, x = iris, withFilter = TRUE)

## Similarly
writeDataTable(wb, 3, iris)

## remove filters
```
removeRowHeights

Remove custom row heights from a worksheet

Description

Remove row heights from a worksheet

Usage

removeRowHeights(wb, sheet, rows)

Arguments

- wb: A workbook object
- sheet: A name or index of a worksheet
- rows: Indices of rows to remove custom height (if any) from.

Author(s)

Alexander Walker

See Also

setRowHeights

Examples

## Create a new workbook
wb <- loadWorkbook(file = system.file("loadExample.xlsx", package = "openxlsx"))

## remove any custom row heights in rows 1 to 10
removeRowHeights(wb, 1, rows = 1:10)
saveWorkbook(wb, "removeRowHeightsExample.xlsx", overwrite = TRUE)
removeTable  

Remove an Excel table in a workbook

Description

List Excel tables in a workbook

Usage

```r
removeTable(wb, sheet, table)
```

Arguments

- `wb`: A workbook object
- `sheet`: A name or index of a worksheet
- `table`: Name of table to remove. See `getTables`

Value

character vector of table names on the specified sheet

Examples

```r
wb <- createWorkbook()
addWorksheet(wb, sheetName = "Sheet 1")
addWorksheet(wb, sheetName = "Sheet 2")
writeDataTable(wb, sheet = "Sheet 1", x = iris, tableName = "iris")
writeDataTable(wb, sheet = 1, x = mtcars, tableName = "mtcars", startCol = 10)

removeWorksheet(wb, sheet = 1)  # delete worksheet removes table objects
writeDataTable(wb, sheet = 1, x = iris, tableName = "iris")
writeDataTable(wb, sheet = 1, x = mtcars, tableName = "mtcars", startCol = 10)

# removeTable() deletes table object and all data
getTables(wb, sheet = 1)
removeTable(wb = wb, sheet = 1, table = "iris")
writeDataTable(wb, sheet = 1, x = iris, tableName = "iris", startCol = 1)

getTables(wb, sheet = 1)
removeTable(wb = wb, sheet = 1, table = "iris")
writeDataTable(wb, sheet = 1, x = iris, tableName = "iris", startCol = 1)

saveWorkbook(wb = wb, file = "removeTableExample.xlsx", overwrite = TRUE)
```
renameWorksheet  
*Remove a worksheet from a workbook*

**Description**

Remove a worksheet from a Workbook object

**Usage**

renameWorksheet(wb, sheet)

**Arguments**

- **wb**  A workbook object
- **sheet**  A name or index of a worksheet

**Author(s)**

Alexander Walker

**Examples**

```r
## load a workbook
wb <- loadWorkbook(file = system.file("loadExample.xlsx", package = "openxlsx"))

## Remove sheet 2
renameWorksheet(wb, 2)

## save the modified workbook
saveWorkbook(wb, "renameWorksheetExample.xlsx", overwrite = TRUE)
```

renameWorksheet  
*Rename a worksheet*

**Description**

Rename a worksheet

**Usage**

renameWorksheet(wb, sheet, newName)
replaceStyle

Arguments

wb A Workbook object containing a worksheet
sheet The name or index of the worksheet to rename
newName The new name of the worksheet. No longer than 31 chars.

Details

DEPRECATED. Use names

Author(s)

Alexander Walker

Examples

## Create a new workbook
wb <- createWorkbook("CREATOR")

## Add 3 worksheets
addWorksheet(wb, "Worksheet Name")
addWorksheet(wb, "This is worksheet 2")
addWorksheet(wb, "Not the best name")

' ## rename all worksheets
names(wb) <- c("A", "B", "C")

## Rename worksheet 1 & 3
renameWorksheet(wb, 1, "New name for sheet 1")
names(wb)[[1]] <- "New name for sheet 1"
names(wb)[[3]] <- "A better name"

## Save workbook
saveWorkbook(wb, "renameWorksheetExample.xlsx", overwrite = TRUE)

replaceStyle |
Replacement an existing cell style

Description

Replace an existing cell style

Usage

replaceStyle(wb, index, newStyle)
Arguments

- wb: A workbook object
- index: Index of style object to replace
- newStyle: A style to replace the existing style as position index

Author(s)

Alexander Walker

See Also

getStyles

Examples

```r
## load a workbook
wb <- loadWorkbook(file = system.file("loadExample.xlsx", package = "openxlsx"))

