Package ‘openxlsx’

Type Package

Title Read, Write and Edit xlsx Files

Version 4.1.4

Date 2019-12-04

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, utils, zip, stringi

Suggests knitr, testthat, roxygen2

LinkingTo Rcpp

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.0.2

Collate 'CommentClass.R' 'HyperlinkClass.R' 'RcppExports.R'
    'class_definitions.R' 'StyleClass.R' 'WorkbookClass.R'
    'baseXML.R' 'borderFunctions.R' 'chartsheet_class.R'
    'conditional_formatting.R' 'helperFunctions.R' 'loadWorkbook.R'
    'onUnload.R' 'openXL.R' 'openxlsx.R' 'openxlsxCoerce.R'
    'readWorkbook.R' 'sheet_data_class.R'
    'workbook_column_widths.R' 'workbook_read_workbook.R'
    'workbook_write_data.R' 'worksheet_class.R' 'wrappers.R'
    'writeData.R' 'writeDataTable.R' 'writexlsx.R'

NeedsCompilation yes
R topics documented:

**Author** Philipp Schauberger [aut, cre],
Alexander Walker [aut],
Luca Braglia [ctb]

**Maintainer** Philipp Schauberger <philipp@schauberger.co.at>

**Repository** CRAN

**Date/Publication** 2019-12-06 12:50:02 UTC

R topics documented:

- addFilter .................................................. 3
- addStyle ..................................................... 4
- addWorksheet ............................................... 5
- all.equal ..................................................... 8
- cloneWorksheet ............................................. 8
- conditionalFormat ......................................... 9
- conditionalFormatting ..................................... 10
- convertFromExcelRef ...................................... 14
- convertToDate ............................................. 14
- convertToDateRef .......................................... 15
- copyWorkbook ............................................. 16
- createComment ............................................ 16
- createNamedRegion ......................................... 18
- createStyle ............................................... 19
- createWorkbook ........................................... 22
- dataValidation ............................................ 23
- deleteData ................................................ 25
- freezePane ............................................... 26
- getBaseFont ............................................... 27
- getCellRefs ............................................... 28
- getDateOrigin ............................................. 28
- getNamedRegions .......................................... 29
- getSheetNames ............................................ 30
- getStyles ................................................... 31
- getTables .................................................. 31
- insertImage ............................................... 32
- insertPlot ................................................. 33
- int2col ..................................................... 35
- loadWorkbook ............................................. 35
- makeHyperlinkString ...................................... 36
- mergeCells ................................................ 37
- modifyBaseFont .......................................... 38
- names ...................................................... 39
- openXL ...................................................... 39
- openxlsx .................................................. 40
- pageBreak ................................................ 41
- pageSetup ................................................ 42
- protectWorkbook ........................................... 45
**addFilter**

**Add column filters**

**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.
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.

Examples

```r
wb <- createWorkbook()
addWorksheet(wb, "Sheet 1")
addWorksheet(wb, "Sheet 2")
addWorksheet(wb, "Sheet 3")

writeData(wb, 1, iris)
addFilter(wb, 1, row = 1, cols = 1:ncol(iris))

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

## Similarly
writeDataTable(wb, 3, iris)

## Not run: saveWorkbook(wb, file = "addFilterExample.xlsx", overwrite = TRUE)
```
Add a worksheet to a workbook

Add a worksheet to a Workbook object

Usage

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

- \&[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)

