

# Package: openxlsx (via r-universe)

August 31, 2024

**Type** Package

**Title** Read, Write and Edit xlsx Files

**Version** 4.2.7

**Date** 2024-08-30

**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** MIT + file LICENSE

**URL** <https://ycphs.github.io/openxlsx/index.html>,  
<https://github.com/ycphs/openxlsx>

**BugReports** <https://github.com/ycphs/openxlsx/issues>

**Depends** R (>= 3.3.0)

**Imports** grDevices, methods, Rcpp, stats, stringi, utils, zip

**Suggests** curl, formula.tools, knitr, rmarkdown, testthat

**LinkingTo** Rcpp

**VignetteBuilder** knitr

**Encoding** UTF-8

**Language** en-US

**RoxygenNote** 7.3.2

**Collate** 'CommentClass.R' 'HyperlinkClass.R' 'RcppExports.R'  
'class\_definitions.R' 'StyleClass.R' 'WorkbookClass.R'  
'asserts.R' 'baseXML.R' 'borderFunctions.R' 'build\_workbook.R'  
'chartsheet\_class.R' 'conditional\_formatting.R'  
'data-fontSizeLookupTables.R' 'helperFunctions.R'  
'loadWorkbook.R' 'onUnload.R' 'openXL.R' 'openxlsx-package.R'  
'openxlsx.R' 'openxlsxCoerce.R' 'readWorkbook.R'  
'setWindowSize.R' 'sheet\_data\_class.R' 'utils.R'  
'workbook\_column\_widths.R' 'workbook\_read\_workbook.R'

'workbook\_write\_data.R' 'worksheet\_class.R' 'wrappers.R'  
 'writeData.R' 'writeDataTable.R' 'writexlsx.R' 'zzz.R'

**LazyData** true

**NeedsCompilation** yes

**Author** Philipp Schauburger [aut], Alexander Walker [aut], Luca Braglia [ctb], Joshua Sturm [ctb], Jan Marvin Garbuszus [ctb, cre], Jordan Mark Barbone [ctb] (<<https://orcid.org/0000-0001-9788-3628>>), David Zimmermann [ctb], Reinhold Kainhofer [ctb]

**Maintainer** Jan Marvin Garbuszus <[jan.garbuszus@ruhr-uni-bochum.de](mailto:jan.garbuszus@ruhr-uni-bochum.de)>

**Repository** CRAN

**Date/Publication** 2024-08-30 08:30:02 UTC

## Contents

activeSheet . . . . .	4
addCreator . . . . .	5
addFilter . . . . .	5
addStyle . . . . .	6
addWorksheet . . . . .	7
all.equal . . . . .	10
as.character.formula . . . . .	10
auto_heights . . . . .	11
buildWorkbook . . . . .	12
cloneWorksheet . . . . .	14
col2int . . . . .	15
conditionalFormat . . . . .	15
conditionalFormatting . . . . .	17
convertFromExcelRef . . . . .	23
convertToDate . . . . .	23
convertToDateTime . . . . .	24
copyWorkbook . . . . .	25
createComment . . . . .	25
createNamedRegion . . . . .	26
createStyle . . . . .	28
createWorkbook . . . . .	31
dataValidation . . . . .	32
deleteData . . . . .	34
deleteDataColumn . . . . .	35
freezePane . . . . .	36
getBaseFont . . . . .	37
getCellRefs . . . . .	38
getCreators . . . . .	38
getDateOrigin . . . . .	39
getNamedRegions . . . . .	40
getSheetNames . . . . .	41

getStyles . . . . .	42
getTables . . . . .	42
get_worksheet_entries . . . . .	43
groupColumns . . . . .	44
groupRows . . . . .	45
if_null_then . . . . .	46
insertImage . . . . .	47
insertPlot . . . . .	48
int2col . . . . .	50
loadWorkbook . . . . .	50
makeHyperlinkString . . . . .	51
mergeCells . . . . .	53
modifyBaseFont . . . . .	54
names . . . . .	55
openXL . . . . .	56
openxlsx . . . . .	57
openxlsxFontSizeLookupTable . . . . .	58
openxlsx_options . . . . .	59
pageBreak . . . . .	60
pageSetup . . . . .	60
protectWorkbook . . . . .	64
protectWorksheet . . . . .	65
read.xlsx . . . . .	67
readWorkbook . . . . .	69
removeCellMerge . . . . .	71
removeColWidths . . . . .	72
removeComment . . . . .	73
removeFilter . . . . .	73
removeRowHeights . . . . .	74
removeTable . . . . .	75
removeWorksheet . . . . .	76
renameWorksheet . . . . .	77
replaceStyle . . . . .	78
saveWorkbook . . . . .	79
setColWidths . . . . .	80
setFooter . . . . .	81
setHeader . . . . .	82
setHeaderFooter . . . . .	83
setLastModifiedBy . . . . .	85
setRowHeights . . . . .	86
setWindowSize . . . . .	87
sheets . . . . .	88
sheetVisibility . . . . .	89
sheetVisible . . . . .	90
showGridLines . . . . .	91
temp.xlsx . . . . .	91
ungroupColumns . . . . .	92
ungroupRows . . . . .	92

worksheetOrder . . . . .	93
write.xlsx . . . . .	94
writeComment . . . . .	96
writeData . . . . .	98
writeDataTable . . . . .	102
writeFormula . . . . .	106

<b>Index</b>	<b>109</b>
--------------	------------

---

activeSheet	<i>Get/set active sheet of the workbook</i>
-------------	---

---

## Description

Get and set active sheet of the workbook

## Usage

```
activeSheet(wb)
```

```
activeSheet(wb) <- value
```

## Arguments

wb	A workbook object
value	index of the active sheet or name of the active sheet

## Value

return the active sheet of the workbook

## Author(s)

Philipp Schauburger

## Examples

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

activeSheet(wb) # default value is the first sheet active
activeSheet(wb) <- 1 ## active sheet S1
activeSheet(wb)
activeSheet(wb) <- "S2" ## active sheet S2
activeSheet(wb)
```

---

addCreator	<i>Add another author to the meta data of the file.</i>
------------	---

---

**Description**

Just a wrapper of `wb$addCreator()`

**Usage**

```
addCreator(wb, Creator)
```

**Arguments**

wb	A workbook object
Creator	A string object with the name of the creator

**Author(s)**

Philipp Schauburger

**Examples**

```
wb <- createWorkbook()
addCreator(wb, "test")
```

---

addFilter	<i>Add column filters</i>
-----------	---------------------------

---

**Description**

Add excel column filters to a worksheet

**Usage**

```
addFilter(wb, sheet, rows, cols)
```

**Arguments**

wb	A workbook object
sheet	A name or index of a worksheet
rows	A row number.
cols	columns to add filter to.

**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.

**See Also**

[writeData\(\)](#)  
[addFilter\(\)](#)

**Examples**

```
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)
## Not run:
saveWorkbook(wb, file = "addFilterExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

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

**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"
)

## 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
## Not run:
saveWorkbook(wb, "addStyleExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

addWorksheet

---

*Add a worksheet to a workbook*


---

**Description**

Add a worksheet to a Workbook object

**Usage**

```

addWorksheet(
  wb,
  sheetName,
  gridLines = openxlsx_getOp("gridLines", TRUE),
  tabColour = NULL,
  zoom = 100,
  header = openxlsx_getOp("header"),
  footer = openxlsx_getOp("footer"),
  evenHeader = openxlsx_getOp("evenHeader"),
  evenFooter = openxlsx_getOp("evenFooter"),
  firstHeader = openxlsx_getOp("firstHeader"),
  firstFooter = openxlsx_getOp("firstFooter"),
  visible = TRUE,
  paperSize = openxlsx_getOp("paperSize", 9),
  orientation = openxlsx_getOp("orientation", "portrait"),
  vdpi = openxlsx_getOp("vdpi", 300),
  hdpi = openxlsx_getOp("hdpi", 300)
)

```

**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**

```
## 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("&[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
## Not run:
saveWorkbook(wb, "addWorksheetExample.xlsx", overwrite = TRUE)

## End(Not run)

```

---

all.equal	<i>Check equality of workbooks</i>
-----------	------------------------------------

---

### Description

Check equality of workbooks

### Usage

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

### Arguments

target	A Workbook object
current	A Workbook object
...	ignored

---

as.character.formula	as.character.formula()
----------------------	------------------------

---

### Description

This function exists to prevent conflicts with `as.character.formula` methods from other packages

### Usage

```
## S3 method for class 'formula'
as.character(x, ...)
```

**Arguments**

x                    object to be coerced or tested.  
 ...                  Not implemented

**Value**

base::as.character.default(x)

---

auto\_heights                    *Compute optimal row heights*

---

**Description**

Compute optimal row heights for cell with fixed width and enabled automatic row heights parameter

**Usage**

```
auto_heights(
  wb,
  sheet,
  selected,
  fontsize = NULL,
  factor = 1,
  base_height = 15,
  extra_height = 12
)
```

**Arguments**

wb                    workbook  
 sheet                worksheet  
 selected             selected rows  
 fontsize             font size, optional (get base font size by default)  
 factor                factor to manually adjust font width, e.g., for bold fonts, optional  
 base\_height         basic row height, optional  
 extra\_height        additional row height per new line of text, optional

**Value**

list of indices of columns with fixed widths and optimal row heights

**Author(s)**

David Breuer

**Examples**

```
## Create new workbook
wb <- createWorkbook()
addWorksheet(wb, "Sheet")
sheet <- 1

## Write dummy data
long_string <- "ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC"
writeData(wb, sheet, c("A", long_string, "CCC"), startCol = 2, startRow = 3)
writeData(wb, sheet, c(4, 5), startCol = 4, startRow = 3)

## Set column widths and get optimal row heights
setColWidths(wb, sheet, c(1,2,3,4), c(10,20,10,20))
auto_heights(wb, sheet, 1:5)
```

---

 buildWorkbook

*Build Workbook*


---

**Description**

Build a workbook from a data.frame or named list

**Usage**

```
buildWorkbook(x, asTable = FALSE, ...)
```

**Arguments**

x	A data.frame or a (named) list of objects that can be handled by <a href="#">writeData()</a> or <a href="#">writeDataTable()</a> to write to file
asTable	If TRUE will use <a href="#">writeDataTable()</a> rather than <a href="#">writeData()</a> to write x to the file (default: FALSE)
...	Additional arguments passed to <a href="#">writeData()</a> , <a href="#">writeDataTable()</a> , <a href="#">setColWidths()</a> (see <a href="#">Optional Parameters</a> )

**Details**

This function can be used as shortcut to create a workbook object from a data.frame or named list. If names are available in the list they will be used as the worksheet names. The parameters in ... are collected and passed to [writeData\(\)](#) or [writeDataTable\(\)](#) to initially create the Workbook objects then appropriate parameters are passed to [setColWidths\(\)](#).