## create a new style and replace style 2
newStyle <- createStyle(fgFill = "#00FF00")

## replace style 2
getStyles(wb)[1:3] ## prints styles
replaceStyle(wb, 2, newStyle = newStyle)

## Save workbook
saveWorkbook(wb, "replaceStyleExample.xlsx", overwrite = TRUE)
```

saveWorkbook

save Workbook to file

Description

save a Workbook object to file

Usage

saveWorkbook(wb, file, overwrite = FALSE)

Arguments

- wb: A Workbook object to write to file
- file: A character string naming an xlsx file
- overwrite: If TRUE, overwrite any existing file.
setColWidths

Author(s)
Alexander Walker

See Also
createWorkbook
addWorksheet
loadWorkbook
writeData
writeDataTable

Examples

## Create a new workbook and add a worksheet
wb <- createWorkbook("Creator of workbook")
addWorksheet(wb, sheetName = "My first worksheet")

## Save workbook to working directory
saveWorkbook(wb, file = "saveWorkbookExample.xlsx", overwrite = TRUE)

Description

Set worksheet column widths to specific width or "auto".

Usage

```
setColWidths(wb, sheet, cols, widths = 8.43, hidden = rep(FALSE, length(cols)), ignoreMergedCells = FALSE)
```

Arguments

- **wb**: A workbook object
- **sheet**: A name or index of a worksheet
- **cols**: Indices of cols to set width
- **widths**: widths to set cols to specified in Excel column width units or "auto" for automatic sizing. The widths argument is recycled to the length of cols.
- **hidden**: Logical vector. If TRUE the column is hidden.
- **ignoreMergedCells**: Ignore any cells that have been merged with other cells in the calculation of "auto" column widths.
Details

The global min and max column width for "auto" columns is set by (default values show):

- `options("openxlsx.minWidth" = 3)
- `options("openxlsx.maxWidth" = 250) ## This is the maximum width allowed in Excel

NOTE: The calculation of column widths can be slow for large worksheets.

Author(s)

Alexander Walker

See Also

`removeColWidths`

Examples

```r
## Create a new workbook
wb <- createWorkbook()

## Add a worksheet
addWorksheet(wb, "Sheet 1")

## set col widths
setColWidths(wb, 1, cols = c(1,4,6,7,9), widths = c(16,15,12,18,33))

## auto columns
addWorksheet(wb, "Sheet 2")
writeData(wb, sheet = 2, x = iris)
setColWidths(wb, sheet = 2, cols = 1:5, widths = "auto")

## Save workbook
saveWorkbook(wb, "setColWidthsExample.xlsx", overwrite = TRUE)
```

---

**setFooter**

*Set footer for all worksheets*

Description

DEPRECATED

Usage

`setFooter(wb, text, position = "center")`
**setHeader**

**Arguments**

- **wb**: A workbook object
- **text**: Footer text. A character vector of length 1.
- **position**: Position of text in footer. One of "left", "center" or "right"

**Author(s)**

Alexander Walker

**Examples**

```r
## Not run:
wb <- createWorkbook("Edgar Anderson")
addWorksheet(wb, "S1")
writeDataTable(wb, "S1", x = iris[1:30,], xy = c("C", 5))

## set all headers
setHeader(wb, "This is a header", position="center")
setHeader(wb, "To the left", position="left")
setHeader(wb, "On the right", position="right")

## set all footers
setFooter(wb, "Center Footer Here", position="center")
setFooter(wb, "Bottom left", position="left")
setFooter(wb, Sys.Date(), position="right")

saveWorkbook(wb, "headerFooterExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

**setDescription**

*Set header for all worksheets*

**Description**

DEPRECATED

**Usage**

```r
setHeader(wb, text, position = "center")
```

**Arguments**

- **wb**: A workbook object
- **text**: Header text. A character vector of length 1.
- **position**: Position of text in header. One of "left", "center" or "right"
Author(s)

Alexander Walker

Examples

```r
## Not run:
wb <- createWorkbook("Edgar Anderson")
addWorksheet(wb, "S1")
writeDataTable(wb, "S1", x = iris[1:30,], xy = c("C", 5))

## set all headers
setHeader(wb, "This is a header", position="center")
setHeader(wb, "To the left", position="left")
setHeader(wb, "On the right", position="right")

## set all footers
setFooter(wb, "Center Footer Here", position="center")
setFooter(wb, "Bottom left", position="left")
setFooter(wb, Sys.Date(), position="right")

saveWorkbook(wb, "headerHeaderExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

**setHeaderFooter**

Set document headers and footers

**Description**

Set document headers and footers

**Usage**

