Package ‘openxlsx’

October 24, 2022

Type Package

Title Read, Write and Edit xlsx Files

Version 4.2.5.1

Date 2022-10-24

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 knitr, rmarkdown, roxygen2, testthat

LinkingTo Rcpp

VignetteBuilder knitr

Encoding UTF-8

Language en-US

RoxygenNote 7.1.2

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'
    'sheet_data_class.R' 'utils.R' 'workbook_column_widths.R'
    'workbook_read_workbook.R' 'workbook_write_data.R'
R topics documented:

'worksheet_class.R' 'wrappers.R' 'writeData.R'
'writeDataTable.R' 'writexlsx.R' 'zzz.R'

NeedsCompilation yes

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

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

Repository CRAN

Date/Publication 2022-10-24 12:57:15 UTC

R topics documented:

activeSheet .................................................. 4
addCreator .................................................. 5
addFilter .................................................. 5
addStyle .................................................. 6
addWorksheet ............................................... 7
all.equal ................................................... 10
buildWorkbook .............................................. 10
cloneWorksheet ............................................ 12
col2int .................................................. 13
conditionalFormat .......................................... 14
conditionalFormatting ...................................... 15
convertFromExcelRef ........................................ 20
convertToDate ............................................... 21
convertToDateTime .......................................... 22
copyWorkbook ............................................... 22
createComment ............................................... 23
createNamedRegion .......................................... 24
createStyle ............................................... 26
createWorkbook ............................................. 29
dataValidation .............................................. 30
deleteData .................................................. 32
freezePane .................................................. 33
getBaseFont ................................................ 34
getCellRefs ................................................ 35
getCreators ................................................ 35
getDateOrigin .............................................. 36
getNamedRegions ............................................ 37
getSheetNames .............................................. 38
getStyles .................................................. 39
getTables .................................................. 39
groupColumns .............................................. 40
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupRows</td>
<td>41</td>
</tr>
<tr>
<td>if_null_then</td>
<td>41</td>
</tr>
<tr>
<td>insertImage</td>
<td>42</td>
</tr>
<tr>
<td>insertPlot</td>
<td>43</td>
</tr>
<tr>
<td>int2col</td>
<td>45</td>
</tr>
<tr>
<td>loadWorkbook</td>
<td>45</td>
</tr>
<tr>
<td>makeHyperlinkString</td>
<td>46</td>
</tr>
<tr>
<td>mergeCells</td>
<td>48</td>
</tr>
<tr>
<td>modifyBaseFont</td>
<td>49</td>
</tr>
<tr>
<td>names</td>
<td>50</td>
</tr>
<tr>
<td>openXL</td>
<td>51</td>
</tr>
<tr>
<td>openxlsx</td>
<td>52</td>
</tr>
<tr>
<td>openxlsxFontSizeLookupTable</td>
<td>53</td>
</tr>
<tr>
<td>openxlsx_options</td>
<td>53</td>
</tr>
<tr>
<td>pageBreak</td>
<td>54</td>
</tr>
<tr>
<td>pageSetup</td>
<td>55</td>
</tr>
<tr>
<td>protectWorkbook</td>
<td>59</td>
</tr>
<tr>
<td>protectWorksheet</td>
<td>60</td>
</tr>
<tr>
<td>read.xlsx</td>
<td>62</td>
</tr>
<tr>
<td>readWorkbook</td>
<td>64</td>
</tr>
<tr>
<td>removeCellMerge</td>
<td>66</td>
</tr>
<tr>
<td>removeColWidths</td>
<td>66</td>
</tr>
<tr>
<td>removeComment</td>
<td>67</td>
</tr>
<tr>
<td>removeFilter</td>
<td>68</td>
</tr>
<tr>
<td>removeRowHeights</td>
<td>68</td>
</tr>
<tr>
<td>removeTable</td>
<td>69</td>
</tr>
<tr>
<td>removeWorksheets</td>
<td>70</td>
</tr>
<tr>
<td>renameWorksheet</td>
<td>71</td>
</tr>
<tr>
<td>replaceStyle</td>
<td>72</td>
</tr>
<tr>
<td>saveWorkbook</td>
<td>73</td>
</tr>
<tr>
<td>setColWidths</td>
<td>74</td>
</tr>
<tr>
<td>setFooter</td>
<td>76</td>
</tr>
<tr>
<td>setHeader</td>
<td>77</td>
</tr>
<tr>
<td>setHeaderFooter</td>
<td>78</td>
</tr>
<tr>
<td>setLastModifiedBy</td>
<td>80</td>
</tr>
<tr>
<td>setRowHeights</td>
<td>80</td>
</tr>
<tr>
<td>sheets</td>
<td>81</td>
</tr>
<tr>
<td>sheetVisibility</td>
<td>82</td>
</tr>
<tr>
<td>sheetVisible</td>
<td>83</td>
</tr>
<tr>
<td>showGridLines</td>
<td>84</td>
</tr>
<tr>
<td>temp_xlsx</td>
<td>85</td>
</tr>
<tr>
<td>ungroupColumns</td>
<td>85</td>
</tr>
<tr>
<td>ungroupRows</td>
<td>86</td>
</tr>
<tr>
<td>worksheetOrder</td>
<td>86</td>
</tr>
<tr>
<td>write.xlsx</td>
<td>87</td>
</tr>
<tr>
<td>writeComment</td>
<td>90</td>
</tr>
<tr>
<td>writeData</td>
<td>91</td>
</tr>
<tr>
<td>writeDataTable</td>
<td>95</td>
</tr>
</tbody>
</table>
activeSheet  