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

**Value**

A Workbook object

**Optional Parameters****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 (or na.string, if not NULL) in Excel, else NA cells will be empty. Defaults to FALSE.

**na.string** If not NULL, and if keepNA is TRUE, NA values are converted to this string in Excel. Defaults to NULL.

**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** May be a single value for all columns (or "auto"), or a list of vectors that will be recycled for each sheet (see examples)

**Author(s)**

Jordan Mark Barbone

**See Also**

[write.xlsx\(\)](#)

**Examples**

```
x <- data.frame(a = 1, b = 2)
wb <- buildWorkbook(x)

y <- list(a = x, b = x, c = x)
buildWorkbook(y, asTable = TRUE)
buildWorkbook(y, asTable = TRUE, tableStyle = "TableStyleLight8")
```

---

cloneWorksheet

*Clone a worksheet to a workbook*

---

**Description**

Clone a worksheet to a Workbook object

**Usage**

```
cloneWorksheet(wb, sheetName, clonedSheet)
```

**Arguments**

wb	A Workbook object to attach the new worksheet
sheetName	A name for the new worksheet
clonedSheet	The name of the existing worksheet to be cloned.

**Value**

XML tree

**Author(s)**

Reinhold Kainhofer

**Examples**

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

## Add 3 worksheets
addWorksheet(wb, "Sheet 1")
cloneWorksheet(wb, "Sheet 2", clonedSheet = "Sheet 1")

## Save workbook
## Not run:
saveWorkbook(wb, "cloneWorksheetExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

col2int	<i>Convert Excel column to integer</i>
---------	--

---

**Description**

Converts an Excel column label to an integer.

**Usage**

```
col2int(x)
```

**Arguments**

x                    A character vector

**Examples**

```
col2int(LETTERS)
```

---

conditionalFormat	<i>Add conditional formatting to cells</i>
-------------------	--

---

**Description**

DEPRECATED! USE [conditionalFormatting\(\)](#)

**Usage**

```
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 <code>createStyle()</code>
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

**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 <code>createStyle(fontColour = "#9C0006", bgFill = "#FFC7CE")</code>
type	Either 'expression', 'colourScale', 'databar', 'duplicates', 'beginsWith', 'endsWith', 'topN', 'bottomN', 'blanks', 'notBlanks', 'contains' or 'notContains' (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)

If type == "blanks"

- style is a Style object. See [createStyle\(\)](#)
- rule is ignored.

If type == "notBlanks"

- style is a Style object. See [createStyle\(\)](#)
- rule is ignored.

If type == "topN"

- style is a Style object. See [createStyle\(\)](#)
- rule is ignored
- ...
  - **rank** numeric vector of length 1 indicating number of highest values.
  - **percent** TRUE if you want top N percentage.

If type == "bottomN"

- style is a Style object. See [createStyle\(\)](#)
- rule is ignored
- ...
  - **rank** numeric vector of length 1 indicating number of lowest values.
  - **percent** TRUE if you want bottom N percentage.

**Author(s)**

Alexander Walker, Philipp Schaubberger

**See Also**

[createStyle\(\)](#)

**Examples**

```

wb <- createWorkbook()
addWorksheet(wb, "cellIs")
addWorksheet(wb, "Moving Row")
addWorksheet(wb, "Moving Col")
addWorksheet(wb, "Dependent on")
addWorksheet(wb, "Duplicates")
addWorksheet(wb, "containsText")
addWorksheet(wb, "notcontainsText")
addWorksheet(wb, "beginsWith")
addWorksheet(wb, "endsWith")
addWorksheet(wb, "colourScale", zoom = 30)
addWorksheet(wb, "databar")
addWorksheet(wb, "between")
addWorksheet(wb, "topN")
addWorksheet(wb, "bottomN")
addWorksheet(wb, "containsBlanks")
addWorksheet(wb, "notContainsBlanks")
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", -5:5)
writeData(wb, "Dependent on", LETTERS[1:11], startCol = 2)
conditionalFormatting(wb, "Dependent on",
  cols = 1:2,
  rows = 1:11, rule = "$A$1<0", style = negStyle
)
conditionalFormatting(wb, "Dependent on",
  cols = 1:2,
  rows = 1:11, rule = "$A$1>0", style = posStyle
)

## highlight cells in column 1 based on value in column 2
writeData(wb, "Dependent on", data.frame(x = 1:10, y = runif(10)), startRow = 15)
conditionalFormatting(wb, "Dependent on",
  cols = 1,
  rows = 16:25, rule = "B16<0.5", style = negStyle
)
conditionalFormatting(wb, "Dependent on",
  cols = 1,
  rows = 16:25, rule = "B16>=0.5", 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")

## cells not containing text
fn <- function(x) paste(sample(LETTERS, 10), collapse = "-")
writeData(wb, "containsText", sapply(1:10, fn))
conditionalFormatting(wb, "notcontainsText", cols = 1,
  rows = 1:10, type = "notcontains", rule = "A")

```

```

## cells begins with text
fn <- function(x) paste(sample(LETTERS, 10), collapse = "-")
writeData(wb, "beginsWith", sapply(1:100, fn))
conditionalFormatting(wb, "beginsWith", cols = 1, rows = 1:100, type = "beginsWith", rule = "A")

## cells ends with text
fn <- function(x) paste(sample(LETTERS, 10), collapse = "-")
writeData(wb, "endsWith", sapply(1:100, fn))
conditionalFormatting(wb, "endsWith", cols = 1, rows = 1:100, type = "endsWith", rule = "A")

## colourscale colours cells based on cell value
df <- read.xlsx(system.file("extdata", "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))

## Top N
writeData(wb, "topN", data.frame(x = 1:10, y = rnorm(10)))
# Highlight top 5 values in column x
conditionalFormatting(wb, "topN", cols = 1, rows = 2:11,
  style = posStyle, type = "topN", rank = 5)#'
# Highlight top 20 percentage in column y
conditionalFormatting(wb, "topN", cols = 2, rows = 2:11,
  style = posStyle, type = "topN", rank = 20, percent = TRUE)

## Bottom N
writeData(wb, "bottomN", data.frame(x = 1:10, y = rnorm(10)))
# Highlight bottom 5 values in column x
conditionalFormatting(wb, "bottomN", cols = 1, rows = 2:11,
  style = negStyle, type = "topN", rank = 5)
# Highlight bottom 20 percentage in column y

```

```

conditionalFormatting(wb, "bottomN", cols = 2, rows = 2:11,
  style = negStyle, type = "topN", rank = 20, percent = TRUE)

## cells containing blanks
sample_data <- sample(c("X", NA_character_), 10, replace = TRUE)
writeData(wb, "containsBlanks", sample_data)
conditionalFormatting(wb, "containsBlanks", cols = 1, rows = 1:10,
  type = "blanks", style = negStyle)

## cells not containing blanks
sample_data <- sample(c("X", NA_character_), 10, replace = TRUE)
writeData(wb, "notContainsBlanks", sample_data)
conditionalFormatting(wb, "notContainsBlanks", cols = 1, rows = 1:10,
  type = "notBlanks", style = posStyle)

## Logical Operators
# You can use Excels 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)"
)
## Not run:
saveWorkbook(wb, "conditionalFormattingExample.xlsx", TRUE)

## End(Not run)

#####
## 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
)
## Not run:
saveWorkbook(wb, file = "databarExample.xlsx", overwrite = TRUE)

## End(Not run)

```

---

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

### Examples

```

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

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

```

---

convertToDate              *Convert from excel date number to R Date type*

---

### Description

Convert from excel date number to R Date type

### Usage

```
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	<i>Convert from excel time number to R POSIXct type.</i>
-------------------	--

---

**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**

```
## 2014-07-01, 2014-06-30, 2014-06-29
x <- c(41821.8127314815, 41820.8127314815, NA, 41819, NaN)
convertToDateTime(x)
convertToDateTime(x, tz = "Australia/Perth")
convertToDateTime(x, tz = "UTC")
```



---

copyWorkbook	<i>Copy a Workbook object.</i>
--------------	--------------------------------

---

**Description**

Just a wrapper of wb\$copy()

**Usage**

```
copyWorkbook(wb)
```

**Arguments**

wb	A workbook object
----	-------------------

**Value**

Workbook

**Examples**

```
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)
```

---

createComment	<i>create a Comment object</i>
---------------	--------------------------------

---

**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 <a href="#">createStyle()</a> .
visible	TRUE or FALSE. Is comment visible.
width, height	Width and height of text box (in number of cells); doubles are rounded with <code>base::round()</code>