---

**all.equal**

*Check equality of workbooks*

**Description**

Check equality of workbooks

**Usage**

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

**Arguments**

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

---

**cloneWorksheet**

*Clone a worksheet to a workbook*

**Description**

Clone a worksheet to a Workbook object

**Usage**

```r
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
### Examples

```r
## 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)
```

### Description

DEPRECATED! USE `conditionalFormatting`

### Usage

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

### Arguments

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

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wb</td>
<td>A workbook object</td>
</tr>
<tr>
<td>sheet</td>
<td>A name or index of a worksheet</td>
</tr>
<tr>
<td>cols</td>
<td>Columns to apply conditional formatting to</td>
</tr>
<tr>
<td>rows</td>
<td>Rows to apply conditional formatting to</td>
</tr>
<tr>
<td>rule</td>
<td>The condition under which to apply the formatting. See examples.</td>
</tr>
<tr>
<td>style</td>
<td>A style to apply to those cells that satisfy the rule. Default is createStyle(fontColour = &quot;#9C0006&quot;, bgFill = &quot;:FFC7CE&quot;)</td>
</tr>
<tr>
<td>type</td>
<td>Either 'expression', 'colorscale', 'databar', 'duplicates' or 'contains' (case insensitive).</td>
</tr>
<tr>
<td>...</td>
<td>See below</td>
</tr>
</tbody>
</table>
Details

See Examples.

If type == "expression"

- style is a Style object. See createStyle
- rule is an expression. Valid operators are "<", "<="", ">", ">="", ">="", "!="."

If type == "colourScale"

- style is a vector of colours with length 2 or 3
- rule can be NULL or a vector of colours of equal length to styles

If type == "databar"

- style is a vector of colours with length 2 or 3
- rule is a numeric vector specifying the range of the databar colours. Must be equal length to style
- ...
  - showvalue If FALSE the cell value is hidden. Default TRUE.
  - gradient If FALSE colour gradient is removed. Default TRUE.
  - border If FALSE the border around the database is hidden. Default TRUE.

If type == "duplicates"

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

If type == "contains"

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

If type == "between"

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

Author(s)

Alexander Walker

See Also

createStyle
Examples

```r
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, "colourScale", zoom = 30)
addWorksheet(wb, "databar")
addWorksheet(wb, "between")
addWorksheet(wb, "logical operators")

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

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

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

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

## highlight entire range cols X rows dependent only on cell A1
writeData(wb, "Dependent on", -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 = "-")
```
### Databar Example

```r
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)
```

---

### Databar Example

```r
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)

---

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", ...)

**convertToDateTime**

**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**

```r
## 2014 April 21st to 25th
convertToDate(c(41750, 41751, 41752, 41753, 41754, NA) )
convertToDate(c(41750.2, 41751.99, NA, 41753 ))
```

**Description**

Convert from excel time number to R POSIXct type.

**Usage**

```r
convert.ToDateTime(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**

```r
x <- c(41821.8127314815, 41820.8127314815, NA, 41819, NaN)
convert.ToDateTime(x)
convert.ToDateTime(x, tx = "Australia/Perth")
```
copyWorkbook  

Create a Workbook object.

**Description**

Just a wrapper of wb$copy()

**Usage**

```r
copyWorkbook(wb)
```

**Arguments**

- `wb`  
  A workbook object

**Value**

Workbook

**Examples**

```r
wb <- createWorkbook()
w2 <- wb ## does not create a copy
wb3 <- copyWorkbook(wb) ## wrapper for wb$copy()

addWorksheet(wb, "Sheet1") ## adds worksheet to both wb and w2 but not wb3

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

createComment  

Create a Comment object to pass to writeComment()

**Description**

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

Usage

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

Arguments

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

See Also

writeComment

Examples

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)
createNamedRegion Create a named region.

Description
Create a named region

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

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

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

Author(s)
Alexander Walker

See Also
getNamedRegions

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
createStyle

Description

Create a new style to apply to worksheet cells

Usage

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

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

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

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

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

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

- none No Border
- thin thin border
- medium medium border
- dashed dashed border
**createStyle**

- `dotted` dotted border
- `thick` thick border
- `double` double line border
- `hair` Hairline border
- `mediumDashed` medium weight dashed border
- `dashDot` dash-dot border
- `mediumDashDot` medium weight dash-dot border
- `dashDotDot` dash-dot-dot border
- `mediumDashDotDot` medium weight dash-dot-dot border
- `slantDashDot` slanted dash-dot border

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

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

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

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

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

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

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

**indent**  Horizontal indentation of cell contents.

**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
createWorkbook

Create a new Workbook object

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
)

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"))
Arguments

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

title: Workbook properties title

subject: Workbook properties subject

category: Workbook properties category

Value

Workbook object

Author(s)

Alexander Walker

See Also

loadWorkbook
saveWorkbook

Examples

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

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

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

Description

Add Excel data validation to cells
dataValidation

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

Arguments

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

Examples

  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)
deleteData

```r
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)
```

```
### 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**  
```r
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

```r
## 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)
```

