Package ‘openxlsx2’

December 18, 2022

Type  Package
Title  Read, Write and Edit ‘xlsx’ Files
Version  0.4.1
Language  en-US
Description  Simplifies the creation of ‘xlsx’ files by providing a high level interface to writing, styling and editing worksheets.
License  MIT + file LICENSE
URL  https://github.com/JanMarvin/openxlsx2
BugReports  https://github.com/JanMarvin/openxlsx2/issues
Depends  R (>= 3.4.0)
Imports  R6, Rcpp, grDevices, magrittr, stringi, utils, zip
LinkingTo  Rcpp
Suggests  covr, ggplot2, knitr, mschart (>= 0.4), rmarkdown, roxygen2, rvg, testthat (>= 3.0.0), waldo
VignetteBuilder  knitr
Encoding  UTF-8
RoxygenNote  7.2.3
SystemRequirements  C++11
Config/testthat.edition  3
Config/testthat/parallel  true
Config/testthat/start-first  read_sources
NeedsCompilation  yes
Author  Jordan Mark Barbone [aut] (<https://orcid.org/0000-0001-9788-3628>), Jan Marvin Garbuszus [aut, cre], openxlsx authors [cph] (openxlsx package), Arseny Kapoulkine [ctb, cph] (Author of included pugxml code)
Maintainer  Jan Marvin Garbuszus <jan.garbuszus@ruhr-uni-bochum.de>
Repository  CRAN
Date/Publication  2022-12-18 09:50:02 UTC
R topics documented:

- as_xml .................................................. 4
- cell_style ............................................ 5
- cleanup ................................................. 6
- clean_worksheet_name .............................. 7
- cloneSheetStyle ..................................... 7
- col2int ................................................. 8
- convertToExcelDate ................................. 8
- convert_date ......................................... 9
- convert_datetime .................................... 9
- create_border ....................................... 10
- create_cell_style .................................. 11
- create_comment ..................................... 13
- create_dxf_style .................................... 15
- create_fill ........................................... 16
- create_font ......................................... 17
- create_hyperlink .................................... 18
- create_numfmt ...................................... 20
- create.sparklines ................................... 21
- dataframe_to_dims .................................. 22
- dims_to_dataframe .................................. 23
- get_cell.refs ....................................... 23
- get_cell_style ...................................... 24
- get_date_origin ..................................... 24
- guess_col_type ..................................... 25
- int2col ............................................... 26
- NamedRegions ....................................... 26
- named_region ....................................... 27
- numfmt.is_date ..................................... 29
- numfmt.is_posix ..................................... 30
- openxlsx2 ........................................... 30
- print.pugi.xml ....................................... 31
- pugixml ............................................... 32
- read_sheet_names ................................... 33
- read.xlsx ............................................. 34
- read_xml .............................................. 36
- select_active_sheet ............................... 37
- set_cell_style ...................................... 38
- sheet_visibility ..................................... 39
- styles_on_sheet ..................................... 40
- style.is_date ....................................... 40
- style.is_posix ....................................... 41
- style_mgr ............................................. 41
- temp.xlsx ............................................. 46
- wbChartSheet ........................................ 46
- wbComment .......................................... 48
- wbHyperlink ......................................... 50
<table>
<thead>
<tr>
<th>function</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>wbSheetData</td>
<td>51</td>
</tr>
<tr>
<td>wbWorkbook</td>
<td>52</td>
</tr>
<tr>
<td>wbWorksheet</td>
<td>84</td>
</tr>
<tr>
<td>wb_add_border</td>
<td>90</td>
</tr>
<tr>
<td>wb_add_cell_style</td>
<td>91</td>
</tr>
<tr>
<td>wb_add_chartsheet</td>
<td>93</td>
</tr>
<tr>
<td>wb_add_chart_xml</td>
<td>94</td>
</tr>
<tr>
<td>wb_add_conditional_formatting</td>
<td>94</td>
</tr>
<tr>
<td>wb_add_data</td>
<td>97</td>
</tr>
<tr>
<td>wb_add_data_table</td>
<td>100</td>
</tr>
<tr>
<td>wb_add_data_validation</td>
<td>102</td>
</tr>
<tr>
<td>wb_add_drawing</td>
<td>104</td>
</tr>
<tr>
<td>wb_add_fill</td>
<td>105</td>
</tr>
<tr>
<td>wb_add_filter</td>
<td>106</td>
</tr>
<tr>
<td>wb_add_font</td>
<td>107</td>
</tr>
<tr>
<td>wb_add_formula</td>
<td>109</td>
</tr>
<tr>
<td>wb_add_image</td>
<td>110</td>
</tr>
<tr>
<td>wb_add_mschart</td>
<td>111</td>
</tr>
<tr>
<td>wb_add_numfmt</td>
<td>112</td>
</tr>
<tr>
<td>wb_add_page_break</td>
<td>113</td>
</tr>
<tr>
<td>wb_add_plot</td>
<td>114</td>
</tr>
<tr>
<td>wb_add_sparklines</td>
<td>115</td>
</tr>
<tr>
<td>wb_add_style</td>
<td>116</td>
</tr>
<tr>
<td>wb_add_worksheet</td>
<td>117</td>
</tr>
<tr>
<td>wb_clone_sheet_style</td>
<td>119</td>
</tr>
<tr>
<td>wb_clone_worksheet</td>
<td>120</td>
</tr>
<tr>
<td>wb_colour</td>
<td>121</td>
</tr>
<tr>
<td>wb_creators</td>
<td>121</td>
</tr>
<tr>
<td>wb_data</td>
<td>122</td>
</tr>
<tr>
<td>wb_freeze_pane</td>
<td>123</td>
</tr>
<tr>
<td>wb_get_base_font</td>
<td>124</td>
</tr>
<tr>
<td>wb_get_sheet_name</td>
<td>125</td>
</tr>
<tr>
<td>wb_get_sheet_names</td>
<td>125</td>
</tr>
<tr>
<td>wb_get_tables</td>
<td>126</td>
</tr>
<tr>
<td>wb_get_worksheet</td>
<td>126</td>
</tr>
<tr>
<td>wb_grid_lines</td>
<td>127</td>
</tr>
<tr>
<td>wb_hyperlink</td>
<td>127</td>
</tr>
<tr>
<td>wb_load</td>
<td>128</td>
</tr>
<tr>
<td>wb_modify_basefont</td>
<td>129</td>
</tr>
<tr>
<td>wb_open</td>
<td>130</td>
</tr>
<tr>
<td>wb_order</td>
<td>130</td>
</tr>
<tr>
<td>wb_protect</td>
<td>131</td>
</tr>
<tr>
<td>wb_protect_worksheet</td>
<td>132</td>
</tr>
<tr>
<td>wb_read</td>
<td>133</td>
</tr>
<tr>
<td>wb_remove_col_widths</td>
<td>135</td>
</tr>
<tr>
<td>wb_remove_filter</td>
<td>136</td>
</tr>
<tr>
<td>wb_remove_row_heights</td>
<td>136</td>
</tr>
<tr>
<td>wb_remove_tables</td>
<td>137</td>
</tr>
</tbody>
</table>
as_xml

loads character string to pugixml and returns an externalptr

Description
loads character string to pugixml and returns an externalptr

Usage
as_xml(x, ...)

Arguments
x             input as xml
...          additional arguments passed to read_xml()

Details
might be useful for larger documents where single nodes are shortened and otherwise the full tree has to be reimported. unsure where we have such a case. is useful, for printing nodes from a larger tree, that have been exported as characters (at some point in time we have to convert the xml to R)
** Examples **

tmp_xlsx <- tempfile()
xlsxFile <- system.file("extdata", "readTest.xlsx", package = "openxlsx2")
unzip(xlsxFile, exdir = tmp_xlsx)

wb <- wb_load(xlsxFile)
styles_xml <- sprintf("%s/xl/styles.xml", tmp_xlsx)

# is external pointer
sxml <- read_xml(styles_xml)

# is character
font <- xml_node(sxml, "styleSheet", "fonts", "font")

# is again external pointer
as_xml(font)

---

**cell_style**

** get and set cell style **

** Description **

get and set cell style

** Usage **

wb_get_cell_style(wb, sheet = current_sheet(), dims)

wb_set_cell_style(wb, sheet = current_sheet(), dims, style)

** Arguments **

- wb
- sheet
- dims
- style

** Value **

wb_get_cell_style returns the style id as character

wb_set_cell_style returns the workbook invisible
Examples

```r
# set a style in B1
wb <- wb_workbook()$add_worksheet()
    add_numfmt(dims = "B1", numfmt = ",0")

# get style from B1 to assign it to A1
numfmt <- wb$get_cell_style(dims = "B1")

# assign style to A1
wb$set_cell_style(dims = "A1", style = numfmt)
```

cleanup

`wb_clean_sheet`

Usage

```r
wb_clean_sheet(
    wb, sheet = current_sheet(),
    numbers = TRUE,
    characters = TRUE,
    styles = TRUE,
    merged_cells = TRUE
)
```

data(wb, sheet, cols, rows)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wb</td>
<td>workbook</td>
</tr>
<tr>
<td>sheet</td>
<td>sheet to clean</td>
</tr>
<tr>
<td>numbers</td>
<td>remove all numbers</td>
</tr>
<tr>
<td>characters</td>
<td>remove all characters</td>
</tr>
<tr>
<td>styles</td>
<td>remove all styles</td>
</tr>
<tr>
<td>merged_cells</td>
<td>remove all merged_cells</td>
</tr>
<tr>
<td>cols</td>
<td>numeric column vector</td>
</tr>
<tr>
<td>rows</td>
<td>numeric row vector</td>
</tr>
</tbody>
</table>
**clean_worksheet_name**

**Description**

Cleans a worksheet name by removing legal characters.

**Usage**

`clean_worksheet_name(x, replacement = " ")`

**Arguments**

- `x`: A vector, coerced to character
- `replacement`: A single value to replace illegal characters by.

**Details**

Illegal characters are considered \\, /, ?, *, : [, and ]. These must be intentionally removed from worksheet names prior to creating a new worksheet.

**Value**

`x` with bad characters removed

---

**cloneSheetStyle**

**Description**

clone sheets style

**Usage**

`cloneSheetStyle(wb, from_sheet, to_sheet)`

**Arguments**

- `wb`: workbook
- `from_sheet`: sheet we select the style from
- `to_sheet`: sheet we apply the style from
**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)
```

**convertToExcelDate**  
convert back to ExcelDate

**Description**  
convert back to ExcelDate

**Usage**  
```
convertToExcelDate(df, date1904 = FALSE)
```

**Arguments**  
`df`  
dataframe
`date1904`  
take different origin

**Examples**  
```
xlsxFile <- system.file("extdata", "readTest.xlsx", package = "openxlsx2")
wb1 <- wb_load(xlsxFile)
df <- wb_to_df(wb1)
# conversion is done on dataframes only
convertToExcelDate(df = df["Var5"], date1904 = FALSE)
```
**convert_date**

Convert from excel date number to R Date type

**Description**

Convert from excel date number to R Date type

**Usage**

```r
convert_date(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**

`write_data()`

**Examples**

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

---

**convert_datetime**

Convert from excel time number to R POSIXct type.

**Description**

Convert from excel time number to R POSIXct type.

**Usage**

```r
convert_datetime(x, origin = "1900-01-01", ...)
```

**Arguments**

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

Excel stores dates as number of days from some origin date

Examples

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

create_border  create border

Description

Border styles can be any of the following: "thin", "thick", "slantDashDot", "none", "mediumDashed", "mediumDashDot", "medium", "hair", "double", "dotted", "dashed", "dashedDotDot", "dashDot"

Border colors are of the following type: c(rgb="FF000000")

Usage

```r
create_border(
    diagonalDown = "",
    diagonalUp = "",
    outline = "",
    bottom = NULL,
    bottom_color = NULL,
    diagonal = NULL,
    diagonal_color = NULL,
    end = "",
    horizontal = "",
    left = NULL,
    left_color = NULL,
    right = NULL,
    right_color = NULL,
    start = "",
    top = NULL,
    top_color = NULL,
    vertical = ""
)
```

Arguments

- `diagonalDown` x
- `diagonalUp` x
- `outline` x
create_cell_style

bottom X
bottom_color X
diagonal X
diagonal_color X,
end x,
horizontal x
left x
left_color x
right x
right_color x
start x
top x
top_color x
vertical x

create_cell_style  create_cell_style

Description

create_cell_style

Usage

create_cell_style(
    borderId = "",
    fillId = "",
    fontId = "",
    numFmtId = "",
    pivotButton = "",
    quotePrefix = "",
    xfId = "",
    horizontal = "",
    indent = "",
    justifyLastLine = "",
    readingOrder = "",
    relativeIndent = "",
    shrinkToFit = "",
    textRotation = "",
    vertical = "",
    wrapText = "",
    extLst = "",
    hidden = "",
    locked = ""
)
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>borderId</td>
<td>dummy</td>
</tr>
<tr>
<td>fillId</td>
<td>dummy</td>
</tr>
<tr>
<td>fontId</td>
<td>dummy</td>
</tr>
<tr>
<td>numFmtId</td>
<td>a numFmt ID for a builtin style</td>
</tr>
<tr>
<td>pivotButton</td>
<td>dummy</td>
</tr>
<tr>
<td>quotePrefix</td>
<td>dummy</td>
</tr>
<tr>
<td>xfId</td>
<td>dummy</td>
</tr>
<tr>
<td>horizontal</td>
<td>alignment can be &quot;&quot;, &quot;center&quot;, &quot;right&quot;</td>
</tr>
<tr>
<td>indent</td>
<td>dummy</td>
</tr>
<tr>
<td>justifyLastLine</td>
<td>dummy</td>
</tr>
<tr>
<td>readingOrder</td>
<td>dummy</td>
</tr>
<tr>
<td>relativeIndent</td>
<td>dummy</td>
</tr>
<tr>
<td>shrinkToFit</td>
<td>dummy</td>
</tr>
<tr>
<td>textRotation</td>
<td>dummy</td>
</tr>
<tr>
<td>vertical</td>
<td>alignment can be &quot;&quot;, &quot;center&quot;, &quot;right&quot;</td>
</tr>
<tr>
<td>wrapText</td>
<td>dummy</td>
</tr>
<tr>
<td>extLst</td>
<td>dummy</td>
</tr>
<tr>
<td>hidden</td>
<td>dummy</td>
</tr>
<tr>
<td>locked</td>
<td>dummy</td>
</tr>
</tbody>
</table>

Details

<table>
<thead>
<tr>
<th>ID</th>
<th>numFmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>General</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>#.##0</td>
</tr>
<tr>
<td>4</td>
<td>#.##0.00</td>
</tr>
<tr>
<td>9</td>
<td>0%</td>
</tr>
<tr>
<td>10</td>
<td>0.00%</td>
</tr>
<tr>
<td>11</td>
<td>0.00E+00</td>
</tr>
<tr>
<td>12</td>
<td># ??!</td>
</tr>
<tr>
<td>13</td>
<td># ?? ?? ??</td>
</tr>
<tr>
<td>14</td>
<td>mm-dd-yy</td>
</tr>
<tr>
<td>15</td>
<td>d-mmm-yy</td>
</tr>
<tr>
<td>16</td>
<td>d-mmm</td>
</tr>
<tr>
<td>17</td>
<td>mmm-yy</td>
</tr>
<tr>
<td>18</td>
<td>h:mm AM/PM</td>
</tr>
<tr>
<td>19</td>
<td>h:mm:ss AM/PM</td>
</tr>
<tr>
<td>20</td>
<td>h:mm</td>
</tr>
</tbody>
</table>
create_comment

Create, write and remove comments

Description

The comment functions (create, write and remove) allow the modification of comments. In newer Excels they are called notes, while they are called comments in openxml. Modification of what Excel now calls comment (openxml calls them threadedComments) is not yet possible

Usage

create_comment(
  text,
  author = Sys.info()["user"],
  style = NULL,
  visible = TRUE,
  width = 2,
  height = 4
)

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

remove_comment(wb, sheet, col, row, gridExpand = TRUE)

wb_add_comment(
  wb,
  sheet = current_sheet(),
  col,
  row,
  dims = rowcol_to_dims(row, col),
  comment
)
create_comment

create_comment

wb_remove_comment(
  wb,
  sheet = current_sheet(),
  col,
  row,
  dims = rowcol_to_dims(row, col),
  gridExpand = TRUE
)

Arguments

text 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.
visible TRUE or FALSE. Is comment visible.
width Textbox integer width in number of cells
height Textbox integer height in number of cells
wb A workbook object
sheet A worksheet of the workbook
col A column to apply the comment
row A row to apply the comment
comment A comment to apply to the worksheet
xy An alternative to specifying col and row individually. A vector of the form c(col, row).
gridExpand Remove all comments inside the grid. Similar to dims "A1:B2"
dims Optional row and column as spreadsheet dimension, e.g. "A1"