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 Schaubberger

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

Add another author to the meta data of the file.

Description

Just a wrapper of wb$addCreator()

Usage

addCreator(wb, Creator)

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>Creator</td>
<td>A string object with the name of the creator</td>
</tr>
</tbody>
</table>

Author(s)

Philipp Schauberger

Examples

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

addFilter

Add column filters

Description

Add excel column filters to a worksheet

Usage

addFilter(wb, sheet, rows, cols)

Arguments

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

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

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.
addWorksheet

Add a worksheet to a workbook

Description

Add a worksheet to a Workbook object

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)
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

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

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

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

addWorksheet(wb, "Sheet 6",
header = c("&[Date]\", "ALL HEAD CENTER 2", "&[Page] / &[Pages]\"),
footer = c("&[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**  
*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

---

**buildWorkbook**  
*Build Workbook*

**Description**  
Build a workbook from a data.frame or named list

**Usage**  
```r
buildWorkbook(x, asTable = FALSE, ...)
```
Arguments
x A data.frame or a (named) list of objects that can be handled by writeData() or writeDataTable() to write to file

asTable If TRUE will use writeDataTable() rather than writeData() to write x to the file (default: FALSE)

... Additional arguments passed to writeData(), writeDataTable(), setColWidths() (see Optional Parameters)

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.
cloneWorksheet

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

```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)

## End(Not run)
```

col2int  

*Convert Excel column to integer*

Description

Converts an Excel column label to an integer.

Usage

`col2int(x)`

Arguments

- `x` A character vector

Examples

`col2int(LETTERS)`
Add conditional formatting to cells

Description

DEPRECATED! USE `conditionalFormatting()`

Usage

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

Arguments

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

Details

DEPRECATED! USE `conditionalFormatting()`

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

Author(s)

Alexander Walker

See Also

`createStyle()`
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.
stype Either 'expression', 'colourScale', 'databar', 'duplicates', 'beginsWith', 'endsWith', 'topN', 'bottomN', '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 == "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

```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, "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, "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, "notcontainsText", 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)

## 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
**convertToDate**

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

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

**Usage**  
```r
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**
```r
x <- c(41821.8127314815, 41820.8127314815, NA, 41819, NaN)
convertToDateTime(x)
convertToDateTime(x, tz = "Australia/Perth")
convertToDateTime(x, tz = "UTC")
```

---

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

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

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

**Arguments**
- `wb`: A workbook object

**Value**
Workbook
createComment

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(w2)
names(wb3)
```

createComment create a Comment object

Description

Create a cell Comment object to pass to writeComment()

Usage

```r
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, height**: Width and height of textbook (in number of cells); doubles are rounded with `base::round()`

See Also

- `writeComment()`
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 musts 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)
createNamedRegion

Author(s)

Alexander Walker

See Also

getNamedRegions()

Examples

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

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

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

out_file <- tempfile(fileext = ".xlsx")
## 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  
Cell formatting
  • GENERAL
  • NUMBER
  • CURRENCY
  • ACCOUNTING
  • DATE
  • LONGDATE
createStyle

• 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
  • 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  
• **justify** Justify 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  
• **accounting** Single accounting underline cell contents  
• **accounting2** 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**  

```r  
## 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",  
```
## 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"))
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)
## 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