---

**freezePane**

*Freeze a worksheet pane*

### Description

Freeze a worksheet pane

### Usage

```r
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
### # Create a new workbook

```r
wb <- createWorkbook("Kenshin")
```

### # Add some worksheets

```r
addWorksheet(wb, "Sheet 1")
addWorksheet(wb, "Sheet 2")
addWorksheet(wb, "Sheet 3")
addWorksheet(wb, "Sheet 4")
```

### # Freeze Panes

```r
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

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

---

### getBaseFont

Return the workbook default font

**Description**

Return the workbook default font

Returns the base font used in the workbook.

**Usage**

```r
getBaseFont(wb)
```

**Arguments**

- **wb** A workbook object

**Author(s)**

Alexander Walker

**Examples**

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

```r
## 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**
```
gegetCellRefs(cellCoords)
```

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

**Value**
Excel alphanumerical cell reference

**Author(s)**
Philipp Schaubberger, Alexander Walker

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

---

**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.
getNamedRegions

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 it 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

```r
# 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  Get named regions

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

Usage
getNamedRegions(x)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>An xlsx file or Workbook object</td>
</tr>
</tbody>
</table>

See Also
createNamedRegion
getSheetNames

Get names of worksheets

Description

Returns the worksheet names within an xlsx file

Usage

getSheetNames(file)

Arguments

file An xlsx or xslm file.

Value

Character vector of worksheet names.
getStyles

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)
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

wb    A workbook object
sheet A name or index of a worksheet
file  An image file. Valid file types are: jpeg, png, bmp
width Width of figure.
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,
  ...)
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

Convert integer to Excel column

Description

Converts an integer to an Excel column label.

Usage

int2col(x)

Arguments

x

A numeric vector

Examples

int2col(1:10)

loadWorkbook

Load an existing .xlsx file

Description

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

Usage

loadWorkbook(file, xlsxFile = NULL, isUnzipped = FALSE)

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
makeHyperlinkString

create Excel hyperlink string

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

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
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
```r
## 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)
```
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)
### names

**Description**

get or set worksheet names

**Usage**

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

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

**Arguments**

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

**Examples**

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

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).
openxlsx

Usage

openXL(file=NULL)

Arguments

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

Author(s)

Luca Braglia

Examples

# file example
eexample(writeData)
#openXL("writeDataExample.xlsx")

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

openxlsx xlsx reading, writing and editing.

Description

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

Details

The openxlsx package uses global options to simplify formatting:

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

See the Formatting vignette for examples.

Additional options
options("openxlsx.compressionLevel" = "9") ## set zip compression level, default is "1".

See Also

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

---

pageBreak

*add a page break to a worksheet*

---

Description

insert page breaks into a worksheet

Usage

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

Arguments

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

See Also

- addWorksheet

Examples

```r
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)
## 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,               # A workbook object
  sheet,           # A name or index of a worksheet
  orientation = NULL,  # Page orientation. One of "portrait" or "landscape"
  scale = 100,      # Print scaling. Numeric value between 10 and 400
  left = 0.7,       # left page margin in inches
  right = 0.7,      # right page margin in inches
  top = 0.75,       # top page margin in inches
  bottom = 0.75,    # bottom page margin in inches
  header = 0.3,     # header margin in inches
  footer = 0.3,     # footer margin in inches
  fitToWidth = FALSE,  # If TRUE, worksheet is scaled to fit to page width on printing.
  fitToHeight = FALSE, # If TRUE, worksheet is scaled to fit to page height on printing.
  paperSize = NULL,  # See details. Default value is 9 (A4 paper).
  printTitleRows = NULL,  # Rows to repeat at top of page when printing. Integer vector.
  printTitleCols = NULL  # Columns to repeat at left when printing. Integer vector.
)
```

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.
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
Examples

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

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

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

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

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

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

protectWorkbook

---

**Description**

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

**Usage**

```r
protectWorkbook(
  wb, 
  protect = TRUE, 
  password = NULL, 
  lockStructure = FALSE, 
  lockWindows = FALSE 
)
```