Value

The wbWorkbook object

Examples

wb <- wb_workbook()
bw$add_worksheet("Sheet 1")

# write comment without author
c1 <- create_comment(text = "this is a comment", author = "")
write_comment(wb, 1, col = "B", row = 10, comment = c1)

# Write another comment with author information
c2 <- create_comment(text = "this is another comment", author = "Marco Polo")
write_comment(wb, 1, col = "C", row = 10, comment = c2)
# write a styled comment with system author
s1 <- create_font(b = "true", color = wb_colour(hex = "FFFF0000"), sz = "12")
s2 <- create_font(color = wb_colour(hex = "FF000000"), sz = "9")
c3 <- create_comment(text = c("This Part Bold red\n\n", "This part black"), style = c(s1, s2))

write_comment(wb, 1, col = 6, row = 3, comment = c3)

# remove the first comment
remove_comment(wb, 1, col = "B", row = 10)

---

create_dxfs_style Create a custom formatting style

**Description**

Create a new style to apply to worksheet cells. Created styles have to be assigned to a workbook to use them

**Usage**

create_dxfs_style(
  font_name = NULL,
  font_size = NULL,
  font_color = NULL,
  numFmt = NULL,
  border = NULL,
  border_color = wb_colour(getOption("openxlsx2.borderColour", "black")),
  border_style = getOption("openxlsx2.borderStyle", "thin"),
  bgFill = NULL,
  text_bold = NULL,
  text_strike = NULL,
  text_italic = NULL,
  text_underline = NULL
)

**Arguments**

- **font_name** 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)
- **font_size** Font size. A numeric greater than 0. If fontSize is NULL, the workbook base font size is used. (Defaults to 11)
- **font_color** 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. Some custom openxml format
- **border** NULL or TRUE
create_fill

| border_color | "black" |
| border_style | "thin" |
| bgFill        | Cell background fill color. |
| text_bold    | bold |
| text_strike  | strikeout |
| text_italic  | italic |
| text_underline | underline 1, true, single or double |

**Value**

A dxfs style node

**See Also**

`wb_add_style()`

**Examples**

```r
# do not apply anthing
style1 <- create_dxfs_style()

# change font color and background color
style2 <- create_dxfs_style(
  font_color = wb_colour(hex = "FF9C0006"),
  bgFill = wb_colour(hex = "FFFFC7CE")
)

# change font (type, size and color) and background
# the old default in openxlsx and openxlsx2 <= 0.3
style3 <- create_dxfs_style(
  font_name = "Calibri",
  font_size = 11,
  font_color = wb_colour(hex = "FF9C0006"),
  bgFill = wb_colour(hex = "FFFFC7CE")
)
```

```
## See package vignettes for further examples
```

**Description**

create fill
Usage

```r
create_fill(  
    gradientFill = "",  
    patternType = "",  
    bgColor = NULL,  
    fgColor = NULL  
)
```

Arguments

- **gradientFill**: complex fills
- **patternType**: various default is "none", but also "solid", or a color like "gray125"
- **bgColor**: hex8 color with alpha, red, green, blue only for patternFill
- **fgColor**: hex8 color with alpha, red, green, blue only for patternFill

---

**create_font**

Create font format

Usage

```r
create_font(  
    b = "",  
    charset = "",  
    color = wb_colour(hex = "FF000000"),  
    condense = "",  
    extend = "",  
    family = "2",  
    i = "",  
    name = "Calibri",  
    outline = "",  
    scheme = "minor",  
    shadow = "",  
    strike = "",  
    sz = "11",  
    u = "",  
    vertAlign = ""  
)
```
Argumets

b  bold
charset  charset
color  rgb color: default "FF000000"
condense  condense
extend  extend
family  font family: default "2"
i  italic
name  font name: default "Calibri"
outline  outline
scheme  font scheme: default "minor"
shadow  shadow
strike  strike
sz  font size: default "11",
u  underline
vertAlign  vertical alignment

Examples

font <- create_font()
# openxml has the alpha value leading
hex8 <- unlist(xml_attr(read_xml(font), "font", "color"))
hex8 <- paste0("#", substr(hex8, 3, 8), substr(hex8, 1,2))

# write test color
# col <- crayon::make_style(col2rgb(hex8, alpha = TRUE))
# cat(col("Test"))

create_hyperlink  create Excel hyperlink string

Description

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

Usage

create_hyperlink(sheet, row = 1, col = 1, text = NULL, file = NULL)
create_hyperlink

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

- `write_formula()`

Examples

```r
## Writing internal hyperlinks
wb <- wb_workbook()
wbs$add_worksheet("Sheet1")
wbs$add_worksheet("Sheet2")
wbs$add_worksheet("Sheet 3")
wbs$add_data(sheet = 3, x = iris)

## External Hyperlink
names(x) <- c("google", "google Aus")
class(x) <- "hyperlink"

wbs$add_data(sheet = 1, x = x, startCol = 10)

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

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

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

## Description
create number format

## Usage

create_numfmt(numFmtId, formatCode)

## Arguments

numFmtId an id
formatCode a format code
create_sparklines  
create sparklines used in add_sparkline()

Description

create sparklines used in add_sparkline()

Usage

create_sparklines(
    sheet = current_sheet(),
    dims,
    sqref,
    type = NULL,
    negative = NULL,
    displayEmptyCellsAs = "gap",
    markers = NULL,
    high = NULL,
    low = NULL,
    first = NULL,
    last = NULL,
    colorSeries = wb_colour(hex = "FF376092"),
    colorNegative = wb_colour(hex = "FFD00000"),
    colorAxis = wb_colour(hex = "FFD00000"),
    colorMarkers = wb_colour(hex = "FFD00000"),
    colorFirst = wb_colour(hex = "FFD00000"),
    colorLast = wb_colour(hex = "FFD00000"),
    colorHigh = wb_colour(hex = "FFD00000"),
    colorLow = wb_colour(hex = "FFD00000")
)

Arguments

sheet    sheet
dims     dims
sqref    sqref
type     type
negative negative
displayEmptyCellsAs displayEmptyCellsAs
markers markers add marker to line
high     highlight highest value
low      highlight lowest value
first    highlight first value
```r
last   highlight last value
colorSeries  colorSeries
colorNegative  colorNegative
colorAxis  colorAxis
colorMarkers  colorMarkers
colorFirst  colorFirst
colorLast  colorLast
colorHigh  colorHigh
colorLow  colorLow
```

**Details**

the colors are all predefined to be rgb. Maybe theme colors can be used too.

**Examples**

```r
# create sparklineGroup
sparklines <- c(
  create_sparklines("Sheet 1", "A3:L3", "M3", type = "column", first = "1"),
  create_sparklines("Sheet 1", "A2:L2", "M2", markers = "1"),
  create_sparklines("Sheet 1", "A4:L4", "M4", type = "stacked", negative = "1")
)

t1 <- AirPassengers
t2 <- do.call(cbind, split(t1, cycle(t1)))
dimnames(t2) <- dimnames(.preformat.ts(t1))

wb <- wb_workbook()
  add_worksheet("Sheet 1")
  add_data(x = t2)
  add_sparklines(sparklines = sparklines)
```

dataframe_to_dims  
create dimensions from dataframe

**Description**

create dimensions from dataframe

**Usage**

dataframe_to_dims(df)

**Arguments**

df  
dataframe with spreadsheet columns and rows
**dims_to_dataframe**

create dataframe from dimensions

**Description**

create dataframe from dimensions

**Usage**

dims_to_dataframe(dims, fill = FALSE)

**Arguments**

dims Character vector of expected dimension.
fill If TRUE, fills the dataframe with variables

**Examples**

```
{ 
  df <- dims_to_dataframe("A1:D5;F1:F6;D8", fill = TRUE)
  dataframe_to_dims(df)
}
```

---

**get_cell_refs**

Return excel cell coordinates from (x,y) coordinates

**Description**

Return excel cell coordinates from (x,y) coordinates

**Usage**

get_cell.refs(cellCoords)

**Arguments**

cellCoords A data.frame with two columns coordinate pairs.

**Value**

Excel alphanumeric cell reference
Examples

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

get_cell_style
helper get_cell_style

Description

helper get_cell_style

Usage

get_cell_style(wb, sheet, cell)

Arguments

- `wb`: a workbook
- `sheet`: a worksheet
- `cell`: a cell

get_date_origin
Get the date origin an xlsx file is using

Description

Return the date origin used internally by an xlsx or xlsxm file

Usage

get_date_origin(xlsxFile, origin = FALSE)

Arguments

- `xlsxFile`: An xlsx or xlsxm file or a wbWorkbook object.
- `origin`: return the origin instead of the character string.

Details

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

Value

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

See Also

convert_date()

Examples

## create a file with some dates
temp <- temp_xlsx()
write_xlsx(as.Date("2015-01-10") - (0:4), file = temp)
m <- read_xlsx(temp)

## convert to dates
do <- get_date_origin(system.file("extdata", "readTest.xlsx", package = "openxlsx2"))
convert_date(m[[1]], do)

get_date_origin(wb_workbook())
get_date_origin(wb_workbook(), origin = TRUE)

---

guess_col_type | function to estimate the column type. 0 = character, 1 = numeric, 2 = date.

Description

function to estimate the column type. 0 = character, 1 = numeric, 2 = date.

Usage

guess_col_type(tt)

Arguments

| tt         | dataframe produced by wb_to_df() |
### int2col

**Convert integer to Excel column**

**Description**

Converts an integer to an Excel column label.

**Usage**

```
int2col(x)
```

**Arguments**

- `x`: A numeric vector

**Examples**

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

### NamedRegions

**Get create or remove named regions**

**Description**

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

**Usage**

```
get_named_regions(x)
```

**Arguments**

- `x`: An xlsx file or Workbook object

**See Also**

`wb_add_named_region()`, `wb_remove_named_region()`
Examples

```r
## create named regions
wb <- wb_workbook()
wb$add_worksheet("Sheet 1")

## specify region
wb$add_data(sheet = 1, x = iris, startCol = 1, startRow = 1)
wb$add_named_region(
  sheet = 1,
  name = "iris",
  rows = seq_len(nrow(iris) + 1),
  cols = seq_along(iris)
)

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

out_file <- temp_xlsx()
wb$save(out_file, overwrite = TRUE)

## see named regions
get_named_regions(wb) ## From Workbook object
get_named_regions(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)
```

---

### named_region

#### Create / delete a named region

**Description**

Create / delete a named region

**Usage**

```r
wb_add_named_region(
  wb,
  sheet = current_sheet(),
  cols,
  rows,
  name,
  localSheet = FALSE,
  overwrite = FALSE,
```
comment = NULL,
customMenu = NULL,
description = NULL,
is_function = NULL,
functionGroupId = NULL,
help = NULL,
hidden = NULL,
localName = NULL,
publishToServer = NULL,
statusBar = NULL,
vbProcedure = NULL,
workbookParameter = NULL,
xml = NULL
)

wb_remove_named_region(wb, sheet = current_sheet(), name = NULL)

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.

localSheet If TRUE the named region will be local for this sheet

overwrite Boolean. Overwrite if exists? Default to FALSE

comment description text for named region

customMenu customMenu (unknown xml feature)

description description (unknown xml feature)

is_function function (unknown xml feature)

functionGroupId function group id (unknown xml feature)

help help (unknown xml feature)

hidden hidden if the named region should be hidden

localName localName (unknown xml feature)

publishToServer publish to server (unknown xml feature)

statusBar status bar (unknown xml feature)

vbProcedure wbProcedure (unknown xml feature)

workbookParameter workbookParameter (unknown xml feature)

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

Examples

```r
## create named regions
wb <- wb_workbook()
wb$add_worksheet("Sheet 1")

## specify region
wb$add_data(sheet = 1, x = iris, startCol = 1, startRow = 1)
wb$add_named_region(
    sheet = 1,
    name = "iris",
    rows = seq_len(nrow(iris) + 1),
    cols = seq_along(iris)
)

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

out_file <- temp_xlsx()
wb_save(wb, out_file, overwrite = TRUE)

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

## delete one
wb$remove_named_region(name = "iris2")
get_named_regions(wb)

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

df <- read_xlsx(out_file, namedRegion = "iris2")
head(df)
```

---

numfmt_is_date

check if numFmt is date. internal function

Description
check if numFmt is date. internal function

Usage
numfmt_is_date(numFmt)
Arguments
numFmt  numFmt xml nodes

numfmt_is_posix  check if numFmt is posix. internal function

Description
check if numFmt is posix. internal function

Usage
numfmt_is_posix(numFmt)

Arguments
numFmt  numFmt xml nodes

openxlsx2  xlsx reading, writing and editing.

Description
This R package is a modern reinterpretation of the widely used popular openxlsx package. Similar to its predecessor, it simplifies the creation of xlsx files by providing a clean interface for writing, designing and editing worksheets. Based on a powerful XML library and focusing on modern programming flows in pipes or chains, openxlsx2 allows to break many new ground.

Details
The openxlsx package uses global options to simplify formatting:

- options("openxlsx2.borderColour" = "black")
- options("openxlsx2.borderStyle" = "thin")
- options("openxlsx2.dateFormat" = "mm/dd/yyyy")
- options("openxlsx2.datetimeFormat" = "yyyy-mm-dd hh:mm:ss")
- options("openxlsx2.numFmt" = NULL)
- options("openxlsx2.paperSize" = 9) ## A4
- options("openxlsx2.orientation" = "portrait") ## page orientation
- options("openxlsx2.sheet.default_name" = "Sheet")
- options("openxlsx2.rightToLeft" = NULL)
Authors and contributions:
For a full list of all authors that have made this package possible and for whom we are greatful, please see:

`system.file("AUTHORS", package = "openxlsx2")`

If you feel like you should be included on this list, please let us know. If you have something to contribute, you are welcome. If something is not working as expected, open issues or if you have solved an issue, open a pull request. Please be respectful and be aware that we are volunteers doing this for fun in our unpaid free time. We will work on problems when we have time or need.

License:
This package is licensed under the MIT license and is based on `openxlsx` (by Alexander Walker and Philipp Schauburger; COPYRIGHT 2014-2022) and `pugixml` (by Arseny Kapoulkine; COPYRIGHT 2006-2022). Both released under the MIT license.

See Also

- `vignette(package = "openxlsx2")`
- `write_data()`
- `write_datatable()`
- `write_xlsx()`
- `read_xlsx()`

for examples

---

**print.pugi_xml**

### Description

print pugi_xml

### Usage

```r
## S3 method for class 'pugi_xml'
print(x, indent = " ", raw = FALSE, attr_indent = FALSE, ...)
```

### Arguments

- `x` something to print
- `indent` indent used default is " 
- `raw` print as raw text
- `attr_indent` print attributes indented on new line
- `...` to please check
Examples

# a pointer
x <- read_xml("<a><b/></a>")
print(x)
print(x, raw = TRUE)

Description

returns xml values as character

Usage

xml_node(xml, level1 = NULL, level2 = NULL, level3 = NULL, ...)
xml_node_name(xml, level1 = NULL, level2 = NULL, ...)
xml_value(xml, level1 = NULL, level2 = NULL, level3 = NULL, ...)
xml_attr(xml, level1 = NULL, level2 = NULL, level3 = NULL, ...)

Arguments

xml something xml
level1 to please check
level2 to please check
level3 to please check
... additional arguments passed to read_xml()

Details

This function returns XML nodes as used in openxlsx2. In theory they could be returned as pointers as well, but this has not yet been implemented. If no level is provided, the nodes on level1 are returned.

Examples

x <- read_xml("<a><b/></a>")
# return a
xml_node(x, "a")
# return b. requires the path to the node
xml_node(x, "a", "b")
xml_node_name("<a/>")
xml_node_name("<a><b/></a>", "a")
x <- read_xml("<a>1</a>")
read_sheet_names

Get names of worksheets

Description

Returns the worksheet names within an xlsx file

Usage

read_sheet_names(file)

Arguments

file An xlsx or xslm file.

Value

Character vector of worksheet names.

Examples

read_sheet_names(system.file("extdata", "readTest.xlsx", package = "openxlsx2"))
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,
  startCol = NULL,
  rowNames = FALSE,
  colNames = TRUE,
  skipEmptyRows = FALSE,
  skipEmptyCols = FALSE,
  rows = NULL,
  cols = NULL,
  detectDates = TRUE,
  namedRegion,
  na.strings = "#N/A",
  na.numbers = NA,
  check.names = FALSE,
  sep.names = ".",
  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.
startCol first column to begin looking for data.
rowNames If TRUE, first column of data will be used as row names.
colNames If TRUE, the first row of data will be used as column names.
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.
**read_xlsx**

- **detectDates**: If TRUE, attempt to recognize dates and perform conversion.
- **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.
- **na.numbers**: A numeric vector of digits which are to be interpreted as NA. Blank cells will be returned as NA.
- **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**: (unimplemented) One character which substitutes blanks in column names. By default, "."
- **fillMergedCells**: If TRUE, the value in a merged cell is given to all cells within the merge.
- **...**: additional arguments passed to wb_to_df()

**Details**

Formulae written using write_formula 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

**See Also**

get_named_regions() wb_to_df()

**Examples**