```
setHeaderFooter(wb, sheet, header = NULL, footer = NULL,
               evenHeader = NULL, evenFooter = NULL, firstHeader = NULL,
               firstFooter = NULL)
```

**Arguments**

- **wb**: A workbook object
- **sheet**: A name or index of a worksheet
- **header**: document header. Character vector of length 3 corresponding to positions left, center, right. Use NA to skip a position.
- **footer**: document footer. Character vector of length 3 corresponding to positions left, center, right. Use NA to skip a position.
- **evenHeader**: document header for even pages.
- **evenFooter**: document footer for even pages.
`setHeaderFooter`

**firstHeader**  
document header for first page only.

**firstFooter**  
document footer for first page only.

**Details**

Headers and footers can contain special tags

- `&[Page]` Page number
- `&[Pages]` Number of pages
- `&[Date]` Current date
- `&[Time]` Current time
- `&[Path]` File path
- `&[File]` File name
- `&[Tab]` Worksheet name

**Author(s)**

Alexander Walker

**See Also**

`addWorksheet` to set headers and footers when adding a worksheet

## Examples

```r
wb <- createWorkbook()
addWorksheet(wb, "S1")
addWorksheet(wb, "S2")
addWorksheet(wb, "S3")
addWorksheet(wb, "S4")
writeData(wb, 1, 1:400)
writeData(wb, 2, 1:400)
writeData(wb, 3, 3:400)
writeData(wb, 4, 3:400)
setHeaderFooter(wb, sheet = "S1",
               header = c("ODD HEAD LEFT", "ODD HEAD CENTER", "ODD HEAD RIGHT"),
               footer = c("ODD FOOT RIGHT", "ODD FOOT CENTER", "ODD FOOT RIGHT"),
               evenHeader = c("EVEN HEAD LEFT", "EVEN HEAD CENTER", "EVEN HEAD RIGHT"),
               evenFooter = c("EVEN FOOT RIGHT", "EVEN FOOT CENTER", "EVEN FOOT RIGHT"),
               firstHeader = c("TOP", "OF FIRST", "PAGE"),
               firstFooter = c("BOTTOM", "OF FIRST", "PAGE"))
setHeaderFooter(wb, sheet = 2,
               header = c("&[Date]", "ALL HEAD CENTER 2", "&[Page] / &[Pages]"),
               footer = c("&[Path] &[File]", NA, "&[Tab]")
               firstHeader = c(NA, "Center Header of First Page", NA),
               firstFooter = c(NA, "Center Footer of First Page", NA))
```
setHeaderFooter(wb, sheet = 3,
    header = c("ALL HEAD LEFT 2", "ALL HEAD CENTER 2", "ALL HEAD RIGHT 2"),
    footer = c("ALL FOOT RIGHT 2", "ALL FOOT CENTER 2", "ALL FOOT RIGHT 2"))

setHeaderFooter(wb, sheet = 4,
    firstHeader = c("FIRST ONLY L", NA, "FIRST ONLY R"),
    firstFooter = c("FIRST ONLY L", NA, "FIRST ONLY R"))

saveWorkbook(wb, "setHeaderFooterExample.xlsx", overwrite = TRUE)

---

**setRowHeights**

*Set worksheet row heights*

**Description**

Set worksheet row heights

**Usage**

`setRowHeights(wb, sheet, rows, heights)`

**Arguments**

- `wb`: A workbook object
- `sheet`: A name or index of a worksheet
- `rows`: Indices of rows to set height
- `heights`: Heights to set rows to specified in Excel column height units.

**Author(s)**

Alexander Walker

**See Also**

`removeRowHeights`

**Examples**

```r
## Create a new workbook
wb <- createWorkbook()

## Add a worksheet
addWorksheet(wb, "Sheet 1")

## set row heights
setRowHeights(wb, 1, rows = c(1,4,22,2,19), heights = c(24,28,32,42,33))
```
sheets

## overwrite row 1 height
setRowHeights(wb, 1, rows = 1, heights = 40)

## Save workbook
saveWorkbook(wb, "setRowHeightsExample.xlsx", overwrite = TRUE)

sheets  Returns names of worksheets.

**Description**

DEPRECATED. Use names().

**Usage**