**Arguments**

- **wb** : A workbook object
protectWorksheet

protectWorksheet

Protect a worksheet from modifications

Description

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

Usage

```r
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

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

```r
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**

```r
read.xlsx(
  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**

- `xlsxFile`  
  An xlsx file, Workbook object or URL to xlsx file.
- `sheet`  
  The name or index of the sheet to read data from.
**startRow**
- first row to begin looking for data. Empty rows at the top of a file are always skipped, regardless of the value of startRow.

**colNames**
- If TRUE, the first row of data will be used as column names.

**rowNames**
- If TRUE, first column of data will be used as row names.

**detectDates**
- If TRUE, attempt to recognise dates and perform conversion.

**skipEmptyRows**
- If TRUE, empty rows are skipped else empty rows after the first row containing data will return a row of NAs.

**skipEmptyCols**
- If TRUE, empty columns are skipped.

**rows**
- A numeric vector specifying which rows in the Excel file to read. If NULL, all rows are read.

**cols**
- A numeric vector specifying which columns in the Excel file to read. If NULL, all columns are read.

**check.names**
- logical. If TRUE then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names.

**sep.names**
- One character which substitutes blanks in column names. By default, ".".

**namedRegion**
- A named region in the Workbook. If not NULL startRow, rows and cols parameters are ignored.

**na.strings**
- A character vector of strings which are to be interpreted as NA. Blank cells will be returned as NA.

**fillMergedCells**
- If TRUE, the value in a merged cell is given to all cells within the merge.

**Details**

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

**Value**

data.frame

**Author(s)**

Alexander Walker

**See Also**

getNamedRegions
readWorkbook

\textit{Read from an Excel file or Workbook object}

\section*{Description}

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

\section*{Usage}

\begin{verbatim}
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
)
\end{verbatim}

\section*{Arguments}

\begin{itemize}
  \item \texttt{xlsxFile} \hspace{1cm} An xlsx file, Workbook object or URL to xlsx file.
  \item \texttt{sheet} \hspace{1cm} The name or index of the sheet to read data from.
  \item \texttt{startRow} \hspace{1cm} first row to begin looking for data. Empty rows at the top of a file are always skipped, regardless of the value of \texttt{startRow}.
  \item \texttt{colNames} \hspace{1cm} If TRUE, the first row of data will be used as column names.
  \item \texttt{rowNames} \hspace{1cm} If TRUE, first column of data will be used as row names.
  \item \texttt{detectDates} \hspace{1cm} If TRUE, attempt to recognise dates and perform conversion.
  \item \texttt{skipEmptyRows} \hspace{1cm} If TRUE, empty rows are skipped else empty rows after the first row containing data will return a row of NAs.
  \item \texttt{skipEmptyCols} \hspace{1cm} If TRUE, empty columns are skipped.
  \item \texttt{rows} \hspace{1cm} A numeric vector specifying which rows in the Excel file to read. If NULL, all rows are read.
  \item \texttt{cols} \hspace{1cm} A numeric vector specifying which columns in the Excel file to read. If NULL, all columns are read.
\end{itemize}
check.names logical. If TRUE then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names

sep.names One character which substitutes blanks in column names. By default, "."

namedRegion A named region in the Workbook. If not NULL startRow, rows and cols parameters are ignored.

na.strings A character vector of strings which are to be interpreted as NA. Blank cells will be returned as NA.

fillMergedCells If TRUE, the value in a merged cell is given to all cells within the merge.

Details

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)
```
removeColWidths

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 | Remove column widths from a worksheet

Description

Remove column widths from a worksheet

Usage

removeColWidths(wb, sheet, cols)

Arguments

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

Author(s)

Alexander Walker

See Also

setColWidths

Examples

```r
## 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)
```
removeComment

**Description**

Remove a cell comment from a worksheet

**Usage**

```r
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

**Description**

Removes filters from addFilter() and writeData()

**Usage**

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

**Arguments**

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

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)

removeRowHeights  Remove custom row heights from a worksheet

Description
Remove row heights from a worksheet

Usage
removeRowHeights(wb, sheet, rows)

Arguments

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

Author(s)
Alexander Walker

See Also

setRowHeights
Examples

## Create a new workbook