```r
xlsxFile <- system.file("extdata", "readTest.xlsx", package = "openxlsx2")
df1 <- read_xlsx(xlsxFile = xlsxFile, sheet = 1, skipEmptyRows = FALSE)
sapply(df1, class)

df2 <- read_xlsx(xlsxFile = xlsxFile, sheet = 3, skipEmptyRows = TRUE)
df2$Date <- convert_date(df2$Date)
sapply(df2, class)
head(df2)

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

wb <- wb_load(system.file("extdata", "readTest.xlsx", package = "openxlsx2"))
```
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 <- wb_load(system.file("extdata", "readTest.xlsx", package = "openxlsx2"))
df3 <- read_xlsx(wb,
    sheet = 2, skipEmptyRows = FALSE,
    cols = c(1, 4), rows = c(1, 3, 4)
)

## URL
##
head(read_xlsx(xlsxFile))

---

### read_xml

**read xml file**

#### Description

read xml file

#### Usage

```r
read_xml(
  xml,
  pointer = TRUE,
  escapes = FALSE,
  declaration = FALSE,
  whitespace = TRUE,
  empty_tags = FALSE,
  skip_control = TRUE
)
```

#### Arguments

- **xml**: something to read character string or file
- **pointer**: should a pointer be returned?
- **escapes**: bool if characters like "&" should be escaped. The default is no escapes. Assuming that the input already provides valid information.
- **declaration**: should the declaration be imported
- **whitespace**: should whitespace pcdata be imported
- **empty_tags**: should `<b/>` or `<b></b>` be returned
- **skip_control**: should whitespace character be exported
select_active_sheet

Details

Read XML files or strings to pointer and checks if the input is valid XML. If the input is read into a character object, it will be reevaluated every time it is called. A pointer is evaluated once, but lives only for the lifetime of the R session or once it is gc().

Examples

# a pointer
x <- read_xml("<a><b/></a>")
print(x)
print(x, raw = TRUE)
str(x)

# a character
y <- read_xml("<a><b/></a>", pointer = FALSE)
print(y)
print(y, raw = TRUE)
str(y)

# Errors if the import was unsuccessful
try(z <- read_xml("<a><b/></a>"))

xml <- '<?xml test="yay"?><a>A & B</a>'
# difference in escapes
read_xml(xml, escapes = TRUE, pointer = FALSE)
read_xml(xml, escapes = FALSE, pointer = FALSE)
read_xml(xml, escapes = TRUE)
read_xml(xml, escapes = FALSE)

# read declaration
read_xml(xml, declaration = TRUE)

select_active_sheet
get and set table of sheets and their state as selected and active

Description

Multiple sheets can be selected, but only a single one can be active (visible). The visible sheet, must not necessarily be a selected sheet.

Usage

wb_get_active_sheet(wb)

wb_set_active_sheet(wb, sheet)

wb_get_selected(wb)

wb_set_selected(wb, sheet)
Arguments

- **wb**: a workbook
- **sheet**: a sheet name of the workbook

Value

- a data frame with tabSelected and names

Examples

```r
wb <- wb_load(file = system.file("extdata", "loadExample.xlsx", package = "openxlsx2"))
# testing is the selected sheet
wb_get_selected(wb)
# change the selected sheet to IrisSample
wb <- wb_set_selected(wb, "IrisSample")
# get the active sheet
wb_get_active_sheet(wb)
# change the selected sheet to IrisSample
wb <- wb_set_active_sheet(wb, sheet = "IrisSample")
```

---

**set_cell_style**  

**helper set_cell_style**

Description

helper set_cell_style

Usage

`set_cell_style(wb, sheet, cell, value)`

Arguments

- **wb**: a workbook
- **sheet**: a worksheet
- **cell**: a cell
- **value**: a value to assign

Examples

```r
# create a numeric data matrix
mat <- matrix(rnorm(28*28, mean = 44444, sd = 555), ncol = 28)
w <- wb_workbook()$
add_worksheet("test")$
add_data("test", mat, colNames = FALSE)

# create known builtins Φ is the default
```
sheet_visibility <- c(# "0",
"47", "48", "49"
)

# assign the style to the first row
for (builtin in builtins) {
  dim <- paste0(int2col(which(builtins %in% builtin)), "1")
  wb$add_cell_style(dims = dim,
      numFmtId = builtin)
}

# new styles are 1:28, because s in a 0-index, they are in the
# order we just assigned them above
for (i in seq_along(wb$styles_mgr$styles$cellXfs)) {
  cell <- sprintf("%s2:%s28", int2col(i), int2col(i))
  wb$set_cell_style("test", cell, as.character(i))
}

---

**sheet_visibility**  
*Get/set worksheet visible state*

**Description**

Get and set worksheet visible state

**Usage**

```r
wb_get_sheet_visibility(wb)
```

```r
wb_set_sheet_visibility(wb, sheet = current_sheet(), value)
```

**Arguments**

- `wb`  
  A `wbWorkbook` object
- `sheet`  
  Worksheet identifier
- `value`  
  a logical/character vector the same length as sheet

**Value**

Character vector of worksheet names.  
Vector of "hidden", "visible", "veryHidden"
Examples

```r
wb <- wb_workbook()
wbb$add_worksheet(sheet = "S1", visible = FALSE)
wbb$add_worksheet(sheet = "S2", visible = TRUE)
wbb$add_worksheet(sheet = "S3", visible = FALSE)

wb$get_sheet_visibility()
wbb$set_sheet_visibility(1, TRUE)  ## show sheet 1
wb$set_sheet_visibility(2, FALSE)  ## hide sheet 2
wb$set_sheet_visibility(3, "hidden")  ## hide sheet 3
wb$set_sheet_visibility(3, "veryHidden")  ## hide sheet 3 from UI
```

---

**styles_on_sheet**

get all styles on a sheet

**Description**

get all styles on a sheet

**Usage**

```r
styles_on_sheet(wb, sheet)
```

**Arguments**

- `wb` workbook
- `sheet` worksheet

---

**style_is_date**

check if style is date. internal function

**Description**

check if style is date. internal function

**Usage**

```r
style_is_date(cellXfs, numfmt_date)
```

**Arguments**

- `cellXfs` cellXfs xml nodes
- `numfmt_date` custom numFmtId dates
**style_is_posix**  
check if style is posix. internal function

**Description**
check if style is posix. internal function

**Usage**

```
style_is_posix(cellXfs, numfmt_date)
```

**Arguments**

- **cellXfs**: cellXfs xml nodes
- **numfmt_date**: custom numFmtId dates

**style_mgr**  
style manager

**Description**
style manager
style manager

**Public fields**
numfmt numfmt-ids
font font-ids
fill fill-ids
border border-ids
xf xf-ids
dxf dxf-ids
styles styles as xml

**Methods**

**Public methods:**
- `style_mgr$new()`
- `style_mgr$get_numfmt()`
- `style_mgr$get_font()`
- `style_mgr$get_fill()`
- `style_mgr$get_border()`
• style_mgr$get_xf()
• style_mgr$get_dxf()
• style_mgr$get_numfmt_id()
• style_mgr$get_font_id()
• style_mgr$get_fill_id()
• style_mgr$get_border_id()
• style_mgr$get_xf_id()
• style_mgr$get_dxf_id()
• style_mgr$next_numfmt_id()
• style_mgr$next_font_id()
• style_mgr$next_fill_id()
• style_mgr$next_border_id()
• style_mgr$next_xf_id()
• style_mgr$next_dxf_id()
• style_mgr$add()
• style_mgr$clone()

**Method new()**: Creates a new wbStylesMgr object

**Usage**:

```r
style_mgr$new(
  numfmt = NA,
  font = NA,
  fill = NA,
  border = NA,
  xf = NA,
  dxf = NA,
  styles = NA
)
```

**Arguments**:
- `numfmt`: numfmt
- `font`: font
- `fill`: fill
- `border`: border
- `xf`: xf
- `dxf`: dxf
- `styles`: styles

**Returns**: a wbStylesMgr object

**Method get_numfmt()**: get numfmt ids

**Usage**:

```r
style_mgr$get_numfmt()
```

**Method get_font()**: get font ids

**Usage**:
style_mgr

style_mgr$get_font()

**Method** get_fill(): get fill ids

**Usage:**

style_mgr$get_fill()

**Method** get_border(): get border ids

**Usage:**

style_mgr$get_border()

**Method** get_xf(): get xfids

**Usage:**

style_mgr$get_xf()

**Method** get_dxf(): get dxf ids

**Usage:**

style_mgr$get_dxf()

**Method** get_numfmt_id(): get numfmt id by name

**Usage:**

style_mgr$get_numfmt_id(name)

**Arguments:**

name name

**Method** get_font_id(): get font id by name

**Usage:**

style_mgr$get_font_id(name)

**Arguments:**

name name

**Method** get_fill_id(): get fill id by name

**Usage:**

style_mgr$get_fill_id(name)

**Arguments:**

name name

**Method** get_border_id(): get border id by name

**Usage:**

style_mgr$get_border_id(name)

**Arguments:**

name name

**Method** get_xf_id(): get xf id by name

**Usage:**
**Method** `get_dxf_id()`: get dxf id by name

*Usage:*

```
style_mgr$get_dxf_id(name)
```

*Arguments:*

- `name`: name

**Method** `next_numfmt_id()`: get next numfmt id

*Usage:*

```
style_mgr$next_numfmt_id()
```

**Method** `next_font_id()`: get next font id

*Usage:*

```
style_mgr$next_font_id()
```

**Method** `next_fill_id()`: get next fill id

*Usage:*

```
style_mgr$next_fill_id()
```

**Method** `next_border_id()`: get next border id

*Usage:*

```
style_mgr$next_border_id()
```

**Method** `next_xf_id()`: get next xf id

*Usage:*

```
style_mgr$next_xf_id()
```

**Method** `next_dxf_id()`: get next xf id

*Usage:*

```
style_mgr$next_dxf_id()
```

**Method** `add()`: add entry

*Usage:*

```
style_mgr$add(style, style_name, skip_duplicates = TRUE)
```

*Arguments:*

- `style`: new_style
- `style_name`: a unique name identifying the style
- `skip_duplicates`: should duplicates be added?

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
style_mgr$clone(deep = FALSE)
```

*Arguments:*

- `deep`: Whether to make a deep clone.
Examples

```r
xlsxFile <- system.file("extdata", "oxlsx2_sheet.xlsx", package = "openxlsx2")
w <- wb_load(xlsxFile)

# ## start style mgr
# style <- style_mgr$new(wb)
# style$initialize(wb)

# wb$styles_mgr$get_numfmt() |> print()
# wb$styles_mgr$next_numfmt_id() |> print()
# wb$styles_mgr$get_numfmt_id("numFmt-166")

# create new number format
new_numfmt <- create_numfmt(numFmtId = wb$styles_mgr$next_numfmt_id(), formatCode = "#,#")

# add it via stylemgr
wb$styles_mgr$add(new_numfmt, "test")

## get numfmts (invisible)
# z <- wb$styles_mgr$get_numfmt()
# z
wb$styles_mgr$styles$numFmts

## create and add huge font
new_huge_font <- create_font(sz = "20", name = "Arial", b = "1",
                           color = wb_colour(hex = "FFFFFFF"))
w$styles_mgr$add(new_huge_font, "arial_huge")

## create another font
new_font <- create_font(name = "Arial")
w$styles_mgr$add(new_font, "arial")

## create and add new fill
new_fill <- create_fill(patternType = "solid", fgColor = wb_colour(hex = "FF00224B"))
w$styles_mgr$add(new_fill, "blue")

# create new style with numfmt enabled
head_xf <- create_cell_style(
    horizontal = "center",
    textRotation = "45",
    numFmtId = "0",
    fontId = wb$styles_mgr$get_font_id("arial_huge"),
    fillId = wb$styles_mgr$get_fill_id("blue")
)

new_xf <- create_cell_style(
    numFmtId = wb$styles_mgr$get_numfmt_id("test"),
    fontId = wb$styles_mgr$get_font_id("arial")
)

## add new styles
wb$styles_mgr$add(head_xf, "head_xf")
```
wb$styles_mgr$add(new_xf, "new_xf")

## get cell style ids (invisible)
# z <- wb$styles_mgr$get_xf()

## get cell style id
# wb$styles_mgr$get_xf_id("new_xf")

## assign styles to cells
wb$set_cell_style("SUM", "B3:I3", wb$styles_mgr$get_xf_id("head_xf"))
wb$set_cell_style("SUM", "C7:C16", wb$styles_mgr$get_xf_id("new_xf"))
# wb_open(wb)

temp_xlsx             helper function to create temporary directory for testing purpose

Description
helper function to create temporary directory for testing purpose

Usage
temp_xlsx(name = "temp_xlsx", macros = FALSE)

Arguments
  name       for the temp file
  macros     logical if the file extension is xlsm or xlsx

wbChartSheet         R6 class for a Workbook Chart Sheet

Description
R6 class for a Workbook Chart Sheet
R6 class for a Workbook Chart Sheet

Details
A chart sheet

Value
A character vector of xml
Public fields

- sheetPr  Sheet something?
- sheetViews  Something
- pageMargins  page margins
- drawing  drawing
- hyperlinks  hyperlinks

Methods

Public methods:

- `wbChartSheet$new()`
- `wbChartSheet$get_prior_sheet_data()`
- `wbChartSheet$set_sheetview()`
- `wbChartSheet$clone()`

Method `new()`: Create a new workbook chart sheet object

Usage:

`wbChartSheet$new(tabColour = tabColour)`

Arguments:

- `tabColour` character a tab colour to set

Returns: The `wbChartSheet` object

Method `get_prior_sheet_data()`: get (prior) sheet data

Usage:

`wbChartSheet$get_prior_sheet_data()`

Method `set_sheetview()`: add sheetview

Usage:

`wbChartSheet$set_sheetview(
  colorId = NULL,
  defaultGridColor = NULL,
  rightToLeft = NULL,
  showFormulas = NULL,
  showGridLines = NULL,
  showOutlineSymbols = NULL,
  showRowColHeaders = NULL,
  showRuler = NULL,
  showWhiteSpace = NULL,
  showZeros = NULL,
  tabSelected = NULL,
  topLeftCell = NULL,
  view = NULL,
  workbookViewId = NULL,
)`,
zoomScale = NULL,
zoomScaleNormal = NULL,
zoomScalePageLayoutView = NULL,
zoomScaleSheetLayoutView = NULL
)

Arguments:
colorId colorId
defaultGridColor defaultGridColor
rightToLeft rightToLeft
showFormulas showFormulas
showGridLines showGridLines
showOutlineSymbols showOutlineSymbols
showRowColHeaders showRowColHeaders
showRuler showRuler
showWhiteSpace showWhiteSpace
showZeros showZeros
tabSelected tabSelected
topLeftCell topLeftCell
view view
windowProtection windowProtection
workbookViewId workbookViewId
zoomScale zoomScale
zoomScaleNormal zoomScaleNormal
zoomScalePageLayoutView zoomScalePageLayoutView
zoomScaleSheetLayoutView zoomScaleSheetLayoutView

Returns: The \texttt{wbWorksheetObject}, invisibly

**Method** \texttt{clone}(): The objects of this class are cloneable with this method.

*Usage:*

ewbChartSheet$\texttt{clone}(deep = \texttt{FALSE})

*Arguments:*

deep Whether to make a deep clone.
Details

A comment

Value

The \texttt{wbComment} object, invisibly; called for its side effects

Public fields

- \texttt{text}  Comment text
- \texttt{author}  The comment author
- \texttt{style}  A style for the comment
- \texttt{visible}  logical, if \texttt{FALSE} is not visible
- \texttt{width}  Width of the comment in ... units
- \texttt{height}  Height of comment in ... units

Methods

Public methods:

- \texttt{wbComment$new()}  
- \texttt{wbComment$print()}  
- \texttt{wbComment$clone()}

Method \texttt{new()}: Creates a new \texttt{wbComment} object

\texttt{Usage:}
\texttt{wbComment$new(text, author, style, visible = TRUE, width = 2, height = 4)}

\texttt{Arguments:}
- \texttt{text}  Comment text
- \texttt{author}  The comment author
- \texttt{style}  A style for the comment
- \texttt{visible}  logical, if \texttt{FALSE} is not visible
- \texttt{width}  Width of the comment in ... units
- \texttt{height}  Height of comment in ... units

\texttt{Returns:}  a \texttt{wbComment} object

Method \texttt{print()}: Prints the object

\texttt{Usage:}
\texttt{wbComment$print()}

Method \texttt{clone()}: The objects of this class are cloneable with this method.