```r
sheets(wb)
```

**Arguments**

- `wb` A workbook object

**Details**

DEPRECATED. Use names

**Value**

Name of worksheet(s) for a given index

**Author(s)**

Alexander Walker

**See Also**

- names to rename a worksheet in a Workbook

**Examples**

```r
## Create a new workbook
wb <- createWorkbook()

## Add some worksheets
addWorksheet(wb, "Worksheet Name")
addWorksheet(wb, "This is worksheet 2")
addWorksheet(wb, "The third worksheet")
```
## sheetVisibility

*Get/set worksheet visible state*

### Description

Get and set worksheet visible state

### Usage

```r
sheetVisibility(wb)
```

```r
sheetVisibility(wb) <- value
```

### Arguments

- **wb**: A workbook object
- **value**: a logical/character vector the same length as `sheetVisibility(wb)`

### Value

Character vector of worksheet names.

Vector of "hidden", "visible", "veryHidden"

### Examples

```r
wb <- createWorkbook()
addWorksheet(wb, sheetName = "S1", visible = FALSE)
addWorksheet(wb, sheetName = "S2", visible = TRUE)
addWorksheet(wb, sheetName = "S3", visible = FALSE)

sheetVisibility(wb)
sheetVisibility(wb)[1] <- TRUE  # show sheet 1
sheetVisibility(wb)[2] <- FALSE  # hide sheet 2
sheetVisibility(wb)[3] <- "hidden"  # hide sheet 3
sheetVisibility(wb)[3] <- "veryHidden"  # hide sheet 3 from UI
```
Get worksheet visible state.

Description

DEPRECATED - Use function `sheetVisibility()`

Usage

```
sheetVisible(wb)

sheetVisible(wb) <- value
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wb</td>
<td>A workbook object</td>
</tr>
<tr>
<td>value</td>
<td>a logical vector the same length as sheetVisible(wb)</td>
</tr>
</tbody>
</table>

Value

Character vector of worksheet names.

TRUE if sheet is visible, FALSE if sheet is hidden

Author(s)

Alexander Walker

Examples

```
wb <- createWorkbook()
addWorksheet(wb, sheetName = "S1", visible = FALSE)
addWorksheet(wb, sheetName = "S2", visible = TRUE)
addWorksheet(wb, sheetName = "S3", visible = FALSE)

sheetVisible(wb)
sheetVisible(wb)[1] <- TRUE  ## show sheet 1
sheetVisible(wb)[2] <- FALSE  ## hide sheet 2
```
showGridLines

Set worksheet gridlines to show or hide.

Description

Set worksheet gridlines to show or hide.

Usage

showGridLines(wb, sheet, showGridLines = FALSE)

Arguments

wb A workbook object
sheet A name or index of a worksheet
showGridLines A logical. If TRUE, grid lines are hidden.

Author(s)

Alexander Walker

Examples

wb <- loadWorkbook(file = system.file("loadExample.xlsx", package = "openxlsx"))
names(wb) # list worksheets in workbook
showGridLines(wb, 1, showGridLines = FALSE)
showGridLines(wb, "testing", showGridLines = FALSE)
saveWorkbook(wb, "showGridLinesExample.xlsx", overwrite = TRUE)

worksheetOrder

Order of worksheets in xlsx file

Description

Get/set order of worksheets in a Workbook object

Usage

worksheetOrder(wb)

worksheetOrder(wb) <- value

Arguments

wb A workbook object
value Vector specifying order to write worksheets to file
write.xlsx

Details

This function does not reorder the worksheets within the workbook object, it simply shuffles the order when writing to file.

Examples

```r
## setup a workbook with 3 worksheets
wb <- createWorkbook()
addWorksheet(wb = wb, sheetName = "Sheet 1", gridLines = FALSE)
writeDataTable(wb = wb, sheet = 1, x = iris)

addWorksheet(wb = wb, sheetName = "mtcars (Sheet 2)", gridLines = FALSE)
writeData(wb = wb, sheet = 2, x = mtcars)

addWorksheet(wb = wb, sheetName = "Sheet 3", gridLines = FALSE)
writeData(wb = wb, sheet = 3, x = Formaldehyde)

worksheetOrder(wb)
names(wb)
worksheetOrder(wb) <- c(1,3,2) # switch position of sheets 2 & 3
writeData(wb, 2, 'This is still the "mtcars" worksheet', startCol = 15)
worksheetOrder(wb)
names(wb) # # ordering within workbook is not changed

saveWorkbook(wb, "worksheetOrderExample.xlsx", overwrite = TRUE)
worksheetOrder(wb) <- c(3,2,1)
saveWorkbook(wb, "worksheetOrderExample2.xlsx", overwrite = TRUE)
```