```r
wb <- loadWorkbook(file = system.file("extdata","loadExample.xlsx", package = "openxlsx"))
```

## remove any custom row heights in rows 1 to 10

```r
removeRowHeights(wb, 1, rows = 1:10)
```

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

---

**removeTable**  
*Remove an Excel table in a workbook*

### Description

List Excel tables in a workbook

### Usage

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

### Arguments

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

### Value

character vector of table names on the specified sheet

### Examples

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

```r
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)
```

```r
## 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)
```
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)
renameWorksheet Rename a worksheet

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)
replaceStyle  Replace an existing cell style

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)
saveWorkbook

Description

save a Workbook object to file

Usage

saveWorkbook(wb, file, overwrite = FALSE)

Arguments

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

Author(s)

Alexander Walker

See Also

createWorkbook
addWorksheet
loadWorkbook
writeData
writeDataTable

Examples

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

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

Set worksheet column widths

Description

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

Usage

```
setColWidths(
  wb,      # A workbook object
  sheet,   # A name or index of a worksheet
  cols,    # Indices of cols to set width
  widths = 8.43,  # 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 = rep(FALSE, length(cols)),  # Logical vector. If TRUE the column is hidden.
  ignoreMergedCells = FALSE  # Ignore any cells that have been merged with other cells in the calculation of "auto" column widths.
)
```

Arguments

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

Details

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

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

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

Author(s)

Alexander Walker

See Also

`removeColWidths`
Examples

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

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

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

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

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

---

**setFooter**

Set footer for all worksheets

---

**Description**

DEPRECATED

**Usage**

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

**Arguments**

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

**Author(s)**

Alexander Walker

**Examples**

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

```r
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

```r
setFooter(wb, "Center Footer Here", position="center")
setFooter(wb, "Bottom left", position="left")
setFooter(wb, Sys.Date(), position="right")
```

\dontrun{saveWorkbook(wb, "headerFooterExample.xlsx", overwrite = TRUE)}

## End(Not run)

---

**setHeader**  
_Set header for all worksheets_

### Description

DEPRECATED

### Usage

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

### Arguments

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

### Author(s)

Alexander Walker

### Examples

```r
## Not run:
w <- 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")
```
**setHeaderFooter**

Set headers and footers.

### Usage

```r
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
**setHeaderFooter**

- **&[Date]** Current date
- **&[Time]** Current time
- **&[Path]** File path
- **&[File]** File name
- **&[Tab]** Worksheet name

**Author(s)**
Alexander Walker

**See Also**

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

**Examples**

```r
wb <- createWorkbook()

addWorksheet(wb, "S1")
addWorksheet(wb, "S2")
addWorksheet(wb, "S3")
addWorksheet(wb, "S4")

writeData(wb, 1, 1:400)
writeData(wb, 2, 1:400)
writeData(wb, 3, 3:400)
writeData(wb, 4, 3:400)

setHeaderFooter(wb, sheet = "S1",
               header = c("ODD HEAD LEFT", "ODD HEAD CENTER", "ODD HEAD RIGHT"),
               footer = c("ODD FOOT RIGHT", "ODD FOOT CENTER", "ODD FOOT RIGHT"),
               evenHeader = c("EVEN HEAD LEFT", "EVEN HEAD CENTER", "EVEN HEAD RIGHT"),
               evenFooter = c("EVEN FOOT RIGHT", "EVEN FOOT CENTER", "EVEN FOOT RIGHT"),
               firstHeader = c("TOP", "OF FIRST", "PAGE"),
               firstFooter = c("BOTTOM", "OF FIRST", "PAGE"))

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

setHeaderFooter(wb, sheet = 3,
               header = c("ALL HEAD LEFT 2", "ALL HEAD CENTER 2", "ALL HEAD RIGHT 2")
               footer = c("ALL FOOT RIGHT 2", "ALL FOOT CENTER 2", "ALL FOOT RIGHT 2")

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

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

---

**setRowHeights**

*Set worksheet row heights*

**Description**

Set worksheet row heights

**Usage**

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

**Arguments**

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

**Author(s)**

Alexander Walker

**See Also**

`removeRowHeights`

**Examples**

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

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

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

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

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

Returns names of worksheets.