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

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

```r
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**
```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
freezePane

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)
## End(Not run)
```

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
Examples

```r
## 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**

Return the workbook default font

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)

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

---

table

| 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 Schauberger, Alexander Walker

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

---

table

| 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
getDateOrigin

Value
vector of creators

Author(s)
Philipp Schauburger

Examples

wb <- createWorkbook()
getCreators(wb)

gDateOrigin(xlsxFile)

Arguments
xlsxFile An xlsx or xlsx 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 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()
getNamedRegions

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  Get named regions

Description

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

Usage

getNamedRegions(x)

Arguments

x An xlsx file or Workbook object

See Also

createNamedRegion()

Examples

## 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

```r
getSheetNames(file)
```

### Arguments

- `file` An xlsx or xlsm file.

### Value

Character vector of worksheet names.

### Author(s)

Alexander Walker

### Examples

```r
gsheetNames(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

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

getTables(wb, sheet = "Sheet 1")
```

groupColumns

<table>
<thead>
<tr>
<th>groupColumns</th>
<th>Group columns</th>
</tr>
</thead>
</table>

Description

Group a selection of columns

Usage

```r
groupColumns(wb, sheet, cols, hidden = FALSE)
```

Arguments

- **wb**: A workbook object.
- **sheet**: A name or index of a worksheet.
- **cols**: Indices of cols to group.
- **hidden**: Logical vector. If TRUE the grouped columns are hidden. Defaults to FALSE.

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

See Also

- `ungroupColumns()` to ungroup columns. `groupRows()` for grouping rows.
Description

Group a selection of rows

Usage

\[ \text{groupRows}(\text{wb, sheet, rows, hidden = FALSE}) \]

Arguments

- \text{wb}: A workbook object
- \text{sheet}: A name or index of a worksheet
- \text{rows}: Indices of rows to group
- \text{hidden}: Logical vector. If TRUE the grouped columns are hidden. Defaults to FALSE

Author(s)

Joshua Sturm

See Also

- \text{ungroupRows()}: to ungroup rows. \text{groupColumns()}: for grouping columns.

\[ \text{if\_null\_then} \]

\[ 
\text{If NULL then ...} 
\]

Description

Replace NULL

Usage

\[ \text{x \%||\% y} \]

Arguments

- \text{x}: A value to check
- \text{y}: A value to substitute if \text{x} is null
Examples

```r
## 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**

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

**Arguments**

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

**Author(s)**

Alexander Walker
**insertPlot**

Insert the current plot into a worksheet

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

**Usage**

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

**See Also**

`insertPlot()`

**Examples**

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

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

## Insert images
img <- system.file("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

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>width</td>
<td>Width of figure. Defaults to 6in.</td>
</tr>
<tr>
<td>height</td>
<td>Height of figure. Defaults to 4in.</td>
</tr>
<tr>
<td>xy</td>
<td>Alternate way to specify startRow and startCol. A vector of length 2 of form (startcol, startRow)</td>
</tr>
<tr>
<td>startRow</td>
<td>Row coordinate of upper left corner of figure. xy[[2]] when xy is given.</td>
</tr>
<tr>
<td>startCol</td>
<td>Column coordinate of upper left corner of figure. xy[[1]] when xy is given.</td>
</tr>
<tr>
<td>fileType</td>
<td>File type of image</td>
</tr>
<tr>
<td>units</td>
<td>Units of width and height. Can be &quot;in&quot;, &quot;cm&quot; or &quot;px&quot;</td>
</tr>
<tr>
<td>dpi</td>
<td>Image resolution</td>
</tr>
</tbody>
</table>

Author(s)

Alexander Walker

See Also

insertImage()

Examples

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

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

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

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

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

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

## End(Not run)

---

### int2col

**Convert integer to Excel column**

#### Description

Converts an integer to an Excel column label.

#### Usage

```r
int2col(x)
```

#### Arguments

- `x`  
  A numeric vector

#### Examples

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

---

### loadWorkbook

**Load an existing .xlsx file**

#### Description

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

#### Usage

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

#### Value

Workbook object.
Author(s)

Alexander Walker, Philipp Schaubeger

See Also

`removeWorksheet()`

Examples

```r
## 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

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

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

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

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

## 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,
                                    text = "Link to file"))
```
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
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

See Also

removeCellMerge()
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  get or set worksheet names

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

```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).