**See Also**

[writeComment\(\)](#)

**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)
## Not run:
saveWorkbook(wb, file = "createCommentExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

createNamedRegion      *Create / delete a named region.*

---

**Description**

Create / delete a named region

**Usage**

```
createNamedRegion(wb, sheet, cols, rows, name, overwrite = FALSE)

deleteNamedRegion(wb, 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.
overwrite	Boolean. Overwrite if exists ? Default to FALSE

**Details**

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

**Author(s)**

Alexander Walker

**See Also**

[getNamedRegions\(\)](#)

**Examples**

```
## 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")
## Not run:
saveWorkbook(wb, out_file, overwrite = TRUE)

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

## delete one
```

```

deleteNamedRegion(wb = wb, name = "iris2")
getNamedRegions(wb)

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

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

## End(Not run)

```

---

createStyle

*Create a cell style*


---

## Description

Create a new style to apply to worksheet cells

## Usage

```

createStyle(
  fontName = NULL,
  fontSize = NULL,
  fontColour = NULL,
  numFmt = openxlsx_getOp("numFmt", "GENERAL"),
  border = NULL,
  borderColour = openxlsx_getOp("borderColour", "black"),
  borderStyle = openxlsx_getOp("borderStyle", "thin"),
  bgFill = NULL,
  fgFill = NULL,
  halign = NULL,
  valign = NULL,
  textDecoration = NULL,
  wrapText = FALSE,
  textRotation = NULL,
  indent = NULL,
  locked = NULL,
  hidden = 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	<p>Cell formatting</p> <ul style="list-style-type: none"> <li>• <b>GENERAL</b></li> <li>• <b>NUMBER</b></li> <li>• <b>CURRENCY</b></li> <li>• <b>ACCOUNTING</b></li> <li>• <b>DATE</b></li> <li>• <b>LONGDATE</b></li> <li>• <b>TIME</b></li> <li>• <b>PERCENTAGE</b></li> <li>• <b>FRACTION</b></li> <li>• <b>SCIENTIFIC</b></li> <li>• <b>TEXT</b></li> <li>• <b>COMMA</b> for comma separated thousands</li> <li>• For date/datetime styling a combination of d, m, y and punctuation marks</li> <li>• For numeric rounding use "0.00" with the preferred number of decimal places</li> </ul>
border	<p>Cell border. A vector of "top", "bottom", "left", "right" or a single string).</p> <p><b>"top"</b> Top border  <b>bottom</b> Bottom border  <b>left</b> Left border  <b>right</b> Right border  <b>TopBottom</b> or c("top", "bottom") Top and bottom border  <b>LeftRight</b> or c("left", "right") Left and right border  <b>TopLeftRight</b> or c("top", "left", "right") Top, Left and right border  <b>TopBottomLeftRight</b> or c("top", "bottom", "left", "right") All borders</p>
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	<p>Border line style vector the same length as the number of sides specified in "border"</p> <p><b>none</b> No Border  <b>thin</b> thin border  <b>medium</b> medium border  <b>dashed</b> dashed border  <b>dotted</b> dotted border  <b>thick</b> thick border  <b>double</b> double line border  <b>hair</b> Hairline border  <b>mediumDashed</b> medium weight dashed border</p>

	<b>dashDot</b> dash-dot border
	<b>mediumDashDot</b> medium weight dash-dot border
	<b>dashDotDot</b> dash-dot-dot border
	<b>mediumDashDotDot</b> medium weight dash-dot-dot border
	<b>slantDashDot</b> slanted dash-dot border
bgFill	Cell background fill colour. A valid colour (belonging to colours()) or a valid hex colour beginning with "#". – <b>Use for conditional formatting styles only.</b>
fgFill	Cell foreground fill colour. A valid colour (belonging to colours()) or a valid hex colour beginning with "#"
halign	Horizontal alignment of cell contents <b>left</b> Left horizontal align cell contents <b>right</b> Right horizontal align cell contents <b>center</b> Center horizontal align cell contents <b>justify</b> Justify horizontal align cell contents
valign	A name Vertical alignment of cell contents <b>top</b> Top vertical align cell contents <b>center</b> Center vertical align cell contents <b>bottom</b> Bottom vertical align cell contents
textDecoration	Text styling. <b>bold</b> Bold cell contents <b>strikeout</b> Strikeout cell contents <b>italic</b> Italicise cell contents <b>underline</b> Underline cell contents <b>underline2</b> Double underline cell contents <b>accounting</b> Single accounting underline cell contents <b>accounting2</b> Double accounting 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.
locked	Whether cell contents are locked (if worksheet protection is turned on)
hidden	Whether the formula of the cell contents will be hidden (if worksheet protection is turned on)

**Value**

A style object

**Author(s)**

Alexander Walker

**See Also**

[addStyle\(\)](#)

**Examples**

```
## See package vignettes for further examples

## Modify default values of border colour and border line style
options("openxlsx.borderColour" = "#4F80BD")
options("openxlsx.borderStyle" = "thin")

## Size 18 Arial, Bold, left horz. aligned, fill colour #1A33CC, all borders,
style <- createStyle(
  fontSize = 18, fontName = "Arial",
  textDecoration = "bold", halign = "left", fgFill = "#1A33CC", border = "TopBottomLeftRight"
)

## Red, size 24, Bold, italic, underline, center aligned Font, bottom border
style <- createStyle(
  fontSize = 24, fontColour = rgb(1, 0, 0),
  textDecoration = c("bold", "italic", "underline"),
  halign = "center", valign = "center", border = "Bottom"
)

# borderColour is recycled for each border or all colours can be supplied

# colour is recycled 3 times for "Top", "Bottom" & "Right" sides.
createStyle(border = "TopBottomRight", borderColour = "red")

# supply all colours
createStyle(border = "TopBottomLeft", borderColour = c("red", "yellow", "green"))
```

---

<code>createWorkbook</code>	<i>Create a new Workbook object</i>
-----------------------------	-------------------------------------

---

**Description**

Create a new Workbook object

**Usage**

```
createWorkbook(
  creator = ifelse(.Platform$OS.type == "windows", Sys.getenv("USERNAME"),
    Sys.getenv("USER")),
  title = NULL,
  subject = NULL,
  category = NULL
)
```

**Arguments**

<code>creator</code>	Creator of the workbook (your name). Defaults to login username
<code>title</code>	Workbook properties title

subject	Workbook properties subject
category	Workbook properties category

**Value**

Workbook object

**Author(s)**

Alexander Walker

**See Also**

[loadWorkbook\(\)](#)

[saveWorkbook\(\)](#)

**Examples**

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

## Save workbook to working directory
## Not run:
saveWorkbook(wb, file = "createWorkbookExample.xlsx", overwrite = TRUE)

## End(Not run)

## Set Workbook properties
wb <- createWorkbook(
  creator = "Me",
  title = "title here",
  subject = "this & that",
  category = "something"
)
```

---

dataValidation

*Add data validation to cells*

---

**Description**

Add Excel data validation to cells

**Usage**

```
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	Contiguous columns to apply conditional formatting to
rows	Contiguous rows to apply conditional formatting to
type	One of 'whole', 'decimal', 'date', 'time', 'textLength', 'list' (see examples)
operator	One of 'between', 'notBetween', 'equal', 'notEqual', 'greaterThan', 'lessThan', 'greaterThanOrEqual', 'lessThanOrEqual'
value	a vector of length 1 or 2 depending on operator (see examples)
allowBlank	logical
showInputMsg	logical
showErrorMsg	logical

### Examples

```

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

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

dataValidation(wb, 1,
  col = 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 = 1, 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$t[c(4, 8)])
)
## Not run:
saveWorkbook(wb, "dataValidationExample.xlsx", overwrite = TRUE)

## End(Not run)

#####
## 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.

### Usage

```
deleteData(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)
## Not run:
saveWorkbook(wb, "deleteDataExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

deleteDataColumn	<i>Deletes a whole column from a workbook</i>
------------------	---

---

**Description**

Deletes the whole column from a workbook, shifting the remaining columns to the left

**Usage**

```
deleteDataColumn(wb, sheet, col)
```

**Arguments**

wb	A workbook object
sheet	A name or index of a worksheet
col	A column to delete

**Author(s)**

David Zimmermann

**Examples**

```
## write some data
wb <- createWorkbook()
addWorksheet(wb, "tester")

for (i in seq(5)) {
```

```

mat <- data.frame(x = rep(paste0(int2col(i), i), 10))
writeData(wb, sheet = 1, startRow = 1, startCol = i, mat)
writeFormula(wb, sheet = 1, startRow = 12, startCol = i,
             x = sprintf("=COUNTA(%s2:%s11)", int2col(i), int2col(i)))
}
deleteDataColumn(wb, 1, col = 3)
## Not run:
saveWorkbook(wb, "deleteDataColumnExample.xlsx", overwrite = TRUE)

## End(Not run)

```

---

freezePane

*Freeze a worksheet pane*


---

### Description

Freeze a worksheet pane

### Usage

```

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

```

### 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
## Not run:
saveWorkbook(wb, "freezePaneExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

getBaseFont	<i>Return the workbook default font</i>
-------------	---

---

**Description**

Return the workbook default font  
Returns the base font used in the workbook.