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  Get/set worksheet visible state

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  Get worksheet visible state.

Description
DEPRECATED - Use function `sheetVisibility()`
showGridLines

Set worksheet gridlines to show or hide.

Description

Set worksheet gridlines to show or hide.

Usage

showGridLines(wb, sheet, showGridLines = FALSE)

Arguments

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

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)

worksheetOrder \hspace{1cm} Order of worksheets in xlsx file

Description

Get/set order of worksheets in a Workbook object

Usage

worksheetOrder(wb)

worksheetOrder(wb) <- value

Arguments

wb \hspace{1cm} A workbook object

value \hspace{1cm} 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)
worksheetOrder(wb) <- c(3,2,1)
## Not run: saveWorkbook(wb, "worksheetOrderExample2.xlsx", overwrite = TRUE)

write.xlsx

write data to an xlsx file

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

Usage
write.xlsx(x, file, asTable = FALSE, ...)

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

Details
Optional parameters are:
createWorkbook Parameters
  • creator A string specifying the workbook author
addWorksheet Parameters
  • sheetName Name of the worksheet
  • gridLines A logical. If FALSE, the worksheet grid lines will be hidden.
- **tabColour** Colour of the worksheet tab. A valid colour (belonging to colours()) or a valid hex colour beginning with "#".
- **zoom** A numeric between 10 and 400. Worksheet zoom level as a percentage.

**writeData/writeDataTable Parameters**

- **startCol** A vector specifying the starting column(s) to write df
- **startRow** A vector specifying the starting row(s) to write df
- **xy** An alternative to specifying startCol and startRow individually. A vector of the form c(startCol, startRow)
- **colNames or col.names** If TRUE, column names of x are written.
- **rowNames or row.names** If TRUE, row names of x are written.
- **headerStyle** Custom style to apply to column names.
- **borders** Either "surrounding", "columns" or "rows" or NULL. If "surrounding", a border is drawn around the data. If "rows", a surrounding border is drawn a border around each row. If "columns", a surrounding border is drawn with a border between each column. If "all" all cell borders are drawn.
- **borderColour** Colour of cell border
- **borderStyle** Border line style.
- **keepNA** If TRUE, NA values are converted to #N/A (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** Must be value "auto". Sets all columns containing data to auto width.

**saveWorkbook Parameters**

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

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

**Value**

A workbook object

**Author(s)**

Alexander Walker
### writeComment

Write a Comment object to a worksheet

#### Usage

```r
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**: Row a row number of letter
- **comment**: a string containing the comment text
- **xy**: A vector of coordinates

#### Description

Write a Comment object to a worksheet

You can write a cell comment.

**See Also**

- `addWorksheet`
- `writeData`
- `createStyle` for style parameters

**Examples**

```r
## 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"))

## 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)
```
writeData

row
A row number.

comment
A Comment object. See createComment.

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

See Also
createComment

Examples

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

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

s1 <- createStyle(fontSize = 12, fontColour = "red", textDecoration = c("BOLD"))
s2 <- createStyle(fontSize = 9, fontColour = "black")

c2 <- createComment(comment = c("This Part Bold red\n\n", "This part black"), style = c(s1, s2))
c2

writeComment(wb, 1, col = 6, row = 3, comment = c2)

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

writeData

Write an object to a worksheet

Description

Write an object to worksheet with optional styling.

Usage

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

Arguments

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

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

keepNA     If TRUE, NA values are converted to #N/A (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).

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

#########################################################################
## Header Styles
hs1 <- createStyle(fgFill = "#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="O", startRow = 23, borders="columns", headerStyle = hs2)

#########################################################################
## Hyperlinks
## - vectors/columns with class 'hyperlink' are written as hyperlinks'
v <- rep("https://CRAN.R-project.org/", 4)
names(v) <- 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'

writeDataTable

Write to a worksheet as an Excel table

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 = "TableStyleLight9",
  tableName = NULL,
  headerStyle = NULL,
  withFilter = TRUE,
  keepNA = FALSE,
  na.string = NULL,
  sep = ",",
  stack = FALSE,
  firstColumn = FALSE,
  lastColumn = FALSE,
  bandedRows = TRUE,
  bandedCols = FALSE
)
Arguments

- **wb**: A Workbook object containing a worksheet.
- **sheet**: The worksheet to write to. Can be the worksheet index or name.
- **x**: A dataframe.
- **startCol**: A vector specifying the starting column to write df
- **startRow**: A vector specifying the starting row to write df
- **xy**: An alternative to specifying startCol and startRow individually. A vector of the form c(startCol, startRow)
- **colNames**: If TRUE, column names of x are written.
- **rowNames**: If TRUE, row names of x are written.
- **tableStyle**: Any excel table style name or ”none” (see ”formatting” vignette).
- **tableName**: name of table in workbook. The table name must be unique.
- **headerStyle**: Custom style to apply to column names.
- **withFilter**: If TRUE, columns with have filters in the first row.
- **keepNA**: If TRUE, NA values are converted to #N/A (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:

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

Details

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

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)
### Pre-defined table styles gallery