Usage

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

Arguments

file

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

Author(s)

Luca Braglia

Examples

```r
# file example
doesn't work yet
example(writeData)

# openXL("writeDataExample.xlsx")

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

---

**openxlsx**

*xlsx reading, writing and editing.*

**Description**

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

**Details**

The openxlsx package uses global options, 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**
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

```r
openxlsxFontSizeLookupTable
```

```r
openxlsxFontSizeLookupTableBold
```

Format

A data.frame with column names corresponding to font names

---

openxlsx_options

*openxlsx Options*

Description

See and get the openxlsx options

Usage

```r
op.openxlsx
```

```r
openxlsx_getOp(x, default = NULL)
```

```r
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"]]

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)
## 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

```r
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**

```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)

## 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

```r
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

```r
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

**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

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**

- **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 param-
eters 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

Formulæ 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**

```r
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**

- `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

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

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.
removeComment

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 Remove a comment from a cell

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

Remove a worksheet filter

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

Remove custom row heights from a worksheet

Description

Remove row heights from a worksheet
removeTable

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 getTables()

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

removeWorksheet

Remove a worksheet from a workbook

Description

Remove a worksheet from a Workbook object

Usage

removeWorksheet(wb, sheet)

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>
</tbody>
</table>
renameWorksheet

Author(s)
Alexander Walker

Examples

```r
## 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 Rename a worksheet

Description

Rename a worksheet

Usage

```r
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

```r
## 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**

*Replace an existing cell style*

### Description

Replace an existing cell style

Replace a style object

### Usage

```r
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
saveWorkbook

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)

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 Schaubberger
setColWidths

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

```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)

## End(Not run)
```
setFooter

Description

DEPRECATED

Usage

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

## 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:
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")
setFooter(wb, "Bottom left", position = "left")
setFooter(wb, Sys.Date(), position = "right")

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

## End(Not run)
**setHeaderFooter**  
*Set document headers and footers*

**Description**

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

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

## End(Not run)

### Description

Add another author to the meta data of the file.

### Usage

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

### Arguments

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

### Author(s)

Philipp Schauberger

### Examples

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

---

setRowHeights

### 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.
sheets

Author(s)
Alexander Walker

See Also
removeRowHeights()

Examples

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

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

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

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

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

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

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

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

## 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

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

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

---

**Description**

DEPRECATED - Use function `sheetVisibility()`

**Usage**

```r
sheetVisible(wb)
```

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

**Arguments**

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

**Value**

Character vector of worksheet names.

- TRUE if sheet is visible, FALSE if sheet is hidden

**Author(s)**

Alexander Walker

**Examples**

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

**helper function to create temporary directory for testing purpose**

---

**temp_xlsx**

**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**

<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>Indices of columns to ungroup</td>
</tr>
</tbody>
</table>

**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**

```r
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**

*Order of worksheets in xlsx file*

**Description**

Get/set order of worksheets in a Workbook object

**Usage**

```r
worksheetOrder(wb)

worksheetOrder(wb) <- value
```

**Arguments**

- `wb`: A workbook object
- `value`: Vector specifying order to write worksheets to file
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 writeData() or writeDataTable() to write to file

file A file path to save the xlsx file
asTable  If TRUE will use `writeDataTable()` rather than `writeData()` to write x to the file (default: FALSE)
overwrite  Overwrite existing file (Defaults to TRUE as with write.table) ...

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**

```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")

## Not run:
write.xlsx(iris, file = "writeXLSX3.xlsx", colNames = TRUE, borders = "rows", headerStyle = hs)

## Lists elements are written to individual worksheets, using list names as sheet names if available
l <- list("IRIS" = iris, "MTCATS" = mtcars, matrix(runif(1000), ncol = 5))
## Not run:
write.xlsx(l, "writeList1.xlsx", colWidths = c(NA, "auto", "auto"))
```

## different sheets can be given different parameters
## Not run:
write.xlsx(1, "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(1, "writeList2.xlsx", colWidths = 20)
write.xlsx(1, "writeList2.xlsx", colWidths = list(100, 200, 300))
write.xlsx(1, "writeList2.xlsx", colWidths = list(rep(10, 5), rep(8, 11), rep(5, 5)))
## End(Not run)

writeComment

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

**Description**

Write a Comment object to a worksheet

**Usage**

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

**Arguments**

- **wb**: A workbook object
- **sheet**: A vector of names or indices of worksheets
- **col**: Column a column number of letter
- **row**: A row number.
- **comment**: A Comment object. See `createComment()`.
- **xy**: An alternative to specifying col and row individually. A vector of the form c(col, row).