**Usage**

```
getBaseFont(wb)
```

**Arguments**

wb                    A workbook object

**Author(s)**

Alexander Walker

**Examples**

```
## 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

```
getCellRefs(cellCoords)
```

### Arguments

cellCoords            A data.frame with two columns coordinate pairs.

### Value

Excel alphanumeric cell reference

### Author(s)

Philipp Schauburger, Alexander Walker

### Examples

```
getCellRefs(data.frame(1, 2))  
# "B1"  
getCellRefs(data.frame(1:3, 2:4))  
# "B1" "C2" "D3"
```

---

getCreators                      *Add another author to the meta data of the file.*

---

### Description

Just a wrapper of wb\$getCreators() Get the names of the

### Usage

```
getCreators(wb)
```

### Arguments

wb                      A workbook object

**Value**

vector of creators

**Author(s)**

Philipp Schauburger

**Examples**

```
wb <- createWorkbook()  
getCreators(wb)
```

---

`getDateOrigin`

*Get the date origin an `xlsx` file is using*

---

**Description**

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

**Usage**

```
getDateOrigin(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
## Not run:
write.xlsx(as.Date("2015-01-10") - (0:4), file = "getDateOriginExample.xlsx")
m <- read.xlsx("getDateOriginExample.xlsx")

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

## End(Not run)
```

---

getNamedRegions	<i>Get named regions</i>
-----------------	--------------------------

---

**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**

```
## 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)
## Not run:
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)

## End(Not run)
```

---

getSheetNames

*Get names of worksheets*

---

### **Description**

Returns the worksheet names within an xlsx file

### **Usage**

```
getSheetNames(file)
```

### **Arguments**

file            An xlsx or xlsxm file.

### **Value**

Character vector of worksheet names.

### **Author(s)**

Alexander Walker

### **Examples**

```
getSheetNames(system.file("extdata", "readTest.xlsx", package = "openxlsx"))
```

---

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

---

**Description**

Returns list of style objects in the workbook

**Usage**

```
getStyles(wb)
```

**Arguments**

wb	A workbook object
----	-------------------

**See Also**

[replaceStyle\(\)](#)

**Examples**

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

---

getTables	List Excel tables in a workbook
-----------	---------------------------------

---

**Description**

List Excel tables in a workbook

**Usage**

```
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

```
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")
```

---

get\_worksheet\_entries *Get entries from workbook worksheet*

---

## Description

Get all entries from workbook worksheet without xml tags

## Usage

```
get_worksheet_entries(wb, sheet)
```

## Arguments

wb	workbook
sheet	worksheet

## Value

vector of strings

## Author(s)

David Breuer

## Examples

```
## Create new workbook
wb <- createWorkbook()
addWorksheet(wb, "Sheet")
sheet <- 1

## Write dummy data
writeData(wb, sheet, c("A", "BB", "CCC"), startCol = 2, startRow = 3)
writeData(wb, sheet, c(4, 5), startCol = 4, startRow = 3)

## Get text entries
get_worksheet_entries(wb, sheet)
```

---

groupColumns	<i>Group columns</i>
--------------	----------------------

---

**Description**

Group a selection of columns

**Usage**

```
groupColumns(wb, sheet, cols, hidden = FALSE, level = -1)
```

**Arguments**

wb	A workbook object.
sheet	A name or index of a worksheet.
cols	Indices of cols to group. Can be either a vector of indices to group at the same level or a (named) list of numeric vectors of indices to create multiple groupings at once. The names of the entries determine the grouping level. If no names are given, the level parameter is used as default.
hidden	Logical vector. If TRUE the grouped columns are hidden. Defaults to FALSE.
level	Grouping level (higher value indicates multiple nestings) for the group. A vector to assign different grouping levels to the indices. A value of -1 indicates that the grouping level should be derived from the existing grouping (one level added)

**Details**

Group columns together, with the option to hide them.

NOTE: [setColWidths\(\)](#) has a conflicting hidden parameter; changing one will update the other.

**Author(s)**

Joshua Sturm, Reinhold Kainhofer

**See Also**

[ungroupColumns\(\)](#) to ungroup columns. [groupRows\(\)](#) for grouping rows.

---

`groupRows`*Group Rows*

---

**Description**

Group a selection of rows

**Usage**

```
groupRows(wb, sheet, rows, hidden = FALSE, level = -1)
```

**Arguments**

<code>wb</code>	A workbook object
<code>sheet</code>	A name or index of a worksheet
<code>rows</code>	Indices of rows to group. Can be either a vector of indices to group at the same level or a (named) list of numeric vectors of indices to create multiple groupings at once. The names of the entries determine the grouping level. If no names are given, the parameter <code>level</code> is used as default.
<code>hidden</code>	Logical vector. If TRUE the grouped columns are hidden. Defaults to FALSE
<code>level</code>	Grouping level (higher value indicates multiple nestings) for the group. A vector to assign different grouping levels to the indices. A value of -1 indicates that the grouping level should be derived from the existing grouping (one level added)

**Author(s)**

Joshua Sturm

**See Also**

[ungroupRows\(\)](#) to ungroup rows. [groupColumns\(\)](#) for grouping columns.

**Examples**

```
wb <- createWorkbook()
addWorksheet(wb, 'Sheet1')
addWorksheet(wb, 'Sheet2')

writeData(wb, "Sheet1", iris)
writeData(wb, "Sheet2", iris)

## create list of groups
# lines used for grouping (here: species)
grp <- list(
  seq(2, 51),
  seq(52, 101),
  seq(102, 151)
```

```

)
# assign group levels
names(grp) <- c("1","0","1")
groupRows(wb, "Sheet1", rows = grp)

# different grouping
names(grp) <- c("1","2","3")
groupRows(wb, "Sheet2", rows = grp)

# alternatively, one can call groupRows multiple times
addWorksheet(wb, 'Sheet3')
writeData(wb, "Sheet3", iris)
groupRows(wb, "Sheet3", 2:51, level = 1)
groupRows(wb, "Sheet3", 102:151, level = 1)

addWorksheet(wb, 'Sheet4')
writeData(wb, "Sheet4", iris)
groupRows(wb, "Sheet4", 2:51, level = 1)
groupRows(wb, "Sheet4", 52:101, level = 2)
groupRows(wb, "Sheet4", 102:151, level = 3)

# Nested grouping can also be achieved without explicitly given the levels
addWorksheet(wb, 'Sheet5')
writeData(wb, "Sheet5", iris)
groupRows(wb, "Sheet5", 2:151)
groupRows(wb, "Sheet5", 52:151)
groupRows(wb, "Sheet5", 102:151)

```

---

if\_null\_then

*If NULL then ...*


---

### Description

Replace NULL

### Usage

```
x %||% y
```

### Arguments

x	A value to check
y	A value to substitute if x is null

**Examples**

```
## Not run:
x <- NULL
x <- x %||% "none"
x <- x %||% NA

## End(Not run)
```

---

**insertImage***Insert an image into a worksheet*

---

**Description**

Insert an image into a worksheet

**Usage**

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

**Arguments**

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

**Author(s)**

Alexander Walker

**See Also**[insertPlot\(\)](#)**Examples**

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

## Save workbook
## Not run:
saveWorkbook(wb, "insertImageExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

`insertPlot`*Insert the current plot into a worksheet*

---

**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**

```
## 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, fileType = "png", units = "cm")
```

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

## End(Not run)
```

---

int2col	<i>Convert integer to Excel column</i>
---------	--

---

**Description**

Converts an integer to an Excel column label.

**Usage**

```
int2col(x)
```

**Arguments**

x	A numeric vector
---	------------------

**Examples**

```
int2col(1:10)
```

---

loadWorkbook	<i>Load an existing .xlsx file</i>
--------------	------------------------------------

---

**Description**

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

**Usage**

```
loadWorkbook(file, xlsxFile = NULL, isUnzipped = FALSE, na.convert = TRUE)
```

**Arguments**

file	A path to an existing .xlsx or .xlsm file
xlsxFile	alias for file
isUnzipped	Set to TRUE if the xlsx file is already unzipped
na.convert	Should empty/blank cells be converted to NA_character_. Defaults to TRUE.

**Value**

Workbook object.

**Author(s)**

Alexander Walker, Philipp Schaubberger

**See Also**

[removeWorksheet\(\)](#)

**Examples**

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

## Save workbook
## Not run:
saveWorkbook(wb, "loadExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

makeHyperlinkString     *create Excel hyperlink string*

---

**Description**

Wrapper to create internal hyperlink string to pass to writeFormula(). Either link to external urls or local files or straight to cells of local Excel sheets.

**Usage**

```
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**

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

## External Hyperlink
x <- c("https://www.google.com", "https://www.google.com.au")
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)
)

## Internal - Text to display
writeFormula(
  wb, "Sheet1",
  startRow = 2,
  x = makeHyperlinkString(
    sheet = "Sheet 3", row = 1, col = 2,
    text = "Link to Sheet 3"
  )
)

## Link to file - No text to display
writeFormula(
  wb, "Sheet1",
  startRow = 4,
  x = makeHyperlinkString(
    sheet = "testing", row = 3, col = 10,
    file = system.file("extdata", "loadExample.xlsx", package = "openxlsx")
  )
)

## Link to file - Text to display
writeFormula(
```

```

wb, "Sheet1",
startRow = 3,
x = makeHyperlinkString(
  sheet = "testing", row = 3, col = 10,
  file = system.file("extdata", "loadExample.xlsx", package = "openxlsx"),
  text = "Link to File."
)
)

## Link to external file - Text to display
writeFormula(
  wb, "Sheet1",
  startRow = 10, startCol = 1,
  x = '=HYPERLINK("[C:/Users]", "Link to an external file")'
)

## Link to internal file
x = makeHyperlinkString(text = "test.png", file = "D:/somepath/somepicture.png")
writeFormula(wb, "Sheet1", startRow = 11, startCol = 1, x = x)