\texttt{Usage:}
\texttt{wbComment$clone(deep = FALSE)}

\texttt{Arguments:}
- \texttt{deep}  Whether to make a deep clone.
R6 class for a Workbook Hyperlink

Description
R6 class for a Workbook Hyperlink

Details
A hyperlink

Value
A character vector of html if is_external is TRUE, otherwise NULL

Public fields
ref ref
target target
location location
display display
is_external is_external

Methods
Public methods:
• wbHyperlink$new()
• wbHyperlink$to_xml()
• wbHyperlink$to_target_xml()
• wbHyperlink$clone()

Method new(): Creates a new wbHyperlink object

Usage:
wbHyperlink$new(ref, target, location, display = NULL, is_external = TRUE)

Arguments:
ref ref
target target
location location
display display
is_external is_external

Returns: a wbHyperlink object
**Method** `to_xml()`: Convert to xml

*Usage:*

```r
wbHyperlink$to_xml(id)
```

*Arguments:*

- `id` ???

*Returns:* A character vector of xml

**Method** `to_target_xml()`: Convert to target xml

*Usage:*

```r
wbHyperlink$to_target_xml(id)
```

*Arguments:*

- `id` ???

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```r
wbHyperlink$clone(deep = FALSE)
```

*Arguments:*

- `deep` Whether to make a deep clone.

---

### Description

R6 class for a Workbook Hyperlink

R6 class for a Workbook Hyperlink

### Usage

```r
wb_sheet_data()
```

### Details

A hyperlink

### Public fields

- `row_attr`  
- `cc`  
- `cc_out`
Methods

Public methods:

• `wbSheetData$new()`
• `wbSheetData$clone()`

Method `new()`: Creates a new `wbSheetData` object

Usage:

`wbSheetData$new()`

Returns: a `wbSheetData` object

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`wbSheetData$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

---

R6 class for a Workbook

Description

R6 class for a Workbook

R6 class for a Workbook

Details

A Workbook

minor helper wrapping `xl_open` which does the entire same thing

Value

The integer position of the sheet

The `wbWorkbook` object

The `wbWorkbook` object

The `wbWorkbook` object

A named character vector of sheet names in their order. The names represent the original value of
the worksheet prior to any character substitutions.

The `wbWorkbook` object

The `wbWorkbook` object

The `wbWorkbook` object

The `wbWorkbook` object
The `wbWorkbook` object

- The `wbWorkbook` object
- The `wbWorkbook` object
- The `wbWorkbook` object
- The `wbWorkbook` object
- The `wbWorkbook` object
- The `wbWorkbook` object
- The `wbWorkbook` object
- Returns sheet visibility

`a character vector of cell styles`

**Public fields**

- `sheet_names`
- `calcChain`
- `apps`
- `charts`
- `is_chartsheet`
- `customXml`
- `connections`
- `ctrlProps`
- `Content_Types`
- `app`
- `core`
- `custom`
- `drawings`
- `drawings_rels`
- `embeddings`
- `externalLinks`
- `externalLinksRels`
- `headFoot`
media media
metadata metadata
persons persons
pivotTables pivotTables
pivotTables.xml.rels pivotTables.xml.rels
pivotDefinitions pivotDefinitions
pivotRecords pivotRecords
pivotDefinitionsRels pivotDefinitionsRels
queryTables queryTables
slicers slicers
slicerCaches slicerCaches
sharedStrings sharedStrings
styles_mgr styles_mgr
styles_xml styles_xml
tables tables
tables.xml.rels tables.xml.rels
theme theme
vbaProject vbaProject
vml vml
vml_rels vml_rels
comments comments
threadComments threadComments
workbook workbook
workbook.xml.rels workbook.xml.rels
worksheets worksheets
worksheets_rels worksheets_rels
sheetOrder The sheet order. Controls ordering for worksheets and worksheet names.
path path
creator A character vector of creators
title title
subject subject
category category
datetimeCreated The datetime (as POSIXt) the workbook is created. Defaults to the current 
Sys.time() when the workbook object is created, not when the Excel files are saved.
Methods

Public methods:

• `wbWorkbook$new()`
• `wbWorkbook$append()`
• `wbWorkbook$append_sheets()`
• `wbWorkbook$validate_sheet()`
• `wbWorkbook$add_chartsheet()`
• `wbWorkbook$add_worksheet()`
• `wbWorkbook$clone_worksheet()`
• `wbWorkbook$add_data()`
• `wbWorkbook$add_data_table()`
• `wbWorkbook$add_formula()`
• `wbWorkbook$add_style()`
• `wbWorkbook$save()`
• `wbWorkbook$open()`
• `wbWorkbook$buildTable()`
• `wbWorkbook$get_base_font()`
• `wbWorkbook$set_base_font()`
• `wbWorkbook$set_bookview()`
• `wbWorkbook$get_sheet_names()`
• `wbWorkbook$set_sheet_names()`
• `wbWorkbook$setSheetName()`
• `wbWorkbook$set_row_heights()`
• `wbWorkbook$remove_row_heights()`
• `wbWorkbook$createCols()`
• `wbWorkbook$group_cols()`
• `wbWorkbook$ungroup_cols()`
• `wbWorkbook$remove_col_widths()`
• `wbWorkbook$set_col_widths()`
• `wbWorkbook$group_rows()`
• `wbWorkbook$ungroup_rows()`
• `wbWorkbook$remove_worksheet()`
• `wbWorkbook$add_data_validation()`
• `wbWorkbook$merge_cells()`
• `wbWorkbook$unmerge_cells()`
• `wbWorkbook$freeze_pane()`
• `wbWorkbook$add_comment()`
• `wbWorkbook$remove_comment()`
• `wbWorkbook$add_conditional_formatting()`
• `wbWorkbook$add_image()`
• `wbWorkbook$add_plot()`
• `wbWorkbook$add_drawing()`
• `wbWorkbook$add_chart_xml()`
• `wbWorkbook$add_mschart()`
• `wbWorkbook$print()`
• `wbWorkbook$protect()`
• `wbWorkbook$protect_worksheet()`
• `wbWorkbook$set_creators()`
• `wbWorkbook$add_creators()`
• `wbWorkbook$remove_creators()`
• `wbWorkbook$set_last_modified_by()`
• `wbWorkbook$page_setup()`
• `wbWorkbook$set_header_footer()`
• `wbWorkbook$get_tables()`
• `wbWorkbook$remove_tables()`
• `wbWorkbook$add_filter()`
• `wbWorkbook$remove_filter()`
• `wbWorkbook$grid_lines()`
• `wbWorkbook$add_named_region()`
• `wbWorkbook$remove_named_region()`
• `wbWorkbook$set_order()`
• `wbWorkbook$set_sheet_visibility()`
• `wbWorkbook$get_sheet_visibility()`
• `wbWorkbook$add_page_break()`
• `wbWorkbook$clean_sheet()`
• `wbWorkbook$add_border()`
• `wbWorkbook$add_fill()`
• `wbWorkbook$add_font()`
• `wbWorkbook$add_numfmt()`
• `wbWorkbook$add_cell_style()`
• `wbWorkbook$get_cell_style()`
• `wbWorkbook$set_cell_style()`
• `wbWorkbook$clone_sheet_style()`
• `wbWorkbook$add_sparklines()`
• `wbWorkbook$clone()`

**Method** `new()`: Creates a new `wbWorkbook` object

**Usage:**

```r
wbWorkbook$new(
  creator = NULL,
  title = NULL,
  subject = NULL,
  category = NULL,
  datetimeCreated = Sys.time()
)
```
Arguments:
creator character vector of creators. Duplicated are ignored.
title title
subject subject
category category
datet imeCreated The datetime (as POSIXt) the workbook is created. Defaults to the current Sys.time() when the workbook object is created, not when the Excel files are saved.

Returns: a wbWorkbook object

Method append(): Append a field. This method is intended for internal use

Usage:
wbWorkbook$append(field, value)

Arguments:
field A valid field name
value A value for the field

Method append_sheets(): Append to self$workbook$sheets This method is intended for internal use

Usage:
wbWorkbook$append_sheets(value)

Arguments:
value A value for self$workbook$sheets

Method validate_sheet(): validate sheet

Usage:
wbWorkbook$validate_sheet(sheet)

Arguments:
sheet A character sheet name or integer location

Method add_chartsheet(): Add a chart sheet to the workbook

Usage:
wbWorkbook$add_chartsheet(
  sheet = next_sheet(),
  tabColour = NULL,
  zoom = 100,
  visible = c("true", "false", "hidden", "visible", "veryhidden")
)

Arguments:
sheet sheet
tabColour tabColour
zoom zoom
visible visible

Returns: The wbWorkbook object, invisibly
Method `add_worksheet()`: Add worksheet to the `wbWorkbook` object

Usage:

```r
wbWorkbook$add_worksheet(
  sheet = next_sheet(),
  gridLines = TRUE,
  rowColHeaders = TRUE,
  tabColour = NULL,
  zoom = 100,
  header = NULL,
  footer = NULL,
  oddHeader = header,
  oddFooter = footer,
  evenHeader = header,
  evenFooter = footer,
  firstHeader = header,
  firstFooter = footer,
  visible = c("true", "false", "hidden", "visible", "veryhidden"),
  hasDrawing = FALSE,
  paperSize = getOption("openxlsx2.paperSize", default = 9),
  orientation = getOption("openxlsx2.orientation", default = "portrait"),
  hdpi = getOption("openxlsx2.hdpi", default = getOption("openxlsx2.dpi", default = 300)),
  vdpi = getOption("openxlsx2.vdpi", default = getOption("openxlsx2.dpi", default = 300))
)
```

Arguments:
- `sheet`
- `gridLines`
- `rowColHeaders`
- `tabColour`
- `zoom`
- `header`
- `footer`
- `oddHeader`
- `oddFooter`
- `evenHeader`
- `evenFooter`
- `firstHeader`
- `firstFooter`
- `visible`
- `hasDrawing`
- `paperSize`
- `orientation`
- `hdpi`
- `vdpi`

Returns: The `wbWorkbook` object, invisibly
Method clone_worksheet(): Clone a workbook sheet

Usage:
wbWorkbook$clone_worksheet(old = current_sheet(), new = next_sheet())

Arguments:
old  name of worksheet to clone
new  name of new worksheet to add

Method add_data(): add data

Usage:
wbWorkbook$add_data(
  sheet = current_sheet(),
  x,
  startCol = 1,
  startRow = 1,
  dims = rowcol_to_dims(startRow, startCol),
  array = FALSE,
  xy = NULL,
  colNames = TRUE,
  rowNames = FALSE,
  withFilter = FALSE,
  name = NULL,
  sep =", ",
  applyCellStyle = TRUE,
  removeCellStyle = FALSE,
  na.strings
)

Arguments:
sheet  sheet
x  x
startCol  startCol
startRow  startRow
dims  dims
array  array
xy  xy
colNames  colNames
rowNames  rowNames
withFilter  withFilter
name  name
sep  sep
applyCellStyle  applyCellStyle
removeCellStyle  if writing into existing cells, should the cell style be removed?
na.strings  na.strings
return  The wbWorkbook object

Method add_data_table(): add a data table
Usage:
wbWorkbook$add_data_table(
    sheet = current_sheet(),
    x,
    startCol = 1,
    startRow = 1,
    dims = rowcol_to_dims(startRow, startCol),
    xy = NULL,
    colNames = TRUE,
    rowNames = FALSE,
    tableStyle = "TableStyleLight9",
    tableName = NULL,
    withFilter = TRUE,
    sep = ", ",
    firstColumn = FALSE,
    lastColumn = FALSE,
    bandedRows = TRUE,
    bandedCols = FALSE,
    applyCellStyle = TRUE,
    removeCellStyle = FALSE,
    na.strings
)

Arguments:
sheet x
startCol startCol
startRow startRow
dims xy
colNames colNames
rowNames rowNames
tableStyle tableStyle
tableName tableName
withFilter withFilter
sep firstColumn
lastColumn bandedRows
bandedCols bandedCols
applyCellStyle removeCellStyle
na.strings

Method add_formula(): add formula

Usage:
wbWorkbook$add_formula(
    sheet = current_sheet(),
    x,
    startCol = 1,
    startRow = 1,
    dims = rowcol_to_dims(startRow, startCol),
    array = FALSE,
    xy = NULL,
    applyCellStyle = TRUE,
    removeCellStyle = FALSE
)

Arguments:
  sheet  sheet
  x      x
  startCol  startCol
  startRow  startRow
  dims  dims
  array  array
  xy  xy
  applyCellStyle  applyCellStyle
  removeCellStyle  if writing into existing cells, should the cell style be removed?

Method add_style(): add style

Usage:
wbWorkbook$add_style(style = NULL, style_name = NULL)

Arguments:
  style  style
  style_name  style_name

Method save(): Save the workbook

Usage:
wbWorkbook$save(path = self$path, overwrite = TRUE)

Arguments:
  path  The path to save the workbook to
  overwrite  If FALSE, will not overwrite when path exists

Returns:  The wbWorkbook object invisibly

Method open(): open wbWorkbook in Excel.

Usage:
wbWorkbook$open(interactive = NA)

Arguments:
  interactive  If FALSE will throw a warning and not open the path. This can be manually set to TRUE, otherwise when NA (default) uses the value returned from base::interactive()
Returns: The wbWorkbook, invisibly

Method buildTable(): Build table

Usage:
wbWorkbook$buildTable(
    sheet = current_sheet(),
    colNames,
    ref,
    showColNames,
    tableStyle,
    tableName,
    withFilter,
    totalsRowCount = 0,
    showFirstColumn = 0,
    showLastColumn = 0,
    showRowStripes = 1,
    showColumnStripes = 0
)

Arguments:
    sheet sheet
    colNames colNames
    ref ref
    showColNames showColNames
    tableStyle tableStyle
    tableName tableName
    withFilter withFilter
    totalsRowCount totalsRowCount
    showFirstColumn showFirstColumn
    showLastColumn showLastColumn
    showRowStripes showRowStripes
    showColumnStripes showColumnStripes

Returns: The wbWorksheet object, invisibly

Method get_base_font(): Get the base font

Usage:
wbWorkbook$get_base_font()

Returns: A list of of the font

Method set_base_font(): Get the base font

Usage:
wbWorkbook$set_base_font(
    fontSize = 11,
    fontColour = wb_colour(theme = "1"),
    fontName = "Calibri"
)
Arguments:
fontSize fontSize
fontColour fontColour
fontName fontName

Returns: The wbWorkbook object

Method set_bookview(): Set the book views

Usage:
wbWorkbook$set_bookview(
  activeTab = NULL,
  autoFilterDateGrouping = NULL,
  firstSheet = NULL,
  minimized = NULL,
  showHorizontalScroll = NULL,
  showSheetTabs = NULL,
  showVerticalScroll = NULL,
  tabRatio = NULL,
  visibility = NULL,
  windowHeight = NULL,
  windowWidth = NULL,
  xWindow = NULL,
  yWindow = NULL
)

Arguments:
activeTab activeTab
autoFilterDateGrouping autoFilterDateGrouping
firstSheet firstSheet
minimized minimized
showHorizontalScroll showHorizontalScroll
showSheetTabs showSheetTabs
showVerticalScroll showVerticalScroll
tabRatio tabRatio
visibility visibility
windowHeight windowHeight
windowWidth windowWidth
xWindow xWindow
yWindow yWindow

Returns: The wbWorkbook object

Method get_sheet_names(): Get sheet names

Usage:
wbWorkbook$get_sheet_names()

Method set_sheet_names(): Sets a sheet name
**Usage:**

wbWorkbook$set_sheet_names(old = NULL, new)

**Arguments:**

- **old** Old sheet name
- **new** New sheet name

**Returns:** The wbWorkbook object, invisibly

**Method** setSheetName(): Deprecated. Use set_sheet_names() instead

**Usage:**

wbWorkbook$setSheetName(sheet = current_sheet(), name)

**Arguments:**

- **sheet** Old sheet name
- **name** New sheet name

**Returns:** The wbWorkbook object, invisibly

**Method** set_row_heights(): Sets a row height for a sheet

**Usage:**

wbWorkbook$set_row_heights(
  sheet = current_sheet(),
  rows,
  heights = NULL,
  hidden = FALSE
)

**Arguments:**

- **sheet** sheet
- **rows** rows
- **heights** heights
- **hidden** hidden

**Returns:** The wbWorkbook object, invisibly

**Method** remove_row_heights(): Sets a row height for a sheet

**Usage:**

wbWorkbook$remove_row_heights(sheet = current_sheet(), rows)

**Arguments:**

- **sheet** sheet
- **rows** rows

**Returns:** The wbWorkbook object, invisibly

**Method** createCols():

**Usage:**

wbWorkbook$createCols(sheet = current_sheet(), n, beg, end)

**Arguments:**
Method group_cols(): Group cols

Usage:
wbWorkbook$group_cols(
  sheet = current_sheet(),
  cols,
  collapsed = FALSE,
  levels = NULL
)

Arguments:
sheet sheet
cols cols
collapsed collapsed
levels levels

Returns: The wbWorkbook object, invisibly

Method ungroup_cols(): ungroup cols

Usage:
wbWorkbook$ungroup_cols(sheet = current_sheet(), cols)

Arguments:
sheet sheet
cols = cols

Method remove_col_widths(): Remove row heights from a worksheet

Usage:
wbWorkbook$remove_col_widths(sheet = current_sheet(), cols)

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

Returns: The wbWorkbook object, invisibly

Method set_col_widths(): Group cols

Usage:
wbWorkbook$set_col_widths(
  sheet = current_sheet(),
  cols,
  widths = 8.43,
  hidden = FALSE
)
Arguments:
sheet  sheet
cols  cols
widths  Width of columns
hidden  A logical vector to determine which cols are hidden; values are repeated across length
        of cols

Returns: The `wbWorkbook` object, invisibly

Method `group_rows()`: Group rows

Usage:
```
wbWorkbook$group_rows(
  sheet = current_sheet(),
  rows,
  collapsed = FALSE,
  levels = NULL
)
```

Arguments:
sheet  sheet
rows  rows
collapsed  collapsed
levels  levels

Returns: The `wbWorkbook` object, invisibly

Method `ungroup_rows()`: ungroup rows

Usage:
```
wbWorkbook$ungroup_rows(sheet = current_sheet(), rows)
```

Arguments:
sheet  sheet
rows  rows

Returns: The `wbWorkbook` object

Method `remove_worksheet()`: Remove a worksheet

Usage:
```
wbWorkbook$remove_worksheet(sheet = current_sheet())
```

Arguments:
sheet  The worksheet to delete

Returns: The `wbWorkbook` object, invisibly

Method `add_data_validation()`: Adds data validation

Usage:
wbWorkbook$add_data_validation(
  sheet = current_sheet(), 
  cols, 
  rows, 
  type, 
  operator, 
  value, 
  allowBlank = TRUE, 
  showInputMsg = TRUE, 
  showErrorMsg = TRUE, 
  errorStyle = NULL, 
  errorTitle = NULL, 
  error = NULL, 
  promptTitle = NULL, 
  prompt = NULL
)

Arguments:
  sheet sheet
cols cols
rows rows
type type
operator operator
value value
allowBlank allowBlank
showInputMsg showInputMsg
showErrorMsg showErrorMsg
errorStyle The icon shown and the options how to deal with such inputs. Default "stop" (cancel), else "information" (prompt popup) or "warning" (prompt accept or change input)
errorTitle The error title
error The error text
promptTitle The prompt title
prompt The prompt text

Method merge_cells(): Set cell merging for a sheet

Usage:
wbWorkbook$merge_cells(sheet = current_sheet(), rows = NULL, cols = NULL)

Arguments:
  sheet sheet
  rows, cols Row and column specifications.

Returns: The wbWorkbook object, invisibly

Method unmerge_cells(): Removes cell merging for a sheet

Usage:
wbWorkbook$unmerge_cells(sheet = current_sheet(), rows = NULL, cols = NULL)
Arguments:
sheet sheet
rows, cols Row and column specifications.
Returns: The \texttt{wbWorkbook} object, invisibly

Method \texttt{freeze\_pane()}: Set freeze panes for a sheet

Usage:
\begin{verbatim}
wbWorkbook$freeze\_pane(
    sheet = current\_sheet(),
    firstActiveRow = NULL,
    firstActiveCol = NULL,
    firstRow = FALSE,
    firstCol = FALSE
)
\end{verbatim}
Arguments:
sheet sheet
firstActiveRow firstActiveRow
firstActiveCol firstActiveCol
firstRow firstRow
firstCol firstCol

Returns: The \texttt{wbWorkbook} object, invisibly

Method \texttt{add\_comment()}: Add comment

Usage:
\begin{verbatim}
wbWorkbook$add\_comment(
    sheet = current\_sheet(),
    col, row,
    dims = rowcol\_to\_dims(row, col),
    comment
)
\end{verbatim}
Arguments:
sheet sheet
col column to apply the comment
row row to apply the comment
dims row and column as spreadsheet dimension, e.g. "A1"
comment a comment to apply to the worksheet

Method \texttt{remove\_comment()}: Remove comment

Usage:
\begin{verbatim}
wbWorkbook$remove\_comment(
    sheet = current\_sheet(),
    col, row,
\end{verbatim}
wbWorkbook$add_conditional_formatting(
  sheet = current_sheet(),
  cols, rows, rule = NULL, style = NULL, type = c("expression", "colorScale", "dataBar", "duplicatedValues", "containsText", "notContainsText", "beginsWith", "endsWith", "between", "topN", "bottomN"),
  params = list(showValue = TRUE, gradient = TRUE, border = TRUE, percent = FALSE, rank = 5L)
)

Arguments:
sheet sheet
cols cols
rows rows
rule rule
style style
type type
params Additional parameters

Method add_image(): Insert an image into a sheet

Usage:
wbWorkbook$add_image(
  sheet = current_sheet(),
  file, width = 6, height = 3, startRow = 1, startCol = 1, rowOffset = 0, colOffset = 0, units = "in", dpi = 300
)
Arguments:
sheet sheet
file file
width width
height height
startRow startRow
startCol startCol
rowOffset rowOffset
colOffset colOffset
units units
dpi dpi

Returns: The wbWorkbook object, invisibly

Method add_plot(): Add plot. A wrapper for add_image()

Usage:
wbWorkbook$add_plot(
  sheet = current_sheet(),
  width = 6,
  height = 4,
  xy = NULL,
  startRow = 1,
  startCol = 1,
  rowOffset = 0,
  colOffset = 0,
  fileType = "png",
  units = "in",
  dpi = 300
)

Arguments:
sheet sheet
width width
height height
xy xy
startRow startRow
startCol startCol
rowOffset rowOffset
colOffset colOffset
fileType fileType
units units
dpi dpi

Method add_drawing(): Add xml drawing

Usage:
wbWorkbook$add_drawing(sheet = current_sheet(), xml, dims = NULL)
**Arguments:**
sheet sheet
xml xml
dims dims

**Method add_chart_xml():** Add xml drawing
Add xml chart

**Usage:**
wbWorkbook$add_chart_xml(sheet = current_sheet(), xml, dims = NULL)

**Arguments:**
sheet sheet
xml xml
dims dims

**Method add_mschart():** Add mschart chart to the workbook

**Usage:**
wbWorkbook$add_mschart(sheet = current_sheet(), dims = NULL, graph)

**Arguments:**
sheet the sheet on which the graph will appear
dims the dimensions where the sheet will appear
graph mschart graph

**Method print():** Prints the wbWorkbook object

**Usage:**
wbWorkbook$print()

**Returns:** The wbWorkbook object, invisibly; called for its side-effects

**Method protect():** Protect a workbook

**Usage:**
wbWorkbook$protect(
  protect = TRUE,
  password = NULL,
  lockStructure = FALSE,
  lockWindows = FALSE,
  type = c("1", "2", "4", "8"),
  fileSharing = FALSE,
  username = unname(Sys.info()["user"]),
  readOnlyRecommended = FALSE
)

**Arguments:**
protect protect
password password
lockStructure lockStructure
Returns: The \texttt{wbWorkbook} object, invisibly

\textbf{Method} \texttt{protect_worksheet}(): protect worksheet

\textit{Usage:}
\begin{verbatim}
\texttt{wbWorkbook$protect\_worksheet(
    sheet = current\_sheet(),
    protect = TRUE,
    password = NULL,
    properties = NULL
)
}
\end{verbatim}

\textit{Arguments:}
\begin{itemize}
  \item \texttt{sheet} sheet
  \item \texttt{protect} protect
  \item \texttt{password} password
  \item \texttt{properties} A character vector of properties to lock. Can be one or more of the following:
    - "selectLockedCells",
    - "selectUnlockedCells",
    - "formatCells",
    - "formatColumns",
    - "formatRows",
    - "insertColumns",
    - "insertRows",
    - "insertHyperlinks",
    - "deleteColumns",
    - "deleteRows",
    - "sort",
    - "autoFilter",
    - "pivotTables",
    - "objects",
    - "scenarios"
\end{itemize}

\textbf{Method} \texttt{set\_creators}(): Set creator(s)

\textit{Usage:}
\begin{verbatim}
\texttt{wbWorkbook$set\_creators(creators)
}
\end{verbatim}

\textit{Arguments:}
\begin{itemize}
  \item \texttt{creators} A character vector of creators to set. Duplicates are ignored.
\end{itemize}

\textbf{Method} \texttt{add\_creators}(): Add creator(s)

\textit{Usage:}
\begin{verbatim}
\texttt{wbWorkbook$add\_creators(creators)
}
\end{verbatim}

\textit{Arguments:}
\begin{itemize}
  \item \texttt{creators} A character vector of creators to add. Duplicates are ignored.
\end{itemize}

\textbf{Method} \texttt{remove\_creators}(): Remove creator(s)

\textit{Usage:}
\begin{verbatim}
\texttt{wbWorkbook$remove\_creators(creators)
}
\end{verbatim}

\textit{Arguments:}
\begin{itemize}
  \item \texttt{creators} A character vector of creators to remove. All duplicated are removed.
\end{itemize}

\textbf{Method} \texttt{set\_last\_modified\_by}(): Change the last modified by
Usage:
wbWorkbook$set_last_modified_by(LastModifiedBy = NULL)

Arguments:
LastModifiedBy  A new value

Returns:  The wbWorkbook object, invisibly

Method page_setup(): page_setup()

Usage:
wWorkbook$Page_setup(
  sheet = current_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:
sheet sheet orientation orientation
scale scale scale scale
left left right right
right right top top
bottom bottom bottom bottom
header header header header
footer footer footer footer
fitToWidth fitToWidth fitToWidth fitToHeight fitToHeight
paperSize paperSize paperSize paperSize
printTitleRows printTitleRows printTitleRows printTitleRows
printTitleCols printTitleCols printTitleCols printTitleCols
summaryRow summaryRow summaryRow summaryRow
summaryCol summaryCol summaryCol summaryCol

Returns:  The wbWorkbook object, invisibly
Method `set_header_footer()`: Sets headers and footers

*Usage:*

```r
wbWorkbook$set_header_footer(
  sheet = current_sheet(),
  header = NULL,
  footer = NULL,
  evenHeader = NULL,
  evenFooter = NULL,
  firstHeader = NULL,
  firstFooter = NULL
)
```

*Arguments:*

- `sheet` sheet
- `header` header
- `footer` footer
- `evenHeader` evenHeader
- `evenFooter` evenFooter
- `firstHeader` firstHeader
- `firstFooter` firstFooter

*Returns:* The `wbWorkbook` object, invisibly

Method `get_tables()`: get tables

*Usage:*

```r
wbWorkbook$get_tables(sheet = current_sheet())
```

*Arguments:*

- `sheet` sheet

Method `remove_tables()`: remove tables

*Usage:*

```r
wbWorkbook$remove_tables(sheet = current_sheet(), table)
```

*Arguments:*

- `sheet` sheet
- `table` table

Method `add_filter()`: add filters

*Usage:*

```r
wbWorkbook$add_filter(sheet = current_sheet(), rows, cols)
```

*Arguments:*

- `sheet` sheet
- `rows` rows
- `cols` cols

Method `remove_filter()`: remove filters
Usage:
wbWorkbook$remove_filter(sheet = current_sheet())

Arguments:
sheet sheet

Method grid_lines(): grid lines

Usage:
wbWorkbook$grid_lines(sheet = current_sheet(), show = FALSE, print = show)

Arguments:
sheet sheet
show show
print print

Method add_named_region(): add a named region

Usage:
wbWorkbook$add_named_region(
  sheet = current_sheet(),
  cols,
  rows,
  name,
  localSheet = FALSE,
  overwrite = FALSE,
  comment = NULL,
  customMenu = NULL,
  description = NULL,
  is_function = NULL,
  functionGroupId = NULL,
  help = NULL,
  hidden = NULL,
  localName = NULL,
  publishToServer = NULL,
  statusBar = NULL,
  vbProcedure = NULL,
  workbookParameter = NULL,
  xml = NULL
)

Arguments:
sheet sheet
cols cols
rows rows
name name
localSheet localSheet
overwrite overwrite
comment comment
customMenu customMenu
Method remove_named_region(): remove a named region  
Usage:  
wbWorkbook$remove_named_region(sheet = current_sheet(), name = NULL)  
Arguments:  
sheet sheet  
name name

Method set_order(): set worksheet order  
Usage:  
wbWorkbook$set_order(sheets)  
Arguments:  
sheets sheets  
Returns: The wbWorkbook object

Method get_sheet_visibility(): Get sheet visibility  
Usage:  
wbWorkbook$get_sheet_visibility()  

Method set_sheet_visibility(): Set sheet visibility  
Usage:  
wbWorkbook$set_sheet_visibility(sheet = current_sheet(), value)  
Arguments:  
sheet sheet  
value value

Method add_page_break(): Add a page break  
Usage:  
wbWorkbook$add_page_break(sheet = current_sheet(), row = NULL, col = NULL)  
Arguments:  
sheet sheet
Method clean_sheet(): clean sheet (remove all values)

Usage:
wbWorkbook$clean_sheet(
    sheet = current_sheet(),
    numbers = TRUE,
    characters = TRUE,
    styles = TRUE,
    merged_cells = TRUE
)

Arguments:
sheet sheet
numbers remove all numbers
characters remove all characters
styles remove all styles
merged_cells remove all merged_cells

Returns: The wbWorksheetObject, invisibly

Method add_border(): create borders for cell region

Usage:
wbWorkbook$add_border(
    sheet = current_sheet(),
    dims = "A1",
    bottom_color = wb_colour(hex = "FF000000"),
    left_color = wb_colour(hex = "FF000000"),
    right_color = wb_colour(hex = "FF000000"),
    top_color = wb_colour(hex = "FF000000"),
    bottom_border = "thin",
    left_border = "thin",
    right_border = "thin",
    top_border = "thin",
    inner_hgrid = NULL,
    inner_hcolor = NULL,
    inner_vgrid = NULL,
    inner_vcolor = NULL
)

Arguments:
sheet a worksheet
bottom_color, left_color, right_color, top_color, inner_hcolor, inner_vcolor a color,
either something openxml knows or some RGB color
left_border, right_border, top_border, bottom_border, inner_hgrid, inner_vgrid
the border style, if NULL no border is drawn. See create_border for possible border styles
Returns: The `wbWorksheetObject`, invisibly

Examples:

```r
wb <- wb_workbook()
w$b$add_worksheet("S1")$add_data("S1", mtcars)
w$b$add_border(1, dims = "A1:K1",
    left_border = NULL, right_border = NULL,
    top_border = NULL, bottom_border = "double")
w$b$add_border(1, dims = "A5",
    left_border = "dotted", right_border = "dotted",
    top_border = "hair", bottom_border = "thick")
w$b$add_border(1, dims = "C2:C5")
w$b$add_border(1, dims = "G2:H3")
w$b$add_border(1, dims = "G12:H13",
    left_color = wb_colour(hex = "FF9400D3"), right_color = wb_colour(hex = "FF4B0082"),
    top_color = wb_colour(hex = "FF0000FF"), bottom_color = wb_colour(hex = "FF00FF00"))
w$b$add_border(1, dims = "A20:C23")
w$b$add_border(1, dims = "B12:D14",
    left_color = wb_colour(hex = "FFFFFFFF"), right_color = wb_colour(hex = "FFFFFFFF"),
    bottom_color = wb_colour(hex = "FFFFFFFF"))
w$b$add_border(1, dims = "D28:E28")
# if (interactive()) wb$b$open()
```

Method `add_fill()`: provide simple fill function

Usage:

```r
wbWorkbook$add_fill(
    sheet = current_sheet(),
    dims = "A1",
    color = wb_colour(hex = "FFFFFFFF"),
    pattern = "solid",
    gradient_fill = "",
    every_nth_col = 1,
    every_nth_row = 1
)
```

Arguments:

- `sheet` the worksheet
- `dims` the cell range
- `color` the colors to apply, e.g. yellow: `wb_colour(hex = "FFFFFFFF")`
- `gradient_fill` a gradient fill xml pattern.
every_nth_col  which col should be filled
every_nth_row  which row should be filled

Returns:  The wbWorksheetObject, invisibly

Examples:

# example from the gradient fill manual page
gradient_fill <- "<gradientFill degree="90">
  <stop position="0"><color rgb="FF92D050"/></stop>
  <stop position="1"><color rgb="FF0070C0"/></stop>
</gradientFill>"

Method add_font(): provide simple font function

Usage:

wbWorkbook$add_font(
  sheet = current_sheet(),
  dims = "A1",
  name = "Calibri",
  color = wb_colour(hex = "FF000000"),
  size = "11",
  bold = "",
  italic = "",
  outline = "",
  strike = "",
  underline = "",
  charset = "",
  condense = "",
  extend = "",
  family = "",
  scheme = "",
  shadow = "",
  vertAlign = ""
)

Arguments:

sheet  the worksheet
dims  the cell range
name  font name: default "Calibri"
color  rgb color: default "FF000000"
size  font size: default "11",
bold  bold
italic  italic
outline  outline
strike  strike
underline  underline
charset  charset
condense  condense
extend  extend
family font family
scheme font scheme
shadow shadow
vertAlign vertical alignment

Returns: The \texttt{wbWorksheetObject}, invisibly

Examples:
\begin{verbatim}
wb <- wb_workbook()$add_worksheet("S1")$add_data("S1", mtcars)
wb$add_font("S1", "A1:K1", name = "Arial", color = wb_colour(theme = "4"))
\end{verbatim}

Method \texttt{add_numfmt()}: provide simple number format function

Usage:
\begin{verbatim}
wbWorkbook$add_numfmt(sheet = current_sheet(), dims = "A1", numfmt)
\end{verbatim}

Arguments:
- sheet the worksheet
- dims the cell range
- numfmt number format id or a character of the format

Returns: The \texttt{wbWorksheetObject}, invisibly

Examples:
\begin{verbatim}
wb <- wb_workbook()$add_worksheet("S1")$add_data("S1", mtcars)
wb$add_numfmt("S1", "A1:A33", numfmt = 1)
\end{verbatim}

Method \texttt{add_cell_style()}: provide simple cell style format function

Usage:
\begin{verbatim}
wbWorkbook$add_cell_style(
  sheet = current_sheet(),
  dims = "A1",
  applyAlignment = NULL,
  applyBorder = NULL,
  applyFill = NULL,
  applyFont = NULL,
  applyNumberFormat = NULL,
  applyProtection = NULL,
  borderId = NULL,
  extLst = NULL,
  fillId = NULL,
  fontId = NULL,
  hidden = NULL,
  horizontal = NULL,
  indent = NULL,
  justifyLastLine = NULL,
  locked = NULL,
  numFmtId = NULL,
  pivotButton = NULL,
  quotePrefix = NULL,
  readingOrder = NULL,
)\end{verbatim}
relativeIndent = NULL,
shrinkToFit = NULL,
textRotation = NULL,
vertical = NULL,
wrapText = NULL,
xfId = NULL
)

Arguments:
sheet the worksheet
dims the cell range
applyAlignment logical apply alignment
applyBorder logical apply border
applyFill logical apply fill
applyFont logical apply font
applyNumberFormat logical apply number format
applyProtection logical apply protection
borderId border ID to apply
extLst extension list something like <extLst>...</extLst>
fillId fill ID to apply
fontId font ID to apply
hidden logical cell is hidden
horizontal align content horizontal ('left', 'center', 'right')
indent logical indent content
justifyLastLine logical justify last line
locked logical cell is locked
numFmtId number format ID to apply
pivotButton unknown
quotePrefix unknown
readingOrder reading order left to right
relativeIndent relative indentation
shrinkToFit logical shrink to fit
textRotation degrees of text rotation
vertical vertical alignment of content ('top', 'center', 'bottom')
wrapText wrap text in cell
xfId xf ID to apply

Returns: The wbWorksheetObject, invisibly

Examples:
wb <- wb_workbook()$add_worksheet("S1")$add_data("S1", mtcars)
wb$add_cell_style("S1", "A1:K1",
    textRotation = "45",
    horizontal = "center",
    vertical = "center",
    wrapText = "1")
Method get_cell_style(): get sheet style

Usage:
wbWorkbook$get_cell_style(sheet = current_sheet(), dims)

Arguments:
sheet  sheet
dims   dims

Method set_cell_style(): set sheet style

Usage:
wbWorkbook$set_cell_style(sheet = current_sheet(), dims, style)

Arguments:
sheet  sheet
dims   dims
style  style

Returns: The wbWorksheetObject, invisibly

Method clone_sheet_style(): clone style from one sheet to another

Usage:
wbWorkbook$clone_sheet_style(from = current_sheet(), to)

Arguments:
from  the worksheet you are cloning
to   the worksheet the style is applied to

Method add_sparklines(): apply sparkline to worksheet

Usage:
wbWorkbook$add_sparklines(sheet = current_sheet(), sparklines)

Arguments:
sheet  the worksheet you are using
sparklines  sparkline created by create_sparkline()

Method clone(): The objects of this class are cloneable with this method.

Usage:
wbWorkbook$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.

See Also

create_border
Examples

```r
# Method `wbWorkbook$add_border`

wb <- wb_workbook()
wb$add_worksheet("S1")$add_data("S1", mtcars)
wb$add_border(1, dims = "A1:K1",
  left_border = NULL, right_border = NULL,
  top_border = NULL, bottom_border = "double")
wb$add_border(1, dims = "A5",
  left_border = "dotted", right_border = "dotted",
  top_border = "hair", bottom_border = "thick")
wb$add_border(1, dims = "C2:C5")
wb$add_border(1, dims = "A5")
wb$add_border(1, dims = "G2:H3")
wb$add_border(1, dims = "G12:H13",
  left_color = wb_colour(hex = "FF9400D3"), right_color = wb_colour(hex = "FF4B0082"),
  top_color = wb_colour(hex = "FF0000FF"), bottom_color = wb_colour(hex = "FF00FF00"))
wb$add_border(1, dims = "A20:C23")
wb$add_border(1, dims = "B12:D14",
  left_color = wb_colour(hex = "FFFF0000"), right_color = wb_colour(hex = "FFFF7F00"),
  bottom_color = wb_colour(hex = "FFFF0000"))
wb$add_border(1, dims = "D28:E28")
# if (interactive()) wb$open()

wb <- wb_workbook()
wb$add_worksheet("S1")$add_data("S1", mtcars)
w$add_border(1, dims = "A2:K33", inner_vgrid = "thin", inner_vcolor = c(rgb="FF808080"))

# Method `wbWorkbook$add_fill`

# example from the gradient fill manual page
gradient_fill <- "<gradientFill degree="90">
  <stop position="0"><color rgb="FF92D050"/></stop>
  <stop position="1"><color rgb="FF0070C0"/></stop>
</gradientFill>"

# Method `wbWorkbook$add_numfmt`

wb <- wb_workbook()$add_worksheet("S1")$add_data("S1", mtcars)
w$add_font("S1", "A1:K1", name = "Arial", color = wb_colour(theme = "4"))
```

# Method `wbWorkbook$add_numfmt`
```r
wb <- wb_workbook()$add_worksheet("S1")$add_data("S1", mtcars)
wb$add_numfmt("S1", "A1:A33", numfmt = 1)

## Method `wbWorkbook$add_cell_style`

wb <- wb_workbook()$add_worksheet("S1")$add_data("S1", mtcars)
wbad_cell_style("S1", "A1:K1",
  textRotation = "45",
  horizontal = "center",
  vertical = "center",
  wrapText = "1")
```

---

**wbWorksheet**  

*R6 class for a Workbook Worksheet*

---

**Description**

R6 class for a Workbook Worksheet  
R6 class for a Workbook Worksheet

---

**Value**

The `wbWorksheet` object  
The `wbWorksheet` object

---

**Public fields**

- `sheetPr`  
- `dimension`  
- `sheetViews`  
- `sheetFormatPr`  
- `sheet_data`  
- `cols_attr`  
- `autoFilter`  
- `mergeCells`  
- `conditionalFormatting`  
- `dataValidations`  
- `freezePane`
Methods

Public methods:

- `wbWorksheet$new()`
- `wbWorksheet$get_prior_sheet_data()`
- `wbWorksheet$get_post_sheet_data()`
- `wbWorksheet$unfold_cols()`
- `wbWorksheet$fold_cols()`
- `wbWorksheet$clean_sheet()`
• `wbWorksheet$add_page_break()`
• `wbWorksheet$set_print_options()`
• `wbWorksheet$append()`
• `wbWorksheet$add_sparklines()`
• `wbWorksheet$set_sheetview()`
• `wbWorksheet$clone()`

**Method new():** Creates a new `wbWorksheet` object

*Usage:*

```r
wbWorksheet$new(
  tabColour = NULL,
  oddHeader = NULL,
  oddFooter = NULL,
  evenHeader = NULL,
  evenFooter = NULL,
  firstHeader = NULL,
  firstFooter = NULL,
  paperSize = 9,
  orientation = "portrait",
  hdpi = 300,
  vdpi = 300,
  printGridLines = FALSE
)
```

*Arguments:*

- `tabColour`
- `oddHeader`
- `oddFooter`
- `evenHeader`
- `evenFooter`
- `firstHeader`
- `firstFooter`
- `paperSize`
- `orientation`
- `hdpi`
- `vdpi`
- `printGridLines`

*Returns:* a `wbWorksheet` object

**Method get_prior_sheet_data():** Get prior sheet data

*Usage:*

```r
wbWorksheet$get_prior_sheet_data()
```

*Returns:* A character vector of xml

**Method get_post_sheet_data():** Get post sheet data
Usage:
wbWorksheet$get_post_sheet_data()

Returns: A character vector of xml

Method unfold_cols(): unfold <cols ..> node to dataframe. <cols><col ..> are compressed. Only columns with attributes are written to the file. This function unfolds them so that each cell beginning with the "A" to the last one found in cc gets a value. TODO might extend this to match either largest cc or largest col. Could be that "Z" is formatted, but the last value is written to "Y". TODO might replace the xml nodes with the data frame?

Usage:
wbWorksheet$unfold_cols()

Returns: The column data frame

Method fold_cols(): fold the column dataframe back into a node.

Usage:
wbWorksheet$fold_cols(col_df)

Arguments:
col_df the column data frame

Returns: The wbWorksheetObject, invisibly

Method clean_sheet(): clean sheet (remove all values)

Usage:
wbWorksheet$clean_sheet(
  numbers = TRUE,
  characters = TRUE,
  styles = TRUE,
  merged_cells = TRUE
)

Arguments:
numbers remove all numbers
characters remove all characters
styles remove all styles
merged_cells remove all merged_cells

Returns: The wbWorksheetObject, invisibly

Method add_page_break(): add page break

Usage:
wbWorksheet$add_page_break(row = NULL, col = NULL)

Arguments:
row row
col col

Method set_print_options(): add print options
Usage:
wbWorksheet$set_print_options(
  gridLines = NULL,
  gridLinesSet = NULL,
  headings = NULL,
  horizontalCentered = NULL,
  verticalCentered = NULL
)

Arguments:
gridLines gridLines
gridLinesSet gridLinesSet
headings If TRUE prints row and column headings
horizontalCentered If TRUE the page is horizontally centered
verticalCentered If TRUE the page is vertically centered

Method append(): append a field. Intended for internal use only. Not guaranteed to remain a public method.

Usage:
wbWorksheet$append(field, value = NULL)

Arguments:
field a field name
value a new value

Returns: The wbWorksheetObject, invisibly

Method add_sparklines(): add sparkline

Usage:
wbWorksheet$add_sparklines(sparklines)

Arguments:
sparklines sparkline created by create_sparkline()

Returns: The wbWorksheetObject, invisibly

Method set_sheetview(): add sheetview

Usage:
wbWorksheet$set_sheetview(
  colorId = NULL,
  defaultGridColor = NULL,
  rightToLeft = NULL,
  showFormulas = NULL,
  showGridLines = NULL,
  showOutlineSymbols = NULL,
  showRowColHeaders = NULL,
  showRuler = NULL,
  showWhiteSpace = NULL,
  showZeros = NULL,
)
tabSelected = NULL,
topLeftCell = NULL,
view = NULL,
windowProtection = NULL,
workbookViewId = NULL,
zoomScale = NULL,
zoomScaleNormal = NULL,
zoomScalePageLayoutView = NULL,
zoomScaleSheetLayoutView = NULL
)

Arguments:
colorId colorId
defaultGridColor defaultGridColor
rightToLeft rightToLeft
showFormulas showFormulas
showGridLines showGridLines
showOutlineSymbols showOutlineSymbols
showRowColHeaders showRowColHeaders
showRuler showRuler
showWhiteSpace showWhiteSpace
showZeros showZeros
tabSelected tabSelected
topLeftCell topLeftCell
view view
windowProtection windowProtection
workbookViewId workbookViewId
zoomScale zoomScale
zoomScaleNormal zoomScaleNormal
zoomScalePageLayoutView zoomScalePageLayoutView
zoomScaleSheetLayoutView zoomScaleSheetLayoutView

Returns: The wbWorksheetObject, invisibly

Method clone(): The objects of this class are cloneable with this method.

Usage:
wbWorksheet$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.
Description

wb wrapper to create borders for cell region

Usage

```r
wb_add_border(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  bottom_color = wb_colour(hex = "FF000000"),
  left_color = wb_colour(hex = "FF000000"),
  right_color = wb_colour(hex = "FF000000"),
  top_color = wb_colour(hex = "FF000000"),
  bottom_border = "thin",
  left_border = "thin",
  right_border = "thin",
  top_border = "thin",
  inner_hgrid = NULL,
  inner_hcolor = NULL,
  inner_vgrid = NULL,
  inner_vcolor = NULL
)
```

Arguments

- **wb**: workbook
- **sheet**: a worksheet
- **bottom_color, left_color, right_color, top_color**: a color, either something `openxml` knows or some RGB color
- **inner_hgrid, inner_hcolor, inner_vgrid, inner_vcolor**: the border style, if NULL no border is drawn. See `create_border` for possible border styles

See Also

- `create_border()`

Other styles: `wb_add_cell_style()`, `wb_add_fill()`, `wb_add_font()`, `wb_add_numfmt()`, `wb_clone_sheet_style()`
Examples

```r
wb <- wb_workbook() %>% wb_add_worksheet("S1") %>% wb_add_data("S1", mtcars)
wb <- wb_add_border(wb, 1, dims = "A1:K1",
                   left_border = NULL, right_border = NULL,
                   top_border = NULL, bottom_border = "double")
wb <- wb_add_border(wb, 1, dims = "A5",
                   left_border = "dotted", right_border = "dotted",
                   top_border = "hair", bottom_border = "thick")
wb <- wb_add_border(wb, 1, dims = "C2:C5")
wb <- wb_add_border(wb, 1, dims = "G2:H3")
wb <- wb_add_border(wb, 1, dims = "G12:H13",
                   left_color = wb_colour(hex = "FF9400D3"),
                   right_color = wb_colour(hex = "FF4B0082"),
                   top_color = wb_colour(hex = "FF0000FF"),
                   bottom_color = wb_colour(hex = "FF00FF00"))
wb <- wb_add_border(wb, 1, dims = "A20:C23")
wb <- wb_add_border(wb, 1, dims = "B12:D14",
                   left_color = wb_colour(hex = "FFFFFFF00"),
                   right_color = wb_colour(hex = "FFFF7F00"),
                   bottom_color = wb_colour(hex = "FF0000FF"))
wb <- wb_add_border(wb, 1, dims = "D28:E28")
```

---

**Description**

add cell style for cell region

**Usage**