**See Also**

`createComment()`
writeData

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,
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 respective ab-
            breviations. If "surrounding", a border is drawn around the data. If "rows",
            a surrounding border is drawn with a border around each row. If "columns", a
            surrounding border is drawn with a border between each column. If "all" all
            cell borders are drawn.
borderColour Colour of cell border. A valid colour (belonging to colours() or a hex colour
            code, eg see here).
borderStyle Border line style
            • none no border
            • thin thin border
            • medium medium border
            • dashed dashed border
            • dotted dotted border
            • thick thick border
            • double double line border
            • hair hairline border
            • mediumDashed medium weight dashed border
            • dashDot dash-dot border
            • mediumDashDot medium weight dash-dot border
            • dashDotDot dash-dot-dot border
            • mediumDashDotDot medium weight dash-dot-dot border
            • slantDashDot slanted dash-dot border
withFilter If TRUE 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.
writeData

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
Depreciated, 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 = "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'

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

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 = openxlsx_getOp("tableStyle", "TableStyleLight9"))
writeDataTable

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:
writeDataTable

- **Header Row**: logical. If TRUE, the first row is bold
- **First Column**: logical. If TRUE, the first column is bold
- **Filter Button**: logical. If TRUE, the column is filterable
- **Total Row**: logical. If TRUE, the last row is bold
- **Last Column**: logical. If TRUE, the last column is bold
- **Banded Rows**: logical. If TRUE, rows are colour banded
- **Banded Columns**: 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,
  "Category" = "Sales",
  "Amount" = c(1234, 5678, 9012, 3456, 7890, 1234, 5678, 9012, 3456, 7890),
  "Currency" = c("USD", "USD", "USD", "USD", "USD", "USD", "USD", "USD", "USD", "USD"),
  "Date" = Sys.Date() - 0:10,
  "Category" = "Finance",
  "Amount" = c(1111, 2222, 3333, 4444, 5555, 6666, 7777, 8888, 9999),
)

df <- df[nrow(df), ]
```

```excel
<table>
<thead>
<tr>
<th>Date</th>
<th>Category</th>
<th>Amount</th>
<th>Currency</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/31/19</td>
<td>Sales</td>
<td>1234</td>
<td>USD</td>
</tr>
<tr>
<td>12/30/19</td>
<td>Sales</td>
<td>5678</td>
<td>USD</td>
</tr>
<tr>
<td>12/29/19</td>
<td>Sales</td>
<td>9012</td>
<td>USD</td>
</tr>
<tr>
<td>12/28/19</td>
<td>Sales</td>
<td>3456</td>
<td>USD</td>
</tr>
<tr>
<td>12/27/19</td>
<td>Sales</td>
<td>7890</td>
<td>USD</td>
</tr>
<tr>
<td>12/26/19</td>
<td>Sales</td>
<td>1234</td>
<td>USD</td>
</tr>
<tr>
<td>12/25/19</td>
<td>Sales</td>
<td>5678</td>
<td>USD</td>
</tr>
<tr>
<td>12/24/19</td>
<td>Sales</td>
<td>9012</td>
<td>USD</td>
</tr>
<tr>
<td>12/23/19</td>
<td>Sales</td>
<td>3456</td>
<td>USD</td>
</tr>
<tr>
<td>12/22/19</td>
<td>Sales</td>
<td>7890</td>
<td>USD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date</th>
<th>Category</th>
<th>Amount</th>
<th>Currency</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/30/19</td>
<td>Finance</td>
<td>1111</td>
<td>EUR</td>
</tr>
<tr>
<td>11/29/19</td>
<td>Finance</td>
<td>2222</td>
<td>EUR</td>
</tr>
<tr>
<td>11/28/19</td>
<td>Finance</td>
<td>3333</td>
<td>EUR</td>
</tr>
<tr>
<td>11/27/19</td>
<td>Finance</td>
<td>4444</td>
<td>EUR</td>
</tr>
<tr>
<td>11/26/19</td>
<td>Finance</td>
<td>5555</td>
<td>EUR</td>
</tr>
<tr>
<td>11/25/19</td>
<td>Finance</td>
<td>6666</td>
<td>EUR</td>
</tr>
<tr>
<td>11/24/19</td>
<td>Finance</td>
<td>7777</td>
<td>EUR</td>
</tr>
<tr>
<td>11/23/19</td>
<td>Finance</td>
<td>8888</td>
<td>EUR</td>
</tr>
<tr>
<td>11/22/19</td>
<td>Finance</td>
<td>9999</td>
<td>EUR</td>
</tr>
</tbody>
</table>
```
"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 = 1, startCol = i * 3 - 2
    )
}
writeFormula(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**

*Write a character vector as an Excel Formula*

**Description**

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

**Usage**