write.xlsx

---

write data to an xlsx file

Description

write a data.frame or list of data.frames to an xlsx file

Usage

```r
write.xlsx(x, file, asTable = FALSE, ...)
```

Arguments

- `x` object or a list of objects that can be handled by `writeData` to write to file
- `file` xlsx file name
- `asTable` write using `writeDataTable` as opposed to `writeData`
- `...` optional parameters to pass to functions:
  - `createWorkbook`
  - `addWorksheet`
• writeData
• freezePane
• saveWorkbook

see details.

Details
Optional parameters are:

createWorkbook Parameters
• creator A string specifying the workbook author

addWorksheet Parameters
• sheetName Name of the worksheet
• gridLines A logical. If FALSE, the worksheet grid lines will be hidden.
• tabColour Colour of the worksheet tab. A valid colour (belonging to colours()) or a valid hex colour beginning with "#".
• zoom A numeric between 10 and 400. Worksheet zoom level as a percentage.

writeData/writeDataTable Parameters
• startCol A vector specifying the starting column(s) to write df
• startRow A vector specifying the starting row(s) to write df
• xy An alternative to specifying startCol and startRow individually. A vector of the form c(startCol, startRow)
• colNames or col.names If TRUE, column names of x are written.
• rowNames or row.names If TRUE, row names of x are written.
• headerStyle Custom style to apply to column names.
• borders Either "surrounding", "columns" or "rows" or NULL. If "surrounding", a border is drawn around the data. If "rows", a surrounding border is drawn a border around each row. If "columns", a surrounding border is drawn with a border between each column. If "all" all cell borders are drawn.
• borderColour Colour of cell border
• borderStyle Border line style.
• keepNA If TRUE, NA values are converted to #N/A in Excel else NA cells will be empty. Defaults to FALSE.

freezePane Parameters
• firstActiveRow Top row of active region to freeze pane.
• firstActiveCol Furthest left column of active region to freeze pane.
• firstRow If TRUE, freezes the first row (equivalent to firstActiveRow = 2)
• firstCol If TRUE, freezes the first column (equivalent to firstActiveCol = 2)

colWidths Parameters
• **colWidths** Must be value "auto". Sets all columns containing data to auto width.

**saveWorkbook Parameters**

• **overwrite** Overwrite existing file (Defaults to TRUE as with write.table)

columns of x with class Date or POSIXt are automatically styled as dates and datetimes respectively.

**Value**

A workbook object

**Author(s)**

Alexander Walker

**See Also**

addWorksheet

writeData

createStyle for style parameters

**Examples**

```r
## write to working directory
options("openxlsx.borderColour" = "#4F80BD") ## set default border colour
write.xlsx(iris, file = "writeXLSX1.xlsx", colNames = TRUE, borders = "columns")
write.xlsx(iris, file = "writeXLSX2.xlsx", colNames = TRUE, borders = "surrounding")

hs <- createStyle(textDecoration = "BOLD", fontColour = "FFFFFFFF", fontSize=12,
    fontName="Arial Narrow", fgFill = "#4F80BD")
write.xlsx(iris, file = "writeXLSX3.xlsx", colNames = TRUE, borders = "rows", headerStyle = hs)

## Lists elements are written to individual worksheets, using list names as sheet names if available
l <- list("IRIS" = iris, "MTCATS" = mtcars, matrix(runif(1000), ncol = 5))
write.xlsx(l, "writeList1.xlsx", colWidths = c(NA, "auto", "auto"))

## different sheets can be given different parameters
write.xlsx(l, "writeList2.xlsx", startCol = c(1,2,3), startRow = c(2),
    asTable = c(TRUE, TRUE, FALSE), withFilter = c(TRUE, FALSE, FALSE))
```
writeComment

write a cell comment

Description

Write a Comment object to a worksheet

Usage

writeComment(wb, sheet, col, row, comment, xy = NULL)

Arguments

wb  
A workbook object

sheet  
A vector of names or indices of worksheets

col  
Column a column number of letter

row  
A row number.

comment  
A Comment object. See createComment.

xy  
An alternative to specifying col and row individually. A vector of the form c(col, row).