```r
wb_add_cell_style(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  applyAlignment = NULL,
  applyBorder = NULL,
  applyFill = NULL,
  applyFont = NULL,
  applyNumberFormat = NULL,
  applyProtection = NULL,
  borderId = NULL,
  extLst = NULL,
  fillId = NULL,
  fontId = NULL,
  hidden = NULL,
  horizontal = NULL,
  indent = NULL,
  justifyLastLine = NULL,
  locked = NULL,
  numFmtId = NULL,
  ```
pivotButton = NULL,
quotePrefix = NULL,
readingOrder = NULL,
relativeIndent = NULL,
shrinkToFit = NULL,
textRotation = NULL,
vertical = NULL,
wrapText = NULL,
xfId = NULL
)

Arguments

wb  a workbook
sheet the worksheet
dims the cell range
applyAlignment logical apply alignment
applyBorder logical apply border
applyFill logical apply fill
applyFont logical apply font
applyNumberOfFormat logical apply number format
applyProtection logical apply protection
borderId border ID to apply
extLst extension list something like <extLst>... </extLst>
fillId fill ID to apply
fontId font ID to apply
hidden logical cell is hidden
horizontal align content horizontal ('left', 'center', 'right')
indent logical indent content
justifyLastLine logical justify last line
locked logical cell is locked
numFmtId number format ID to apply
pivotButton unknown
quotePrefix unknown
readingOrder reading order left to right
relativeIndent relative indentation
shrinkToFit logical shrink to fit
textRotation degrees of text rotation
vertical vertical alignment of content ('top', 'center', 'bottom')
wrapText wrap text in cell
xfId xf ID to apply
Value

The `wbWorksheetObject`, invisibly

See Also

Other styles: `wb_add_border()`, `wb_add_fill()`, `wb_add_font()`, `wb_add_numfmt()`, `wb_clone_sheet_style()`

Examples

```r
wb <-
  wb_workbook() %>%
  wb_add_worksheet("S1") %>%
  wb_add_data("S1", mtcars)

wb %>%
  wb_add_cell_style(
    "S1",
    "A1:K1",
    textRotation = "45",
    horizontal = "center",
    vertical = "center",
    wrapText = "1"
  )
```

Description

Add a chartsheet to a workbook

Usage

```r
wb_add_chartsheet(
  wb,
  sheet = next_sheet(),
  tabColour = NULL,
  zoom = 100,
  visible = c("true", "false", "hidden", "visible", "veryhidden")
)
```

Arguments

- `wb`: A Workbook object to attach the new worksheet
- `sheet`: A name for the new worksheet
- `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.
- `visible`: If FALSE, sheet is hidden else visible.
Details

After chartsheet creation a chart must be added to the sheet. Otherwise the chartsheet will break the workbook.

See Also

wb_add_mschart()

Other workbook wrappers: wb_add_data_table(), wb_add_data(), wb_add_formula(), wb_add_worksheet(), wb_clone_worksheet(), wb_creators, wb_freeze-pane(), wb_get_base_font(), wb_save(), wb_set_col_widths(), wb_set_last_modified_by(), wb_set_row_heights(), wb_workbook(), workbook_grouping, ws_cell_merge

---

wb_add_chart_xml

**dummy function to add a chart to an existing workbook currently only a barplot is possible**

---

Description

dummy function to add a chart to an existing workbook currently only a barplot is possible

Usage

\[
\text{wb_add_chart_xml}(\text{wb}, \text{sheet} = \text{current_sheet}(), \text{xm}\text{l}, \text{dims} = \text{NULL})
\]

Arguments

- \text{wb} \quad \text{a workbook}
- \text{sheet} \quad \text{the sheet on which the graph will appear}
- \text{xm}\text{l} \quad \text{chart xml}
- \text{dims} \quad \text{the dimensions where the sheet will appear}

---

wb_add_conditional_formatting

**Add conditional formatting to cells**

---

Description

Add conditional formatting to cells
wb_add_conditional_formatting

Usage

wb_add_conditional_formatting(
  wb,
  sheet = current_sheet(),
  cols,
  rows,
  rule = NULL,
  style = NULL,
  type = c("expression", "colorScale", "dataBar", "duplicatedValues", "containsText", 
            "notContainsText", "beginsWith", "endsWith", "between", "topN", "bottomN"),
  params = list(showValue = TRUE, gradient = TRUE, border = TRUE, percent = FALSE, rank = 5L)
)

wb_conditional_formatting(
  wb,
  sheet,
  cols,
  rows,
  rule = NULL,
  style = NULL,
  type = c("expression", "colorScale", "dataBar", "duplicatedValues", "containsText", 
            "notContainsText", "beginsWith", "endsWith", "between", "topN", "bottomN"),
  ...
)

Arguments

wb       A workbook object
sheet    A name or index of a worksheet
cols     Columns to apply conditional formatting to
rows     Rows to apply conditional formatting to
rule     The condition under which to apply the formatting. See examples.
style    A style to apply to those cells that satisfy the rule. Default is 'font_color = "FF9C0006"' and 'bgFill = "FFFFC7CE"'
type     The type of conditional formatting rule to apply.
params   Additional parameters passed. See Details for more
...      passed to params

Details

See Examples.

Conditional formatting types accept different parameters. Unless noted, unlisted parameters are ignored.
expression [style]
A Style object

[rule]
An Excel expression (as a character). Valid operators are: <, <=, >, >=, ==, !=
colorScale [style]
A character vector of valid colors with length 2 or 3

dataBar [style]
A character vector of valid colors with length 2 or 3

[rule]
A numeric vector specifying the range of the databar colors. Must be equal length to style

[params$showValue]
If FALSE the cell value is hidden. Default TRUE

[params$gradient]
If FALSE colour gradient is removed. Default TRUE

[params$border]
If FALSE the border around the database is hidden. Default TRUE
duplicated [style]
A Style object
contains [style]
A Style object

[rule]
The text to look for within cells
between [style]
A Style object.

[rule]
A numeric vector of length 2 specifying lower and upper bound (Inclusive)
topN [style]
A Style object

[params$rank]
A numeric vector of length 1 indicating number of highest values. Default 5L

[params$percent] If TRUE uses percentage
bottomN [style]
A Style object

[params$rank]
A numeric vector of length 1 indicating number of lowest values. Default 5L
wb_add_data

[params$percent]
If TRUE uses percentage

Examples

wb <- wb_workbook()
wb$add_worksheet("a")
wb$add_data("a", 1:4, colNames = FALSE)
wb$add_conditional_formatting("a", 1, 1:4, ">2")

wb_add_data

Add data to a worksheet

Description

Add data to worksheet with optional styling.
Write an object to worksheet with optional styling.

Usage

wb_add_data(
  wb,
  sheet = current_sheet(),
  x,
  startCol = 1,
  startRow = 1,
  dims = rowcol_to_dims(startRow, startCol),
  array = FALSE,
  xy = NULL,
  colNames = TRUE,
  rowNames = FALSE,
  withFilter = FALSE,
  name = NULL,
  sep = ",", ",",
  applyCellStyle = TRUE,
  removeCellStyle = FALSE,
  na.strings
)

write_data(
  wb,
  sheet,
  x,
  startCol = 1,
  startRow = 1,
  dims = rowcol_to_dims(startRow, startCol),
  array = FALSE,
  xy = NULL,
  colNames = TRUE,
  rowNames = FALSE,
  withFilter = FALSE,
  name = NULL,
  sep = ",", ",",
  applyCellStyle = TRUE,
  removeCellStyle = FALSE,
  na.strings
)
array = FALSE,
xy = NULL,
colNames = TRUE,
rowNames = FALSE,
withFilter = FALSE,
sep = ",",
name = NULL,
applyCellStyle = TRUE,
removeCellStyle = FALSE,
na.strings
)

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.
withFilter If TRUE, add filters to the column name row. NOTE can only have one filter per worksheet.
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).
applyCellStyle apply styles when writing on the sheet
removeCellStyle if writing into existing cells, should the cell style be removed?
na.strings optional na.strings argument. if missing #N/A is used. If NULL no cell value is written, if character or numeric this is written (even if NA is part of numeric data)

Details

Formulae written using write_formula 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. The string "_openxlsx_NA" is reserved for openxlsx2. If the data frame contains this string, the output will be broken.
Formulae written using write_formula 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. The string "_openxlsx_NA" is reserved for openxlsx2. If the data frame contains this string, the output will be broken.

Value
A clone of `wb`
invisible(0)

See Also
write_datatable()

Other workbook wrappers: wb_add_chartsheet(), wb_add_data_table(), wb_add_formula(), wb_add_worksheet(), wb_clone_worksheet(), wb_creators, wb_freeze_pane(), wb_get_base_font(), wb_save(), wb_set_col_widths(), wb_set_last_modified_by(), wb_set_row_heights(), wb_workbook(), workbook_grouping, ws_cell_merge

Examples

## See formatting vignette for further examples.
## Options for default styling (These are the defaults)
options("openxlsx2.dateFormat" = "mm/dd/yyyy")
options("openxlsx2.datetimeFormat" = "yyyy-mm-dd hh:mm:ss")
options("openxlsx2.numFmt" = NULL)

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

## Add worksheets
wb$add_worksheet("Cars")
wb$add_worksheet("Formula")

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

## 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"
wb$add_data("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 = paste(paste0("A", 1:3 + 1L), paste0("B", 1:3 + 1L), sep = "+"),
  stringsAsFactors = FALSE
)

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

wb$add_data(sheet = "Formula", x = df)

# update cell range and add mtcars
xlsxFile <- system.file("extdata", "inline_str.xlsx", package = "openxlsx2")
wb2 <- wb_load(xlsxFile)

# read dataset with inlinestr
wb_to_df(wb2)
# read_xlsx(wb2)
write_data(wb2, 1, mtcars, startCol = 4, startRow = 4)
wb_to_df(wb2)

---

**wb_add_data_table**  
Add data to a worksheet as an Excel table

**Description**
Add data to a worksheet and format as an Excel table

**Usage**
```r
wb_add_data_table(
  wb, 
  sheet = current_sheet(), 
  x, 
  startCol = 1, 
  startRow = 1, 
  dims = rowcol_to_dims(startRow, startCol), 
  xy = NULL, 
  colNames = TRUE, 
  rowNames = FALSE, 
  tableStyle = "TableStyleLight9", 
  tableName = NULL, 
  withFilter = TRUE, 
  sep = ",", 
  firstColumn = FALSE, 
  lastColumn = FALSE, 
  bandedRows = TRUE, 
  bandedCols = FALSE,
)```

applyCellStyle = TRUE,
removeCellStyle = FALSE,
na.strings
)

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.
withFilter  If TRUE, columns with have filters in the first row.
sep  Only applies to list columns. The separator used to collapse list columns to a character vector e.g. sapply(x$list_column, paste, collapse = sep).

The below options correspond to Excel table options:

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

Table Style Options

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

applyCellStyle  Should we write cell styles to the workbook
removeCellStyle  keep the cell style?
na.strings  optional
wb_add_data_validation

Add data validation to cells

Description
Add Excel data validation to cells

Usage
```r
wb_add_data_validation(
    wb,
    sheet = current_sheet(),
    cols,
    rows,
    type,
    operator,
    value,
    allowBlank = TRUE,
    showInputMsg = TRUE,
    showErrorMsg = TRUE,
    errorStyle = NULL,
    errorTitle = NULL,
    error = NULL,
    promptTitle = NULL,
    prompt = NULL
)
```

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

Details
Columns of x with class Date/POSIXt, currency, accounting, hyperlink, percentage are automatically styled as dates, currency, accounting, hyperlinks, percentages respectively. The string "_openxlsx_NA" is reserved for openxlsx2. If the data frame contains this string, the output will be broken.

See Also
Other workbook wrappers: `wb_add_chartsheet()`, `wb_add_data()`, `wb_add_formula()`, `wb_add_worksheet()`, `wb_clone_worksheet()`, `wb_creators`, `wb_freeze_pane()`, `wb_get_base_font()`, `wb_save()`, `wb_set_col_widths()`, `wb_set_last_modified_by()`, `wb_set_row_heights()`, `wb_workbook()`, `workbook_grouping`, `ws_cell_merge`
**wb_add_data_validation**

- **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
- **errorStyle**: The icon shown and the options how to deal with such inputs. Default "stop" (cancel), else "information" (prompt popup) or "warning" (prompt accept or change input)
- **errorTitle**: The error title
- **error**: The error text
- **promptTitle**: The prompt title
- **prompt**: The prompt text

**Examples**

```r
wb <- wb_workbook()
wb$add_worksheet("Sheet 1")
wb$add_worksheet("Sheet 2")

wb$add_data_table(1, x = iris[1:30, ])
wb$add_data_validation(1,
    col = 1:3, rows = 2:31, type = "whole",
    operator = "between", value = c(1, 9)
)
wb$add_data_validation(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
)
wb$add_data_table(2, x = df)
wb$add_data_validation(2,
    col = 1, rows = 2:12, type = "date",
    operator = "greaterThanOrEqual", value = as.Date("2016-01-01")
)
wb$add_data_validation(2,
    col = 2, rows = 2:12, type = "time",
    operator = "between", value = df$t[c(4, 8)]
)
```

*******************************************************************************
## If type == 'list'
# operator argument is ignored.

```r
wb <- wb_workbook()
wb$add_worksheet("Sheet 1")
wb$add_worksheet("Sheet 2")

wb$add_data_table(sheet = 1, x = iris[1:30, ])
wb$add_data(sheet = 2, x = sample(iris$Sepal.Length, 10))

wb$add_data_validation(1, col = 1, rows = 2:31, type = "list", value = "'Sheet 2'!A$1:A$10")
```

---

**wb_add_drawing**

*add drawings to workbook*

### Description

Add drawings to workbook.

### Usage

```r
wb_add_drawing(wb, sheet = current_sheet(), xml, dims = NULL)
```

### Arguments

- **wb**: A `wbWorkbook`.
- **sheet**: A sheet in the workbook.
- **xml**: The drawing XML as character or file.
- **dims**: The dimension where the drawing is added. Can be NULL.

### Examples

```r
if (requireNamespace("rvg") && interactive()) {
  ## rvg example
  require(rvg)
  tmp <- tempfile(fileext = ".xml")
  dml_xlsx(file = tmp)
  plot(1,1)
  dev.off()

  wb <- wb_workbook()
  add_worksheet()
  add_drawing(xml = tmp)
  add_drawing(xml = tmp, dims = NULL)
}
```
wb_add_fill

**Description**

wb wrapper to create fill for cell region

**Usage**

```r
wb_add_fill(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  color = wb_colour(hex = "FFFFFF00"),
  pattern = "solid",
  gradient_fill = "",
  every_nth_col = 1,
  every_nth_row = 1
)
```

**Arguments**

- `wb` a workbook
- `sheet` the worksheet
- `dims` the cell range
- `color` the colors to apply, e.g. yellow: `wb_colour(hex = "FFFFFF00")`
- `gradient_fill` a gradient fill xml pattern.
- `every_nth_col` which col should be filled
- `every_nth_row` which row should be filled

**Value**

The `wbWorksheetObject`, invisibly

**See Also**

Other styles: `wb_add_border()`, `wb_add_cell_style()`, `wb_add_font()`, `wb_add_numfmt()`, `wb_clone_sheet_style()`
Examples

```r
wb <- wb_workbook() %>% wb_add_worksheet("S1") %>% wb_add_data("S1", mtcars)

wb <- wb %>% wb_add_fill("S1", dims = "D5:J23", color = wb_colour(hex = "FFFFFF00"))

wb <- wb %>% wb_add_fill("S1", dims = "B22:D27", color = wb_colour(hex = "FF00FF00"))

wb <- wb %>% wb_add_worksheet("S2") %>% wb_add_data("S2", mtcars)

gradient_fill1 <- '<gradientFill degree="90">
<stop position="0"><color rgb="FF92D050"/></stop>
<stop position="1"><color rgb="FF0070C0"/></stop>
</gradientFill>'

wb <- wb %>% wb_add_fill("S2", dims = "A2:K5", gradient_fill = gradient_fill1)

gradient_fill2 <- '<gradientFill type="path" left="0.2" right="0.8" top="0.2" bottom="0.8">
<stop position="0"><color theme="0"/></stop>
<stop position="1"><color theme="4"/></stop>
</gradientFill>'

wb <- wb %>% wb_add_fill("S2", dims = "A7:K10", gradient_fill = gradient_fill2)
```

---

**wb_add_filter**  
Add column filters

**Description**

Add excel column filters to a worksheet

**Usage**

```r
wb_add_filter(wb, sheet = current_sheet(), rows, cols)
```

**Arguments**

- `wb`: A workbook object
- `sheet`: A name or index of a worksheet
- `rows`: A row number
- `cols`: Columns to add filter to.