```r
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
)

---

writeFormula
```r
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)

## 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, "writeFormulaExample.xlsx", overwrite = TRUE)

## End(Not run)
```
Index

* datasets
  openxlsx_options, 53
  openxlsxFontSizeLookupTable, 53

activeSheet, 4
activeSheet<- (activeSheet), 4
addCreator, 5
addFilter, 5
addFilter(), 6
addStyle, 6
addStyle(), 28
addWorksheet, 7
addWorksheet(), 54, 74, 79, 89, 97
all.equal, 10

buildWorkbook, 10
buildWorkbook(), 88, 89

cloneworksheet, 12
col2int, 13
conditionalFormat, 14
conditionalFormatting, 15
conditionalFormatting(), 14
convertFromExcelRef, 20
convertToDate, 21
convertToDate(), 36
convertToDateTime, 22
copyWorkbook, 22
createComment, 23
createComment(), 67, 90
createNamedRegion, 24
createNamedRegion(), 37
createStyle, 26
createStyle(), 7, 14–16, 23, 89
createWorkbook, 29
createWorkbook(), 74

databar (conditionalFormatting), 15
dataValidation, 30
deletedata, 32
deleteNamedRegion (createNamedRegion),
24

freezePane, 33

getBaseFont, 34
getCellRefs, 35
getCreators, 35
generateOrigin, 36
getNamedRegions, 37
getNamedRegions(), 25, 63, 65
getSheetNames, 38
getStyles, 39
getStyles(), 73
getTables, 39
getTables(), 69, 97
groupColumns, 40
groupColumns(), 41
groupRows, 41
groupRows(), 40

if_null_then, 41
insertImage, 42
insertImage(), 44
insertPlot, 43
insertPlot(), 43
int2col, 45

loadWorkbook, 45
loadWorkbook(), 30, 74

makeHyperlinkString, 46
makeHyperlinkString(), 100
mergeCells, 48
mergeCells(), 66
modifyBaseFont, 49

names, 50
names(), 71, 81, 82
names<-.Workbook (names), 50
INDEX

op.openxlsx (openxlsx_options), 53
op.openxlsx(), 53
openXL, 51
openxlsx, 52
openxlsx_getOp (openxlsx_options), 53
openxlsx_options, 53
openxlsx_setOp (openxlsx_options), 53
openxlsxFontSizeLookupTable, 53
openxlsxFontSizeLookupTableBold (openxlsxFontSizeLookupTable), 53
pageBreak, 54
pageSetup, 55
protectWorkbook, 59
protectWorksheet, 60
read.xlsx, 62
read.xlsx(), 53, 65
readWorkbook, 64
removeCellMerge, 66
removeCellMerge(), 49
removeColWidths, 66
removeColWidths(), 75
removeComment, 67
removeFilter, 68
removeRowHeights, 68
removeRowHeights(), 81
removeTable, 69
removeTable(), 97
removeWorksheet, 70
removeWorksheet(), 46
renameWorksheet, 71
replaceStyle, 72
replaceStyle(), 39
saveWorkbook, 73
saveWorkbook(), 30
setColWidths, 74
setColWidths(), 11, 40, 67
setFooter, 76
setHeader, 77
setHeaderFooter, 78
setLastModifiedBy, 80
setRowHeights, 80
setRowHeights(), 69
sheets, 81
sheetVisibility, 82
sheetVisibility<-(sheetVisibility), 83
showGridLines, 84
temp.xlsx, 85
ungroupColumns, 85
ungroupColumns(), 40, 86
ungroupRows, 86
ungroupRows(), 41, 85
worksheetOrder, 86
worksheetOrder<-(worksheetOrder), 86
write.xlsx, 87
write.xlsx(), 12, 53
writeComment, 90
writeComment(), 23, 67
writeData, 91
writeData(), 6, 11, 21, 53, 74, 87–89, 97, 100
writeDataTable, 95
writeDataTable(), 11, 53, 74, 87, 88, 93
writeFormula, 99
writeFormula(), 46