## Not run:
saveWorkbook(wb, "internalHyperlinks.xlsx", overwrite = TRUE)

## End(Not run)

```

---

mergeCells

---

*Merge cells within a worksheet*


---

### 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\(\)](#)

**Examples**

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

## Add a worksheet
addWorksheet(wb, "Sheet 1")
addWorksheet(wb, "Sheet 2")

## Merge cells: Row 2 column C to F (3:6)
mergeCells(wb, "Sheet 1", cols = 2, rows = 3:6)

## Merge cells: Rows 10 to 20 columns A to J (1:10)
mergeCells(wb, 1, cols = 1:10, rows = 10:20)

## Intersecting merges
mergeCells(wb, 2, cols = 1:10, rows = 1)
mergeCells(wb, 2, cols = 5:10, rows = 2)
mergeCells(wb, 2, cols = c(1, 10), rows = 12) ## equivalent to 1:10 as only min/max are used
# mergeCells(wb, 2, cols = 1, rows = c(1,10)) # Throws error because intersects existing merge

## remove merged cells
removeCellMerge(wb, 2, cols = 1, rows = 1) # removes any intersecting merges
mergeCells(wb, 2, cols = 1, rows = 1:10) # Now this works

## Save workbook
## Not run:
saveWorkbook(wb, "mergeCellsExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

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 anyway. Excel replaces unknown font names with Arial. Base font is black, size 11, Calibri.

**Author(s)**

Alexander Walker

**Examples**

```
## 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
## Not run:
saveWorkbook(wb, "modifyBaseFontExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

names	<i>get or set worksheet names</i>
-------	-----------------------------------

---

**Description**

get or set worksheet names

**Usage**

```
## 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**

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

names(wb)
names(wb)[[2]] <- "S2a"
names(wb)
names(wb) <- paste("Sheet", 1:3)
```

---

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.excelApp')` is set to the app bin path), and if it finds anything, sets `options('openxlsx.excelApp')` to the program chosen 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**

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

**Arguments**

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

**Author(s)**

Luca Braglia



**Examples**

```
# file example
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 the process of writing and styling Excel xlsx files from R and removes the dependency on Java.

**Details**

The openxlsx package uses global options, most to simplify formatting. These are stored in the `op.openxlsx` object.

```
openxlsx.bandedCols FALSE
openxlsx.bandedRows TRUE
openxlsx.borderColour "black"
openxlsx.borders "none"
openxlsx.borderStyle "thin"
openxlsx.compressionLevel "9"
openxlsx.creator ""
openxlsx.dateFormat "mm/dd/yyyy"
openxlsx.datetimeFormat "yyyy-mm-dd hh:mm:ss"
openxlsx.headerStyle NULL
openxlsx.keepNA FALSE
openxlsx.na.string NULL
openxlsx.numFmt NULL
openxlsx.orientation "portrait"
openxlsx.paperSize 9
openxlsx.tabColour "TableStyleLight9"
openxlsx.tableStyle "TableStyleLight9"
```

**openxlsx.withFilter** NA Whether to write data with or without a filter. If NA will make filters with `writeDataTable` and will not for `writeData`

See the Formatting vignette for examples.

Additional options

### Author(s)

**Maintainer:** Jan Marvin Garbuszus <jan.garbuszus@ruhr-uni-bochum.de> [contributor]

Authors:

- Philipp Schauburger <philipp@schauburger.co.at>
- Alexander Walker <Alexander.Walker1989@gmail.com>

Other contributors:

- Luca Braglia [contributor]
- Joshua Sturm [contributor]
- Jordan Mark Barbone <jmbarbone@gmail.com> (**ORCID**) [contributor]
- David Zimmermann <david\_j\_zimmermann@hotmail.com> [contributor]
- Reinhold Kainhofer <reinhold@kainhofer.com> [contributor]

### See Also

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

for examples

---

openxlsxFontSizeLookupTable

*Font Size Lookup tables*

---

### Description

Lookup tables for font size

### Usage

`openxlsxFontSizeLookupTable`

`openxlsxFontSizeLookupTableBold`

**Format**

A data.frame with column names corresponding to font names

**Examples**

```
data(openxlsxFontSizeLookupTable)
data(openxlsxFontSizeLookupTableBold)
```

---

openxlsx_options	<i>openxlsx Options</i>
------------------	-------------------------

---

**Description**

See and get the openxlsx options

**Usage**

```
op.openxlsx
openxlsx_getOp(x, default = NULL)
openxlsx_setOp(x, value)
```

**Arguments**

x	An option name ("openxlsx." prefix optional)
default	A default value if NULL
value	The new value for the option (optional if x is a named list)

**Format**

An object of class list of length 34.

**Details**

openxlsx\_getOp() retrieves the "openxlsx" options found in op.openxlsx. If none are set (currently NULL) retrieves the default option from op.openxlsx. This will also check that the intended option is a standard option (listed in op.openxlsx) and will provide a warning otherwise.

openxlsx\_setOp() is a safer way to set an option as it will first check that the option is a standard option (as above) before setting.

**Examples**

```
openxlsx_getOp("borders")
op.openxlsx[["openxlsx.borders"]]
```

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")
## Not run:
saveWorkbook(wb, "pageBreakExample.xlsx", TRUE)

## End(Not run)
## 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,
  summaryRow = NULL,
  summaryCol = NULL
)

```

**Arguments**

wb	A workbook object
sheet	A name or index of a worksheet
orientation	Page orientation. One of "portrait" or "landscape"
scale	Print scaling. Numeric value between 10 and 400
left	left page margin in inches
right	right page margin in inches
top	top page margin in inches
bottom	bottom page margin in inches
header	header margin in inches
footer	footer margin in inches
fitToWidth	If TRUE, worksheet is scaled to fit to page width on printing.
fitToHeight	If TRUE, worksheet is scaled to fit to page height on printing.
paperSize	See details. Default value is 9 (A4 paper).
printTitleRows	Rows to repeat at top of page when printing. Integer vector.
printTitleCols	Columns to repeat at left when printing. Integer vector.
summaryRow	Location of summary rows in groupings. One of "Above" or "Below".
summaryCol	Location of summary columns in groupings. One of "Right" or "Left".

**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)
- 50 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, Joshua Sturm

**Examples**

```

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)
## Not run:
saveWorkbook(wb, "pageSetupExample.xlsx", overwrite = TRUE)

## End(Not run)

```

---

protectWorkbook

*Protect a workbook from modifications*


---

**Description**

Protect or unprotect a workbook from modifications by the user in the graphical user interface. Replaces an existing protection.

**Usage**

```

protectWorkbook(
  wb,
  protect = TRUE,
  password = NULL,
  lockStructure = FALSE,
  lockWindows = FALSE,
  type = 1L
)

```



**Arguments**

wb	A workbook object
protect	Whether to protect or unprotect the sheet (default=TRUE)
password	(optional) password required to unprotect the workbook
lockStructure	Whether the workbook structure should be locked
lockWindows	Whether the window position of the spreadsheet should be locked
type	Lock type, default 1. From the xml documentation: 1 - Document is password protected. 2 - Document is recommended to be opened as read-only. 4 - Document is enforced to be opened as read-only. 8 - Document is locked for annotation.

**Author(s)**

Reinhold Kainhofer

**Examples**

```
wb <- createWorkbook()
addWorksheet(wb, "S1")
protectWorkbook(wb, protect = TRUE, password = "Password", lockStructure = TRUE)
## Not run:
saveWorkbook(wb, "WorkBook_Protection.xlsx", overwrite = TRUE)

## End(Not run)
# Remove the protection
protectWorkbook(wb, protect = FALSE)
## Not run:
saveWorkbook(wb, "WorkBook_Protection_unprotected.xlsx", overwrite = TRUE)

## End(Not run)
```

---

protectWorksheet	<i>Protect a worksheet from modifications</i>
------------------	---

---

**Description**

Protect or unprotect a worksheet from modifications by the user in the graphical user interface. Replaces an existing protection.

**Usage**

```
protectWorksheet(
  wb,
  sheet,
  protect = TRUE,
  password = NULL,
```

```

lockSelectingLockedCells = NULL,
lockSelectingUnlockedCells = NULL,
lockFormattingCells = NULL,
lockFormattingColumns = NULL,
lockFormattingRows = NULL,
lockInsertingColumns = NULL,
lockInsertingRows = NULL,
lockInsertingHyperlinks = NULL,
lockDeletingColumns = NULL,
lockDeletingRows = NULL,
lockSorting = NULL,
lockAutoFilter = NULL,
lockPivotTables = NULL,
lockObjects = NULL,
lockScenarios = NULL
)

```

### **Arguments**

<code>wb</code>	A workbook object
<code>sheet</code>	A name or index of a worksheet
<code>protect</code>	Whether to protect or unprotect the sheet (default=TRUE)
<code>password</code>	(optional) password required to unprotect the worksheet
<code>lockSelectingLockedCells</code>	Whether selecting locked cells is locked
<code>lockSelectingUnlockedCells</code>	Whether selecting unlocked cells is locked
<code>lockFormattingCells</code>	Whether formatting cells is locked
<code>lockFormattingColumns</code>	Whether formatting columns is locked
<code>lockFormattingRows</code>	Whether formatting rows is locked
<code>lockInsertingColumns</code>	Whether inserting columns is locked
<code>lockInsertingRows</code>	Whether inserting rows is locked
<code>lockInsertingHyperlinks</code>	Whether inserting hyperlinks is locked
<code>lockDeletingColumns</code>	Whether deleting columns is locked
<code>lockDeletingRows</code>	Whether deleting rows is locked
<code>lockSorting</code>	Whether sorting is locked
<code>lockAutoFilter</code>	Whether auto-filter is locked

```
lockPivotTables      Whether pivot tables are locked
lockObjects          Whether objects are locked
lockScenarios        Whether scenarios are locked
```