See Also

createComment

Examples

wb <- createWorkbook()
addWorksheet(wb, "Sheet 1")
c1 <- createComment(comment = "this is comment")
writeComment(wb, 1, col = "B", row = 10, comment = c1)
s1 <- createStyle(fontSize = 12, fontColour = "red", textDecoration = c("BOLD"))
s2 <- createStyle(fontSize = 9, fontColour = "black")
c2 <- createComment(comment = c("This Part Bold red\n\n", "This part black"), style = c(s1, s2))
c2
writeComment(wb, 1, col = 6 , row = 3, comment = c2)
saveWorkbook(wb, file = "writeCommentExample.xlsx", overwrite = TRUE)
writeData  Write an object to a worksheet

Description
Write an object to worksheet with optional styling.

Usage
writeData(wb, sheet, x, startCol = 1, startRow = 1, xy = NULL, colNames = TRUE, rowNames = FALSE, headerStyle = NULL, borders = c("none", "surrounding", "rows", "columns", "all"), borderColour = getOption("openxlsx.borderColour", "black"), borderStyle = getOption("openxlsx.borderStyle", "thin"), withFilter = FALSE, keepNA = FALSE, name = NULL, sep = ", ")

Arguments
wb  A Workbook object containing a worksheet.
sheet  The worksheet to write to. Can be the worksheet index or name.
x  Object to be written. For classes supported look at the examples.
startCol  A vector specifying the starting column to write to.
startRow  A vector specifying the starting row to write to.
xy  An alternative to specifying startCol and startRow individually. A vector of the form c(startCol, startRow).
colNames  If TRUE, column names of x are written.
rowNames  If TRUE, data.frame row names of x are written.
headerStyle  Custom style to apply to column names.
borders  Either "none" (default), "surrounding", "columns", "rows" or respective abbreviations. If "surrounding", a border is drawn around the data. If "rows", a surrounding border is drawn with a border around each row. If "columns", a surrounding border is drawn with a border between each column. If "all" all cell borders are drawn.
borderColour  Colour of cell border. A valid colour (belonging to colours() or a hex colour code, eg see here).
borderStyle  Border line style
  • none  no border
  • thin  thin border
  • medium  medium border
  • dashed  dashed border
  • dotted  dotted border
  • thick  thick border
• **double** double line border
• **hair** hairline border
• **mediumDashed** medium weight dashed border
• **dashDot** dash-dot border
• **mediumDashDot** medium weight dash-dot border
• **dashDotDot** dash-dot-dot border
• **mediumDashDotDot** medium weight dash-dot-dot border
• **slantDashDot** slanted dash-dot border

withFilter If TRUE, add filters to the column name row. NOTE can only have one filter per worksheet.

keepNA If TRUE, NA values are converted to #N/A in Excel else NA cells will be empty.

name If not NULL, a named region is defined.

sep Only applies to list columns. The separator used to collapse list columns to a character vector e.g. `sapply(x$list_column, paste, collapse = sep)`.

Details

Formulae written using writeFormula to a Workbook object will not get picked up by read.xlsx(). This is because only the formula is written and left to Excel to evaluate the formula when the file is opened in Excel.

Author(s)

Alexander Walker

See Also

writeDataTable

Examples

```r
## See formatting vignette for further examples.
## Options for default styling (These are the defaults)
options("openxlsx.borderColour" = "black")
options("openxlsx.borderStyle" = "thin")
options("openxlsx.dateFormat" = "mm/dd/yyyy")
options("openxlsx.datetimeFormat" = "yyyy-mm-dd hh:mm:ss")
options("openxlsx.numFmt" = NULL)

## Change the default border colour to #F81BD
options("openxlsx.borderColour" = "#F81BD")
```

```r
# Create Workbook object and add worksheets
wb <- createWorkbook()
```
writeData

```r
## Add worksheets
addWorksheet(wb, "Cars")
addWorksheet(wb, "Formula")

x <- mtcars[1:6,]
writeData(wb, "Cars", x, startCol = 2, startRow = 3, rowNames = TRUE)

# Bordering
writeData(wb, "Cars", x, rowNames = TRUE, startCol = "O", startRow = 3, borders="surrounding", borderColour = "black") ## black border
writeData(wb, "Cars", x, rowNames = TRUE, startCol = 2, startRow = 12, borders="columns")
writeData(wb, "Cars", x, rowNames = TRUE, startCol="O", startRow = 12, borders="rows")

# Header Styles
hs1 <- createStyle(fgFill = "#DCE6FF", halign = "CENTER", textDecoration = "italic", border = "Bottom")
writeData(wb, "Cars", x, colNames = TRUE, rowNames = TRUE, startCol="B", startRow = 23, borders="rows", headerStyle = hs1, borderStyle = "dashed")

hs2 <- createStyle(fontColour = "#ffffff", fgFill = "#4F80BD", halign = "center", valign = "center", textDecoration = "bold", border = "TopBottomLeftRight")
writeData(wb, "Cars", x, colNames = TRUE, rowNames = TRUE, startCol="O", startRow = 23, borders="columns", headerStyle = hs2)

# Hyperlinks
## - vectors/columns with class 'hyperlink' are written as hyperlinks'
v <- rep("https://CRAN.R-project.org/", 4)
names(v) <- paste("Hyperlink", 1:4) # Optional: names will be used as display text
class(v) <- 'hyperlink'
writeData(wb, "Cars", x = v, xy = c("B", 32))
```
writeDataTable