```r
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)
}
```

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

---

**writeFormula**

*Write a character vector as an Excel Formula*

**Description**

Write a character vector containing Excel formula to a worksheet

**Usage**

```r
writeFormula(wb, sheet, x, startCol = 1, startRow = 1, 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.
xy An alternative to specifying startCol and startRow individually. A vector of the form c(startCol, startRow).

Author(s)
Alexander Walker

See Also
writeData

Examples

## There are 3 ways to write a formula

```r
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)")
```
writeFormula

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)

## Writing internal hyperlinks
wb <- createWorkbook()
addWorksheet(wb, "Sheet1")
addWorksheet(wb, "Sheet2")
writeFormula(wb, "Sheet1", x = '=HYPERLINK("#Sheet2!B3", "Text to Display - Link to Sheet2")')
## Not run: saveWorkbook(wb, "writeFormulaHyperlinkExample.xlsx", overwrite = TRUE)
Index

addFilter, 3, 4
addStyle, 4, 22
addWorksheet, 5, 41, 59, 64, 72, 79
all.equal, 8
cloneworksheet, 8
conditionalFormat, 9
conditionalFormatting, 9, 10, 10
convertFromExcelRef, 14
convertToDate, 14, 29
conVertToDateime, 15
copyWorkbook, 16
createComment, 16, 53, 73
createnamedRegion, 18, 29
createstyle, 5, 10, 11, 17, 19, 72
createWorkbook, 22, 59
databar (conditionalFormatting), 10
dataValidation, 23
deleteData, 25
freezepane, 26
getBaseFont, 27
cellRefs, 28
dateOrigin, 28
getNamedRegions, 18, 29, 49, 51
getsheetNames, 30
getstyles, 31, 58
geta1s, 31, 55, 79
insertImage, 32, 34
insertPlot, 33, 33
int2col, 35
loadWorkbook, 23, 35, 59
makeHyperlinkString, 36
mergeCells, 37, 52
modifyBaseFont, 38
names, 39, 57, 66
names<-.Workbook (names), 39
openxl, 39
openxlsx, 40
pageBreak, 41
pageSetup, 42
protectWorkbook, 45
protectWorksheet, 46
read.xlsx, 41, 48, 51
readWorkbook, 50
removeCellMerge, 37, 51
removeColWidths, 52, 60
removeComment, 53
removeFilter, 53
removeRowHeights, 54, 65
removeTable, 55, 79
renameWorksheet, 36, 56
renameWorkbook, 57
replaceStyle, 31, 58
saveWorkbook, 23, 35, 59
setColWidths, 23, 59
setFooter, 61
setHeader, 62
setHeaderFooter, 63
setRowHeights, 54, 65
sheets, 66
sheetVisibility, 67
sheetVisibility<- (sheetVisibility), 67
sheetVisible, 67
sheetVisible<- (sheetVisible), 67
showGridLines, 68
worksheetOrder, 69
worksheetOrder<- (worksheetOrder), 69
write.xlsx, 41, 70
writeComment, 17, 53, 72
writeData, 4, 15, 41, 59, 70, 72, 73, 79, 81
writeDataTable, 41, 59, 75, 77
writeFormula, 36, 80