**Author(s)**

Reinhold Kainhofer

**Examples**

```
wb <- createWorkbook()
addWorksheet(wb, "S1")
writeDataTable(wb, 1, x = iris[1:30, ])
# Formatting cells / columns is allowed , but inserting / deleting columns is protected:
protectWorksheet(wb, "S1",
  protect = TRUE,
  lockFormattingCells = FALSE, lockFormattingColumns = FALSE,
  lockInsertingColumns = TRUE, lockDeletingColumns = TRUE
)

# Remove the protection
protectWorksheet(wb, "S1", protect = FALSE)
## Not run:
saveWorkbook(wb, "pageSetupExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

read.xlsx

*Read from an Excel file or Workbook object*

---

**Description**

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

**Usage**

```
read.xlsx(
  xlsxFile,
  sheet,
  startRow = 1,
  colNames = TRUE,
  rowNames = FALSE,
  detectDates = FALSE,
  skipEmptyRows = TRUE,
  skipEmptyCols = TRUE,
  rows = NULL,
  cols = NULL,
```

```

    check.names = FALSE,
    sep.names = ".",
    namedRegion = NULL,
    na.strings = "NA",
    fillMergedCells = FALSE
  )

```

### Arguments

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

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

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

## URL
##
## Not run:
xlsxFile <- "https://github.com/awalker89/openxlsx/raw/master/inst/readTest.xlsx"
head(read.xlsx(xlsxFile))

## End(Not run)
```

---

readWorkbook

*Read from an Excel file or Workbook object*

---

**Description**

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

**Usage**

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

**Arguments**

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

**See Also**

[getNamedRegions\(\)](#)

[read.xlsx\(\)](#)

**Examples**

```
xlsxFile <- system.file("extdata", "readTest.xlsx", package = "openxlsx")
df1 <- readWorkbook(xlsxFile = xlsxFile, sheet = 1)
```

```
xlsxFile <- system.file("extdata", "readTest.xlsx", package = "openxlsx")
df1 <- readWorkbook(xlsxFile = xlsxFile, sheet = 1, rows = c(1, 3, 5), cols = 1:3)
```

---

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**

```
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\(\)](#)

---

removeColWidths	<i>Remove column widths from a worksheet</i>
-----------------	--

---

**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("extdata", "loadExample.xlsx", package = "openxlsx"))

## remove column widths in columns 1 to 20
removeColWidths(wb, 1, cols = 1:20)
## Not run:
saveWorkbook(wb, "removeColWidthsExample.xlsx", overwrite = TRUE)

## End(Not run)
```



---

removeComment	<i>Remove a comment from a cell</i>
---------------	-------------------------------------

---

**Description**

Remove a cell comment from a worksheet

**Usage**

```
removeComment(wb, sheet, cols, rows, gridExpand = TRUE)
```

**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\(\)](#)

---

removeFilter	<i>Remove a worksheet filter</i>
--------------	----------------------------------

---

**Description**

Removes filters from `addFilter()` and `writeData()`

**Usage**

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

**Arguments**

wb	A workbook object
sheet	A vector of names or indices of worksheets

### Examples

```
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
removeFilter(wb, 1:2) ## remove filters
removeFilter(wb, 3) ## Does not affect tables!
## Not run:
saveWorkbook(wb, file = "removeFilterExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

removeRowHeights	<i>Remove custom row heights from a worksheet</i>
------------------	---

---

### 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("extdata", "loadExample.xlsx", package = "openxlsx"))

## remove any custom row heights in rows 1 to 10
removeRowHeights(wb, 1, rows = 1:10)
## Not run:
saveWorkbook(wb, "removeRowHeightsExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

removeTable	<i>Remove an Excel table in a workbook</i>
-------------	--

---

## Description

List Excel tables in a workbook

## Usage

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

## Arguments

wb	A workbook object
sheet	A name or index of a worksheet
table	Name of table to remove. See <a href="#">getTables()</a>

## Value

character vector of table names on the specified sheet

## Examples

```
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)
## Not run:
saveWorkbook(wb = wb, file = "removeTableExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

removeWorksheet

*Remove a worksheet from a workbook*

---

### Description

Remove a worksheet from a Workbook object

Remove a worksheet from a workbook

### Usage

```
removeWorksheet(wb, sheet)
```

### Arguments

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

### Author(s)

Alexander Walker

### Examples

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

## Remove sheet 2
removeWorksheet(wb, 2)

## save the modified workbook
## Not run:
saveWorkbook(wb, "removeWorksheetExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

renameWorksheet	<i>Rename a worksheet</i>
-----------------	---------------------------

---

### Description

Rename a worksheet

### Usage

```
renameWorksheet(wb, sheet, newName)
```

### 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
## Not run:
saveWorkbook(wb, "renameWorksheetExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

replaceStyle	<i>Replace an existing cell style</i>
--------------	---------------------------------------

---

**Description**

Replace an existing cell style

Replace a style object

**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**

```
## load a workbook
wb <- loadWorkbook(file = system.file("extdata", "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
## Not run:
saveWorkbook(wb, "replaceStyleExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

saveWorkbook	<i>save Workbook to file</i>
--------------	------------------------------

---

**Description**

save a Workbook object to file

**Usage**

```
saveWorkbook(wb, file, overwrite = FALSE, returnValue = 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.
returnValue	If TRUE, returns TRUE in case of a success, else FALSE. If flag is FALSE, then no return value is returned.

**Author(s)**

Alexander Walker, Philipp Schauburger

**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  
## Not run:  
saveWorkbook(wb, file = "saveWorkbookExample.xlsx", overwrite = TRUE)  
  
## End(Not run)
```

---

 setColWidths

*Set worksheet column widths*


---

**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.

NOTE: The hidden parameter may conflict with the one set in `groupColumns`; changing one will update the other.

**Author(s)**

Alexander Walker



**See Also**[removeColWidths\(\)](#)**Examples**

```
## 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
## Not run:
saveWorkbook(wb, "setColWidthsExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

`setFooter`*Set footer for all worksheets*

---

**Description**

DEPRECATED

**Usage**

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

**Arguments**

<code>wb</code>	A workbook object
<code>text</code>	footer text. A character vector of length 1.
<code>position</code>	Position of text in footer. One of "left", "center" or "right"

**Author(s)**

Alexander Walker

**Examples**

```
## 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)
```

---

setHeader

*Set header for all worksheets*


---

**Description**

DEPRECATED

**Usage**

```
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**

```
## 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	<i>Set document headers and footers</i>
-----------------	---

---

## 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.
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**

```
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 LEFT"),
  evenHeader = c("EVEN HEAD LEFT", "EVEN HEAD CENTER", "EVEN HEAD RIGHT"),
  evenFooter = c("EVEN FOOT RIGHT", "EVEN FOOT CENTER", "EVEN FOOT LEFT"),
  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")
)
## Not run:
saveWorkbook(wb, "setHeaderFooterExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

setLastModifiedBy      *Add another author to the meta data of the file.*

---

### Description

Just a wrapper of `wb$changeLastModifiedBy()`

### Usage

```
setLastModifiedBy(wb, LastModifiedBy)
```

### Arguments

`wb`                    A workbook object  
`LastModifiedBy`      A string object with the name of the LastModifiedBy-User

### Author(s)

Philipp Schauburger

### Examples

```
wb <- createWorkbook()
setLastModifiedBy(wb, "test")
```

---

`setRowHeights`*Set worksheet row heights*

---

**Description**

Set worksheet row heights

**Usage**

```
setRowHeights(  
    wb,  
    sheet,  
    rows,  
    heights,  
    fontsize = NULL,  
    factor = 1,  
    base_height = 15,  
    extra_height = 12,  
    wrap = TRUE  
)
```

**Arguments**

<code>wb</code>	workbook object
<code>sheet</code>	name or index of a worksheet
<code>rows</code>	indices of rows to set height
<code>heights</code>	heights to set rows to specified in Excel column height units
<code>fontsize</code>	font size, optional (get base font size by default)
<code>factor</code>	factor to manually adjust font width, e.g., for bold fonts, optional
<code>base_height</code>	basic row height, optional
<code>extra_height</code>	additional row height per new line of text, optional
<code>wrap</code>	wrap text of entries which exceed the column width, optional

**Author(s)**

Alexander Walker

**See Also**

[removeRowHeights\(\)](#)

## Examples

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

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

## Write dummy data
writeData(wb, sheet, "fixed w/fixed h", startCol = 1, startRow = 1)
writeData(wb, sheet, "fixed w/auto h ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC",
  startCol = 2, startRow = 2)
writeData(wb, sheet, "variable w/fixed h", startCol = 3, startRow = 3)

## Set column widths and row heights
setColWidths(wb, sheet, cols = c(1, 2, 3, 4), widths = c(10, 20, "auto", 20))
setRowHeights(wb, sheet, rows = c(1, 2, 8, 4, 6), heights = c(30, "auto", 15, 15, 30))

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

## Save workbook
## Not run:
saveWorkbook(wb, "setRowHeightsExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

setWindowSize

*Set and Get Window Size for xlsx file*

---

## Description

Set and Get Window Size for xlsx file

## Usage

```
setWindowSize(
  wb,
  xWindow = NULL,
  yWindow = NULL,
  windowWidth = NULL,
  windowHeight = NULL
)

getWindowSize(wb)
```

**Arguments**

wb	A Workbook object
xWindow	the horizontal coordinate of the top left corner of the window
yWindow	the vertical coordinate of the top left corner of the window
windowWidth	the width of the window
windowHeight	the height of the window

Set the size and position of the window when you open the xlsx file. The units are in twips. See [Microsoft's documentation for the xlsx standard](#)

**Examples**