## Formulas
- vectors/columns with class 'formula' are written as formulas'

```r
df <- data.frame(x=1:3, y = 1:3,
                  z = paste(paste0("A", 1:3+1L), paste0("B", 1:3+1L), sep = "+"),
                  stringsAsFactors = FALSE)

class(df$z) <- c(class(df$z), "formula")

writeData(wb, sheet = "Formula", x = df)
```

## Save workbook
- Open in excel without saving file: openXL(wb)

```r
saveWorkbook(wb, "writeDataExample.xlsx", overwrite = TRUE)
```

---

**writeDataTable**  
*Write to a worksheet as an Excel table*

### Description
Write to a worksheet and format as an Excel table

### Usage
```r
writeDataTable(wb, sheet, x, startCol = 1, startRow = 1, xy = NULL, 
               colNames = TRUE, rowNames = FALSE, tableStyle = "TableStyleLight9", 
               tableName = NULL, headerStyle = NULL, withFilter = TRUE, 
               keepNA = FALSE, sep = ",", stack = FALSE, firstColumn = FALSE, 
               lastColumn = FALSE, bandedRows = TRUE, bandedCols = FALSE)
```

### Arguments
- **wb**: A Workbook object containing a worksheet.
- **sheet**: The worksheet to write to. Can be the worksheet index or name.
- **x**: A dataframe.
- **startCol**: A vector specifying the starting column to write df
- **startRow**: A vector specifying the starting row to write df
- **xy**: An alternative to specifying startCol and startRow individually. A vector of the form c(startCol, startRow)
- **colNames**: If TRUE, column names of x are written.
- **rowNames**: If TRUE, row names of x are written.
- **tableStyle**: Any excel table style name or "none" (see "formatting" vignette).
- **tableName**: name of table in workbook. The table name must be unique.
writeDataTable

headerStyle Custom style to apply to column names.
withFilter If TRUE, columns with have filters in the first row.
keepNA If TRUE, NA values are converted to #N/A in Excel else NA cells will be empty.
sep Only applies to list columns. The separator used to collapse list columns to a character vector e.g. sapply(x$list_column, paste, collapse = sep).

The below options correspond to Excel table options:

- Header Row
- First Column
- Total Row
- Last Column
- Banded Rows
- Banded Columns

Table Style Options

stack If TRUE the new style is merged with any existing cell styles. If FALSE, any existing style is replaced by the new style.

firstColumn logical. If TRUE, the first column is bold
lastColumn logical. If TRUE, the last column is bold
bandedRows logical. If TRUE, rows are colour banded
bandedCols logical. If TRUE, the columns are colour banded

Details

columns of x with class Date/POSIXt, currency, accounting, hyperlink, percentage are automatically styled as dates, currency, accounting, hyperlinks, percentages respectively.

Author(s)
Alexander Walker

See Also
addWorksheet
writeData
removeTable
getTables

Examples

```r
## see package vignettes for further examples.

### Create Workbook object and add worksheets
wb <- createWorkbook()
addWorksheet(wb, "S1")
addWorksheet(wb, "S2")
addWorksheet(wb, "S3")
```
## writeFormula

### Description

Write a character vector containing Excel formula to a worksheet

### Usage

```r
writeFormula(wb, sheet, x, startCol = 1, startRow = 1, xy = NULL)
```
writeFormula

Arguments

- **wb**: A Workbook object containing a worksheet.
- **sheet**: The worksheet to write to. Can be the worksheet index or name.
- **x**: A character vector.
- **startCol**: A vector specifying the starting column to write to.
- **startRow**: A vector specifying the starting row to write to.
- **xy**: An alternative to specifying `startCol` and `startRow` individually. A vector of the form `c(startCol, startRow)`.