```
## Create Workbook object and add worksheets
wb <- createWorkbook()
addWorksheet(wb, "S1")
getWindowSize(wb)
setWindowSize(wb, windowWidth = 10000)
```

---

sheets	<i>Returns names of worksheets.</i>
--------	-------------------------------------

---

**Description**

DEPRECATED. Use `names()`.

**Usage**

```
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**

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

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

## Return names of sheets, can not be used for assignment.
names(wb)
# openXL(wb)

names(wb) <- c("A", "B", "C")
names(wb)
# openXL(wb)
```

---

sheetVisibility	<i>Get/set worksheet visible state</i>
-----------------	--

---

**Description**

Get and set worksheet visible state

**Usage**

```
sheetVisibility(wb)

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**

```
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
```

---

sheetVisible	<i>Get worksheet visible state.</i>
--------------	-------------------------------------

---

### Description

DEPRECATED - Use function 'sheetVisibility()

### Usage

```
sheetVisible(wb)

sheetVisible(wb) <- value
```

### Arguments

wb	A workbook object
value	a logical vector the same length as sheetVisible(wb)

### 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	<i>Set worksheet gridlines to show or hide.</i>
---------------	---

---

**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 FALSE, grid lines are hidden.

**Author(s)**

Alexander Walker

**Examples**

```
wb <- loadWorkbook(file = system.file("extdata", "loadExample.xlsx", package = "openxlsx"))
names(wb) ## list worksheets in workbook
showGridLines(wb, 1, showGridLines = FALSE)
showGridLines(wb, "testing", showGridLines = FALSE)
## Not run:
saveWorkbook(wb, "showGridLinesExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

temp_xlsx	<i>helper function to create temporary directory for testing purpose</i>
-----------	--

---

**Description**

helper function to create temporary directory for testing purpose

**Usage**

```
temp_xlsx(name = "temp_xlsx")
```

**Arguments**

name	for the temp file
------	-------------------

---

ungroupColumns      *Ungroup Columns*

---

**Description**

Ungroup a selection of columns

**Usage**

```
ungroupColumns(wb, sheet, cols)
```

**Arguments**

wb	A workbook object
sheet	A name or index of a worksheet
cols	Indices of columns to ungroup

**Details**

If column was previously hidden, it will now be shown

**Author(s)**

Joshua Sturm

**See Also**

[ungroupRows\(\)](#) To ungroup rows

---

ungroupRows      *Ungroup Rows*

---

**Description**

Ungroup a selection of rows

**Usage**

```
ungroupRows(wb, sheet, rows)
```

**Arguments**

wb	A workbook object
sheet	A name or index of a worksheet
rows	Indices of rows to ungroup

**Details**

If row was previously hidden, it will now be shown

**Author(s)**

Joshua Sturm

**See Also**

[ungroupColumns\(\)](#)

---

worksheetOrder	<i>Order of worksheets in xlsx file</i>
----------------	---

---

**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

**Details**

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

**Examples**

```
## 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
## Not run:
saveWorkbook(wb, "worksheetOrderExample.xlsx", overwrite = TRUE)

## End(Not run)
worksheetOrder(wb) <- c(3, 2, 1)
## Not run:
saveWorkbook(wb, "worksheetOrderExample2.xlsx", overwrite = TRUE)

## End(Not run)

```

---

write.xlsx

*write data to an xlsx file*


---

### Description

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

### Usage

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

### Arguments

x	A data.frame or a (named) list of objects that can be handled by <a href="#">writeData()</a> or <a href="#">writeDataTable()</a> to write to file
file	A file path to save the xlsx file
asTable	If TRUE will use <a href="#">writeDataTable()</a> rather than <a href="#">writeData()</a> to write x to the file (default: FALSE)
overwrite	Overwrite existing file (Defaults to TRUE as with <code>write.table</code> )
...	Additional arguments passed to <a href="#">buildWorkbook()</a> ; see details

### Value

A workbook object

### Optional Parameters

#### 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 (or na.string, if not NULL) in Excel, else NA cells will be empty. Defaults to FALSE.

**na.string** If not NULL, and if keepNA is TRUE, NA values are converted to this string in Excel. Defaults to NULL.

#### **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** May be a single value for all columns (or "auto"), or a list of vectors that will be recycled for each sheet (see examples)

#### **Author(s)**

Alexander Walker, Jordan Mark Barbone

#### **See Also**

[addWorksheet\(\)](#)

[writeData\(\)](#)

[createStyle\(\)](#) for style parameters

[buildWorkbook\(\)](#)

**Examples**

```

## write to working directory
options("openxlsx.borderColour" = "#4F80BD") ## set default border colour
## Not run:
write.xlsx(iris, file = "writeXLSX1.xlsx", colNames = TRUE, borders = "columns")
write.xlsx(iris, file = "writeXLSX2.xlsx", colNames = TRUE, borders = "surrounding")

## End(Not run)

hs <- createStyle(
  textDecoration = "BOLD", fontColour = "#FFFFFF", fontSize = 12,
  fontName = "Arial Narrow", fgFill = "#4F80BD"
)
## Not run:
write.xlsx(iris,
  file = "writeXLSX3.xlsx",
  colNames = TRUE, borders = "rows", headerStyle = hs
)

## End(Not run)

## 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))
## Not run:
write.xlsx(l, "writeList1.xlsx", colWidths = c(NA, "auto", "auto"))

## End(Not run)

## different sheets can be given different parameters
## Not run:
write.xlsx(l, "writeList2.xlsx",
  startCol = c(1, 2, 3), startRow = 2,
  asTable = c(TRUE, TRUE, FALSE), withFilter = c(TRUE, FALSE, FALSE)
)

## End(Not run)

# specify column widths for multiple sheets
## Not run:
write.xlsx(l, "writeList2.xlsx", colWidths = 20)
write.xlsx(l, "writeList2.xlsx", colWidths = list(100, 200, 300))
write.xlsx(l, "writeList2.xlsx", colWidths = list(rep(10, 5), rep(8, 11), rep(5, 5)))

## End(Not run)

```



**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 <a href="#">createComment()</a> .
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)
## Not run:
saveWorkbook(wb, file = "writeCommentExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

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,
  array = FALSE,
  xy = NULL,
  colNames = TRUE,
  rowNames = FALSE,
  headerStyle = openxlsx_getOp("headerStyle"),
  borders = openxlsx_getOp("borders", "none"),
  borderColour = openxlsx_getOp("borderColour", "black"),
  borderStyle = openxlsx_getOp("borderStyle", "thin"),
  withFilter = openxlsx_getOp("withFilter", FALSE),
  keepNA = openxlsx_getOp("keepNA", FALSE),
  na.string = openxlsx_getOp("na.string"),
  name = NULL,
  sep = ", ",
  col.names,
  row.names
)
```

### 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.
array	A bool if the function written is of type array
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 <i>respective abbreviations</i> . 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 <a href="#">here</a> ).
borderStyle	Border line style <b>none</b> no border <b>thin</b> thin border <b>medium</b> medium border <b>dashed</b> dashed border <b>dotted</b> dotted border <b>thick</b> thick border <b>double</b> double line border <b>hair</b> hairline border <b>mediumDashed</b> medium weight dashed border <b>dashDot</b> dash-dot border <b>mediumDashDot</b> medium weight dash-dot border <b>dashDotDot</b> dash-dot-dot border <b>mediumDashDotDot</b> medium weight dash-dot-dot border <b>slantDashDot</b> slanted dash-dot border
withFilter	If TRUE or NA, 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 (or na.string, if not NULL) in Excel, else NA cells will be empty.
na.string	If not NULL, and if keepNA is TRUE, NA values are converted to this string in Excel.
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).
row.names, col.names	Deprecated, please use rowNames, colNames instead

### 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.

### Value

invisible(0)

**Author(s)**

Alexander Walker

**See Also**[writeDataTable\(\)](#)**Examples**

```
## 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 #4F81BD
options("openxlsx.borderColour" = "#4F81BD")

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

## 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 = "0", 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 = "0", startRow = 12, borders = "rows"
)
```

```
#####
## Header Styles

hs1 <- createStyle(
  fgFill = "#DCE6F1", 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 = "0", 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) <- paste0("Hyperlink", 1:4) # Optional: names will be used as display text
class(v) <- "hyperlink"
writeData(wb, "Cars", x = v, xy = c("B", 32))

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

df <- data.frame(
  x = 1:3, y = 1:3,
  z = paste0(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)
## Not run:
saveWorkbook(wb, "writeDataExample.xlsx", overwrite = TRUE)

## End(Not run)
```

---

writeDataTable	<i>Write to a worksheet as an Excel table</i>
----------------	---

---

### Description

Write to a worksheet and format as an Excel table

### Usage

```
writeDataTable(
  wb,
  sheet,
  x,
  startCol = 1,
  startRow = 1,
  xy = NULL,
  colNames = TRUE,
  rowNames = FALSE,
  tableStyle = openxlsx_getOp("tableStyle", "TableStyleLight9"),
  tableName = NULL,
  headerStyle = openxlsx_getOp("headerStyle"),
  withFilter = openxlsx_getOp("withFilter", TRUE),
  keepNA = openxlsx_getOp("keepNA", FALSE),
  na.string = openxlsx_getOp("na.string"),
  sep = ", ",
  stack = FALSE,
  firstColumn = openxlsx_getOp("firstColumn", FALSE),
  lastColumn = openxlsx_getOp("lastColumn", FALSE),
  bandedRows = openxlsx_getOp("bandedRows", TRUE),
  bandedCols = openxlsx_getOp("bandedCols", FALSE),
  col.names,
  row.names
)
```

### 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.
headerStyle	Custom style to apply to column names.
withFilter	If TRUE or NA, columns with have filters in the first row.
keepNA	If TRUE, NA values are converted to #N/A (or na.string, if not NULL) in Excel, else NA cells will be empty.
na.string	If not NULL, and if keepNA is TRUE, NA values are converted to this string in Excel.
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).
stack	If TRUE the new style is merged with any existing cell styles. If FALSE, any existing style is replaced by the new style.

**The below options correspond to Excel table options:**

Header Row     First Column     Filter Button  
 Total Row     Last Column  
 Banded Rows     Banded Columns

Table Style Options

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
row.names, col.names	Deprecated, please use rowNames, colNames instead

**Details**

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

**See Also**

```

addWorksheet()
writeData()
removeTable()
getTables()

```

**Examples**

```

## see package vignettes for further examples.

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

#####
## -- write data.frame as an Excel table with column filters
## -- default table style is "TableStyleMedium2"

writeDataTable(wb, "S1", x = iris)

writeDataTable(wb, "S2",
  x = mtcars, xy = c("B", 3), rowNames = TRUE,
  tableStyle = "TableStyleLight9"
)

df <- data.frame(
  "Date" = Sys.Date() - 0:19,
  "T" = TRUE, "F" = FALSE,
  "Time" = Sys.time() - 0:19 * 60 * 60,
  "Cash" = paste("$", 1:20), "Cash2" = 31:50,
  "hLink" = "https://CRAN.R-project.org/",
  "Percentage" = seq(0, 1, length.out = 20),
  "TinyNumbers" = runif(20) / 1E9, stringsAsFactors = FALSE
)

## openxlsx will apply default Excel styling for these classes
class(df$Cash) <- c(class(df$Cash), "currency")
class(df$Cash2) <- c(class(df$Cash2), "accounting")
class(df$hLink) <- "hyperlink"
class(df$Percentage) <- c(class(df$Percentage), "percentage")
class(df$TinyNumbers) <- c(class(df$TinyNumbers), "scientific")

writeDataTable(wb, "S3", x = df, startRow = 4, rowNames = TRUE, tableStyle = "TableStyleMedium9")

#####
## Additional Header Styling and remove column filters

```



```

writeDataTable(wb,
  sheet = 1, x = iris, startCol = 7, headerStyle = createStyle(textRotation = 45),
  withFilter = FALSE
)

#####
## Save workbook
## Open in excel without saving file: openXL(wb)
## Not run:
saveWorkbook(wb, "writeDataTableExample.xlsx", overwrite = TRUE)

## End(Not run)

#####
## Pre-defined table styles gallery

wb <- createWorkbook(paste0("tableStylesGallery.xlsx"))
addWorksheet(wb, "Style Samples")
for (i in 1:21) {
  style <- paste0("TableStyleLight", i)
  writeDataTable(wb,
    x = data.frame(style), sheet = 1,
    tableStyle = style, startRow = 1, startCol = i * 3 - 2
  )
}

for (i in 1:28) {
  style <- paste0("TableStyleMedium", i)
  writeDataTable(wb,
    x = data.frame(style), sheet = 1,
    tableStyle = style, startRow = 4, startCol = i * 3 - 2
  )
}

for (i in 1:11) {
  style <- paste0("TableStyleDark", i)
  writeDataTable(wb,
    x = data.frame(style), sheet = 1,
    tableStyle = style, startRow = 7, startCol = i * 3 - 2
  )
}

## openXL(wb)
## Not run:
saveWorkbook(wb, file = "tableStylesGallery.xlsx", overwrite = TRUE)

## End(Not run)

```

---

writeFormula	<i>Write a character vector as an Excel Formula</i>
--------------	---

---

**Description**

Write a a character vector containing Excel formula to a worksheet.

**Usage**

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

**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.
array	A bool if the function written is of type array
xy	An alternative to specifying startCol and startRow individually. A vector of the form c(startCol, startRow).

**Details**

Currently only the english version of functions are supported. Please don't use the local translation. The examples below show a small list of possible formulas:

- SUM(B2:B4)
- AVERAGE(B2:B4)
- MIN(B2:B4)
- MAX(B2:B4)
- ...

**Author(s)**

Alexander Walker

**See Also**

[writeData\(\)](#) [makeHyperlinkString\(\)](#)

**Examples**

```
## 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)

## 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)

## Save workbook
## Not run:
saveWorkbook(wb, "writeFormulaExample.xlsx", overwrite = TRUE)

## End(Not run)
```

```
## 4. - Writing internal hyperlinks

wb <- createWorkbook()
addWorksheet(wb, "Sheet1")
addWorksheet(wb, "Sheet2")
writeFormula(wb, "Sheet1", x = '=HYPERLINK("#Sheet2!B3", "Text to Display - Link to Sheet2")')

## Save workbook
## Not run:
saveWorkbook(wb, "writeFormulaHyperlinkExample.xlsx", overwrite = TRUE)

## End(Not run)
```

# Index

- \* **datasets**
  - openxlsx\_options, 59
  - openxlsxFontSizeLookupTable, 58
- activeSheet, 4
- activeSheet<- (activeSheet), 4
- addCreator, 5
- addFilter, 5
- addFilter(), 6
- addStyle, 6
- addStyle(), 30
- addWorksheet, 7
- addWorksheet(), 60, 79, 84, 95, 104
- all.equal, 10
- as.character.formula, 10
- auto\_heights, 11
  
- buildWorkbook, 12
- buildWorkbook(), 94, 95
  
- cloneWorksheet, 14
- col2int, 15
- conditionalFormat, 15
- conditionalFormatting, 17
- conditionalFormatting(), 15, 16
- convertFromExcelRef, 23
- convertToDate, 23
- convertToDate(), 39
- convertToDateTime, 24
- copyWorkbook, 25
- createComment, 25
- createComment(), 73, 97
- createNamedRegion, 26
- createNamedRegion(), 40
- createStyle, 28
- createStyle(), 7, 16–19, 26, 95
- createWorkbook, 31
- createWorkbook(), 79
  
- databar (conditionalFormatting), 17
  
- dataValidation, 32
- deleteData, 34
- deleteDataColumn, 35
- deleteNamedRegion (createNamedRegion), 26
  
- freezePane, 36
  
- get\_worksheet\_entries, 43
- getBaseFont, 37
- getCellRefs, 38
- getCreators, 38
- getDateOrigin, 39
- getNamedRegions, 40
- getNamedRegions(), 27, 69, 71
- getSheetNames, 41
- getStyles, 42
- getStyles(), 78
- getTables, 42
- getTables(), 75, 104
- getWindowSize (setWindowSize), 87
- groupColumns, 44
- groupColumns(), 45
- groupRows, 45
- groupRows(), 44
  
- if\_null\_then, 46
- insertImage, 47
- insertImage(), 49
- insertPlot, 48
- insertPlot(), 48
- int2col, 50
  
- loadWorkbook, 50
- loadWorkbook(), 32, 79
  
- makeHyperlinkString, 51
- makeHyperlinkString(), 107
- mergeCells, 53
- mergeCells(), 72
- modifyBaseFont, 54

names, 55  
 names(), 77, 88  
 names<- .Workbook (names), 55  
  
 op. openxlsx (openxlsx\_options), 59  
 op. openxlsx(), 58  
 openXL, 56  
 openxlsx, 57  
 openxlsx-package (openxlsx), 57  
 openxlsx\_getOp (openxlsx\_options), 59  
 openxlsx\_options, 59  
 openxlsx\_setOp (openxlsx\_options), 59  
 openxlsxFontSizeLookupTable, 58  
 openxlsxFontSizeLookupTableBold  
     (openxlsxFontSizeLookupTable),  
     58  
  
 pageBreak, 60  
 pageSetup, 60  
 protectWorkbook, 64  
 protectWorksheet, 65  
  
 read.xlsx, 67  
 read.xlsx(), 58, 71  
 readWorkbook, 69  
 removeCellMerge, 71  
 removeCellMerge(), 54  
 removeColWidths, 72  
 removeColWidths(), 81  
 removeComment, 73  
 removeFilter, 73  
 removeRowHeights, 74  
 removeRowHeights(), 86  
 removeTable, 75  
 removeTable(), 104  
 removeWorksheet, 76  
 removeWorksheet(), 51  
 renameWorksheet, 77  
 replaceStyle, 78  
 replaceStyle(), 42  
  
 saveWorkbook, 79  
 saveWorkbook(), 32  
 setColWidths, 80  
 setColWidths(), 12, 44, 72  
 setFooter, 81  
 setHeader, 82  
 setHeaderFooter, 83  
 setLastModifiedBy, 85  
  
 setRowHeights, 86  
 setRowHeights(), 74  
 setWindowSize, 87  
 sheets, 88  
 sheetVisibility, 89  
 sheetVisibility<- (sheetVisibility), 89  
 sheetVisible, 90  
 sheetVisible<- (sheetVisible), 90  
 showGridLines, 91  
  
 temp\_xlsx, 91  
  
 ungroupColumns, 92  
 ungroupColumns(), 44, 93  
 ungroupRows, 92  
 ungroupRows(), 45, 92  
  
 worksheetOrder, 93  
 worksheetOrder<- (worksheetOrder), 93  
 write.xlsx, 94  
 write.xlsx(), 14, 58  
 writeComment, 96  
 writeComment(), 26, 73  
 writeData, 98  
 writeData(), 6, 12, 24, 58, 79, 94, 95, 104,  
     107  
 writeDataTable, 102  
 writeDataTable(), 12, 58, 79, 94, 100  
 writeFormula, 106  
 writeFormula(), 51