Author(s)

Alexander Walker

See Also

- writeData

Examples

```r
## There are 3 ways to write a formula

wb <- createWorkbook()
addWorksheet(wb, "Sheet 1")
writeData(wb, "Sheet 1", x = iris)

## SEE int2col() to convert int to Excel column label

## 1. - As a character vector using writeFormula

v <- c("SUM(A2:A151)", "AVERAGE(B2:B151)") ## skip header row
writeFormula(wb, sheet = 1, x = v, startCol = 10, startRow = 2)
writeFormula(wb, 1, x = "A2 + B2", startCol = 10, startRow = 10)

## 2. - As a data.frame column with class "formula" using writeData

df <- data.frame(x=1:3,
                 y = 1:3,
                 z = paste(paste0("A", 1:3+1L), paste0("B", 1:3+1L), sep = " + ",
                        z2 = sprintf("ADDRESS(1,%s)", 1:3),
                 stringsAsFactors = FALSE)

class(df$z) <- c(class(df$z), "formula")
class(df$z2) <- c(class(df$z2), "formula")

addWorksheet(wb, "Sheet 2")
writeData(wb, sheet = 2, x = df)
```
```r
## 3. - As a vector with class "formula" using writeData

v2 <- c("SUM(A2:A4)", "AVERAGE(B2:B4)", "MEDIAN(C2:C4)"
class(v2) <- c(class(v2), "formula")

writeData(wb, sheet = 2, x = v2, startCol = 10, startRow = 2)

## Writing internal hyperlinks
wb <- createWorkbook()
addWorksheet(wb, "Sheet1")
addWorksheet(wb, "Sheet2")
writeFormula(wb, "Sheet1", x = 'HYPERLINK("#Sheet2!B3", "Text to Display - Link to Sheet2")')
saveWorkbook(wb, "writeFormulaHyperlinkExample.xlsx", overwrite = TRUE)
```
Index

addFilter, 3, 4
addStyle, 4, 19
addWorksheet, 5, 40, 55, 59, 67, 73
allNequal, 7
conditionalFormat, 8
conditionalFormatting, 8, 9
convertFromExcelRef, 12
convertToDate, 13, 26
convertToDateTime, 14
copyWorkbook, 14
createComment, 15, 49, 68
createNamedRegion, 16, 27
createStyle, 5, 9, 10, 15, 17, 67
createWorkbook, 20, 55
databar (conditionalFormatting), 9
dataValidation, 21
deleteData, 22
freezePane, 23
generateBaseFont, 24
generateCellRefs, 25
generateDateOrigin, 26
generateNamedRegions, 16, 27, 45, 47
generateSheetNames, 28
generateStyles, 28, 54
generateTables, 29, 51, 73
insertImage, 30, 31
insertPlot, 30, 31
int2col, 32
loadWorkbook, 21, 33, 55
makeHyperlinkString, 34
mergeCells, 35, 47
modifyBaseFont, 36
naming, 37, 53, 61
names <- .Workbook (names), 37
openXL, 38
openxlsx, 39
openxlsx-package (openxlsx), 39
pageBreak, 40
pageSetup, 40
read.xlsx, 39, 44, 47
readWorkbook, 45
removeCellMerge, 35, 47
removeColWidths, 48, 56
removeComment, 48
removeFilter, 49
removeRowHeights, 50, 60
removeTable, 51, 73
renameWorksheet, 33, 52
replaceStyle, 28, 53
saveWorkbook, 21, 54
setColWidths, 48, 55
setFooter, 56
setHeader, 57
setHeaderFooter, 58
setRowHeights, 50, 60
sheets, 61
sheetVisibility, 62
sheetVisibility <- (sheetVisibility), 62
sheetVisible, 63
sheetVisible <- (sheetVisible), 63
showGridLines, 64
worksheetOrder, 64
worksheetOrder <- (worksheetOrder), 64
write.xlsx, 39, 65
writeComment, 15, 49, 68
writeData, 4, 13, 39, 55, 65, 67, 69, 73, 75
writeDataTable, 39, 55, 70, 72
writeFormula, 34, 74