**Details**

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

**See Also**

- `write_data()`
- `wb_add_filter()`
Examples

```r
wb <- wb_workbook()
wb$add_worksheet("Sheet 1")
wb$add_worksheet("Sheet 2")
wb$add_worksheet("Sheet 3")

wb$add_data(1, iris)
wb$add_filter(1, row = 1, cols = seq_along(iris))

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

## Similarly
wb$add_data_table(3, iris)
```

## Description

add font for cell region

## Usage

```r
wb_add_font(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  name = "Calibri",
  color = wb_colour(hex = "FF000000"),
  size = "11",
  bold = "",
  italic = "",
  outline = "",
  strike = "",
  underline = "",
  charset = "",
  condense = "",
  extend = "",
  family = "",
  scheme = "",
  shadow = "",
  vertAlign = ""
)
```
**Arguments**

- **wb**: a workbook
- **sheet**: the worksheet
- **dims**: the cell range
- **name**: font name: default "Calibri"
- **color**: rgb color: default "FF000000"
- **size**: font size: default "11",
- **bold**: bold, "single" or "double", default: ""
- **italic**: italic
- **outline**: outline
- **strike**: strike
- **underline**: underline
- **charset**: charset
- **condense**: condense
- **extend**: extend
- **family**: font family
- **scheme**: font scheme
- **shadow**: shadow
- **vertAlign**: vertical alignment

**Details**

`add_font` provides all the options openxml accepts for a font node, not all have to be set. Usually name, size and color should be what the user wants.

**Value**

The `wbWorksheetObject`, invisibly

**See Also**

Other styles: `wb_add_border()`, `wb_add_cell_style()`, `wb_add_fill()`, `wb_add_numfmt()`, `wb_clone_sheet_style()`

**Examples**

```r
wb <- wb_workbook() %>% wb_add_worksheet("S1") %>% wb_add_data("S1", mtcars)
wbn%>%wb_add_font("S1", "A1:K1", name = "Arial", color = wb_colour(theme = "4"))
```
wb_add_formula

Add a character vector as an Excel Formula

Description

Add a character vector containing Excel formula to a worksheet.

Usage

wb_add_formula(
  wb,
  sheet = current_sheet(),
  x,
  startCol = 1,
  startRow = 1,
  dims = rowcol_to_dims(startRow, startCol),
  array = FALSE,
  xy = NULL,
  applyCellStyle = TRUE,
  removeCellStyle = FALSE
)

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).
applyCellStyle Should we write cell styles to the workbook
removeCellStyle keep the cell style?

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)
• ...

See Also

Other workbook wrappers: wb_add_chartsheet(), wb_add_data_table(), wb_add_data(), wb_add_worksheet(), wb_clone_worksheet(), wb_creators, wb_freeze_pane(), wb_get_base_font(), wb_save(), wb_set_col_widths(), wb_set_last_modified_by(), wb_set_row_heights(), wb_workbook(), workbook_grouping, ws_cell_merge

Insert an image into a worksheet

Description

Insert an image into a worksheet

Usage

wb_add_image(
    wb,
    sheet = current_sheet(),
    file,
    width = 6,
    height = 3,
    startRow = 1,
    startCol = 1,
    rowOffset = 0,
    colOffset = 0,
    units = "in",
    dpi = 300
)

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>file</td>
<td>An image file. Valid file types are: &quot;jpeg&quot;, &quot;png&quot;, &quot;bmp&quot;</td>
</tr>
<tr>
<td>width</td>
<td>Width of figure.</td>
</tr>
<tr>
<td>height</td>
<td>Height of figure.</td>
</tr>
<tr>
<td>startRow</td>
<td>Row coordinate of upper left corner of the image</td>
</tr>
<tr>
<td>startCol</td>
<td>Column coordinate of upper left corner of the image</td>
</tr>
<tr>
<td>rowOffset</td>
<td>offset within cell (row)</td>
</tr>
<tr>
<td>colOffset</td>
<td>offset within cell (column)</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 used for conversion between units.</td>
</tr>
</tbody>
</table>
\texttt{wb_add_mschart}

\textbf{See Also}

\texttt{wb_add_plot()}

\textbf{Examples}

\begin{verbatim}
## Create a new workbook
wb <- wb_workbook("Ayanami")

## Add some worksheets
wb$add_worksheet("Sheet 1")
wb$add_worksheet("Sheet 2")
wb$add_worksheet("Sheet 3")

## Insert images
img <- system.file("extdata", "einstein.jpg", package = "openxlsx2")
wb$add_image("Sheet 1", img, startRow = 5, startCol = 3, width = 6, height = 5)
wb$add_image(2, img, startRow = 2, startCol = 2)
wb$add_image(3, img, width = 15, height = 12, startRow = 3, startCol = "G", units = "cm")
\end{verbatim}

\textbf{wb_add_mschart} \hspace{2cm} \textit{Add mschart object to an existing workbook}

\textbf{Description}

Add mschart object to an existing workbook

\textbf{Usage}

\texttt{wb_add_mschart(wb, sheet = current_sheet(), dims = NULL, graph)}

\textbf{Arguments}

- \texttt{wb} \hspace{1cm} a workbook
- \texttt{sheet} \hspace{1cm} the sheet on which the graph will appear
- \texttt{dims} \hspace{1cm} the dimensions where the sheet will appear
- \texttt{graph} \hspace{1cm} mschart object

\textbf{See Also}

\texttt{wb_data()}
Examples

```r
if (requireNamespace("mschart")) {
  require(mschart)

  ## Add mschart to worksheet (adds data and chart)
  scatter <- ms_scatterchart(data = iris, x = "Sepal.Length", y = "Sepal.Width", group = "Species")
  scatter <- chart_settings(scatter, scatterstyle = "marker")
  wb <- wb_workbook() %>%
    wb_add_worksheet() %>%
    wb_add_mschart(dims = "F4:L20", graph = scatter)

  ## Add mschart to worksheet and use available data
  wb <- wb_workbook() %>%
    wb_add_worksheet() %>%
    wb_add_data(x = mtcars, dims = "B2")

  # create wb.data object
  dat <- wb_data(wb, 1, dims = "B2:E6")

  # call ms_scatterplot
  data_plot <- ms_scatterchart(
    data = dat,
    x = "mpg",
    y = c("disp", "hp"),
    labels = c("disp", "hp")
  )

  # add the scatterplot to the data
  wb <- wb %>%
    wb_add_mschart(dims = "F4:L20", graph = data_plot)
}
```

---

**wb_add_numfmt**

Add numfmt for cell region

**Description**

Add numfmt for cell region

**Usage**

`wb_add_numfmt(wb, sheet = current_sheet(), dims = "A1", numfmt)`

**Arguments**

- `wb`: a workbook
- `sheet`: the worksheet
- `dims`: the cell range
- `numfmt`: either an id or a character
Value
The `wbWorksheetObject`, invisibly

See Also
Other styles: `wb_add_border()`, `wb_add_cell_style()`, `wb_add_fill()`, `wb_add_font()`, `wb_clone_sheet_style()`

Examples
```r
wb <- wb_workbook() %>% wb_add_worksheet("S1") %>% wb_add_data("S1", mtcars)
wb %>% wb_add_numfmt("S1", dims = "F1:F33", numfmt = ".0")
```

---

**wb_add_page_break**  
Add a page break to a worksheet

Description
Insert page breaks into a worksheet

Usage
```r
wb_add_page_break(wb, sheet = current_sheet(), row = NULL, col = NULL)
```

Arguments
- `wb` A workbook object
- `sheet` A name or index of a worksheet
- `row, col` Either a row number or column number. One must be `NULL`

See Also
- `wb_add_worksheet()`

Examples
```r
wb <- wb_workbook()
w$add_worksheet("Sheet 1")
w$add_data(sheet = 1, x = iris)

w$add_page_break(sheet = 1, row = 10)
w$add_page_break(sheet = 1, row = 20)
w$add_page_break(sheet = 1, col = 2)

## In Excel: View tab -> Page Break Preview
```
wb_add_plot

Insert the current plot into a worksheet

Description

The current plot is saved to a temporary image file using grDevices::dev.copy() This file is then written to the workbook using wb_add_image().

Usage

wb_add_plot(
  wb,
  sheet = current_sheet(),
  width = 6,
  height = 4,
  xy = NULL,
  startRow = 1,
  startCol = 1,
  rowOffset = 0,
  colOffset = 0,
  fileType = "png",
  units = "in",
  dpi = 300
)

Arguments

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

See Also

wb_add_image()
Examples

```r
if (requireNamespace("ggplot2") && interactive()) {
  ## Create a new workbook
  wb <- wb_workbook()

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

  ## create plot objects
  require(ggplot2)
p1 <- ggplot(mtcars, aes(x = mpg, fill = as.factor(gear))) +
    geom_density(alpha = I(.5))
p2 <- ggplot(Orange, aes(x = age, y = circumference, colour = Tree)) +
    geom_point() + geom_line()

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

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

---

`wb_add_sparklines`  
*add sparklines to workbook*

**Description**

add sparklines to workbook

**Usage**

`wb_add_sparklines(wb, sheet = current_sheet(), sparklines)`

**Arguments**

- `wb` : workbook
- `sheet` : sheet to add the sparklines to
- `sparklines` : sparklines object created with `create_sparklines()`

**See Also**

`create_sparklines()`
Examples

```r
sl <- create_sparklines("Sheet 1", "A3:K3", "L3")
w <- wb_workbook() %>%
  wb_add_worksheet() %>%
  wb_add_data(x = mtcars) %>%
  wb_add_sparklines(sparklines = sl)
```

```
wb_add_style

---

add style to workbook

---

Description

wb wrapper to add style to workbook

Usage

```r
wb_add_style(wb, style = NULL, style_name = NULL)
```

Arguments

- **wb**: workbook
- **style**: style xml character
- **style_name**: style name used optional argument

See Also

`create_border()`, `create_cell_style()`, `create_dxfs_style()`, `create_fill()`, `create_font()`, `create_numfmt()`

Examples

```r
yellow_f <- wb_colour(hex = "FF9C6500")
yellow_b <- wb_colour(hex = "FFFFEB9C")

yellow <- create_dxfs_style(font_color = yellow_f, bgFill = yellow_b)
w <- wb_workbook() %>%
  wb_add_style(yellow)
```
wb_add_worksheet

Add a worksheet to a workbook

Description
Add a worksheet to a workbook

Usage

```r
wb_add_worksheet(
  wb,
  sheet = next_sheet(),
  gridLines = TRUE,
  rowColHeaders = TRUE,
  tabColour = NULL,
  zoom = 100,
  header = NULL,
  footer = NULL,
  oddHeader = header,
  oddFooter = footer,
  evenHeader = header,
  evenFooter = footer,
  firstHeader = header,
  firstFooter = footer,
  visible = c("true", "false", "hidden", "visible", "veryhidden"),
  hasDrawing = FALSE,
  paperSize = getOption("openxlsx2.paperSize", default = 9),
  orientation = getOption("openxlsx2.orientation", default = "portrait"),
  hdpi = getOption("openxlsx2.hdpi", default = getOption("openxlsx2.dpi", default = 300)),
  vdpi = getOption("openxlsx2.vdpi", default = getOption("openxlsx2.dpi", default = 300))
)
```

Arguments

- **wb**: A Workbook object to attach the new worksheet
- **sheet**: A name for the new worksheet
- **gridLines**: A logical. If FALSE, the worksheet grid lines will be hidden.
- **rowColHeaders**: A logical. If FALSE, the worksheet colname and rowname 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, oddHeader, evenHeader, firstHeader, footer, oddFooter, evenFooter, firstFooter**: Character vector of length 3 corresponding to positions left, center, right. header and footer are used to default additional arguments. Setting even, odd, or first, overrides header/footer. Use NA to skip a position.
visible    If FALSE, sheet is hidden else visible.
hasDrawing If TRUE prepare a drawing output (TODO does this work?)
paperSize  An integer corresponding to a paper size. See \?ws_page_setup for details.
orientation One of "portrait" or "landscape"
hdpi       Horizontal DPI. Can be set with options("openxlsx2.dpi" = X) or options("openxlsx2.hdpi" = X)
vdpi       Vertical DPI. Can be set with options("openxlsx2.dpi" = X) or options("openxlsx2.vdpi" = 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

The \texttt{wbWorkbook} object \texttt{wb}

See Also

Other workbook wrappers: \texttt{wb_add_chartsheet()}, \texttt{wb_add_data_table()}, \texttt{wb_add_data()}, \texttt{wb_add_formula()}, \texttt{wb_clone_worksheet()}, \texttt{wb_creators}, \texttt{wb_freeze_pane()}, \texttt{wb_get_base_font()}, \texttt{wb_save()}, \texttt{wb_set_col_widths()}, \texttt{wb_set_last_modified_by()}, \texttt{wb_set_row_heights()}, \texttt{wb_workbook()}, \texttt{workbook_grouping}, \texttt{ws_cell_merge}

Examples

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

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

## Headers and Footers
wb$add_worksheet("Sheet 5",
                   header = c("ODD HEAD LEFT", "ODD HEAD CENTER", "ODD HEAD RIGHT"),
                   footer = c("ODD FOOT RIGHT", "ODD FOOT CENTER", "ODD FOOT RIGHT"),
                   ...)
```
wb_clone_sheet_style

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")
)

wb$add_worksheet("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)
)

wb$add_worksheet("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")
)

wb$add_worksheet("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
wb$add_data(sheet = 5, 1:400)
wb$add_data(sheet = 6, 1:400)
wb$add_data(sheet = 7, 1:400)
wb$add_data(sheet = 8, 1:400)

---

**Description**

clone sheets style

**Usage**

`wb_clone_sheet_style(wb, from = current_sheet(), to)`

**Arguments**

- `wb` workbook
- `from` sheet we select the style from
- `to` sheet we apply the style from

**See Also**

Other styles: `wb_add_border()`, `wb_add_cell_style()`, `wb_add_fill()`, `wb_add_font()`, `wb_add_numfmt()`
wb_clone_worksheet  
*Clone a worksheet to a workbook*

**Description**

Clone a worksheet to a Workbook object

**Usage**

```r
wb_clone_worksheet(wb, old = current_sheet(), new = next_sheet())
```

**Arguments**

- **wb**: A `wbWorkbook` object
- **old**: Name of existing worksheet to copy
- **new**: Name of new worksheet to create

**Value**

The `wb` object

**See Also**

Other workbook wrappers: `wb_add_chartsheet()`, `wb_add_data_table()`, `wb_add_data()`, `wb_add_formula()`, `wb_add_worksheet()`, `wb_creators`, `wb_freezePane()`, `wb_get_base_font()`, `wb_save()`, `wb_set_col_widths()`, `wb_set_last_modified_by()`, `wb_set_row_heights()`, `wb_workbook()`, `workbook_grouping`, `ws_cell_merge`

**Examples**

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

# Add worksheets
wb$add_worksheet("Sheet 1")
wb$clone_worksheet("Sheet 1", "Sheet 2")
```
**wb_colour**  
*Create a new hyperlink object*

**Description**
Create a new hyperlink object

**Usage**

```r
wb_colour(
    name = NULL,
    auto = NULL,
    indexed = NULL,
    hex = NULL,
    theme = NULL,
    tint = NULL
)
```

**Arguments**
- **name**: A name of a color known to R
- **auto**: A boolean.
- **indexed**: An indexed color values.
- **hex**: A rgb color as ARGB hex value "FF000000".
- **theme**: A zero based index referencing a value in the theme.
- **tint**: A tint value applied. Range from -1 (dark) to 1 (light).

**Value**
a `wbColour` object

---

**wb_creators**  
*Workbook creators*

**Description**
Modify and get workbook creators

**Usage**

```r
wb_add_creators(wb, creators)
wb_set_creators(wb, creators)
wb_remove_creators(wb, creators)
wb_get_creators(wb)
```
wb_data

provide wb_data object as mschart input

**Description**

provide wb_data object as mschart input

**Usage**

```
wb_data(wb, sheet = current_sheet(), dims, ...)
```
wb_freeze_pane

Arguments

wb
a workbook

sheet
a sheet in the workbook either name or index

dims
the dimensions

... additional arguments for wb_to_df. Be aware that not every argument is valid.

See Also

wb_to_df() wb_add_mschart()

Examples

wb <- wb_workbook() %>%
   wb_add_worksheet() %>%
   wb_add_data(x = mtcars, dims = "B2")

wb_data(wb, 1, dims = "B2:E6")

wb_freeze_pane

Freeze a worksheet pane

Description

Freeze a worksheet pane

Usage

wb_freeze_pane(
  wb,
  sheet = current_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)
See Also

Other workbook wrappers: `wb_add_chartsheet()`, `wb_add_data_table()`, `wb_add_data()`, `wb_add_formula()`, `wb_add_worksheet()`, `wb_clone_worksheet()`, `wb_creators`, `wb_get_base_font()`, `wb_save()`, `wb_set_col_widths()`, `wb_set_last_modified_by()`, `wb_set_row_heights()`, `wb_workbook()`, `workbook_grouping`, `ws_cell_merge`

Examples

```r
## Create a new workbook
wb <- wb_workbook("Kenshin")

## Add some worksheets
wb$add_worksheet("Sheet 1")
wb$add_worksheet("Sheet 2")
wb$add_worksheet("Sheet 3")
wb$add_worksheet("Sheet 4")

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

Description

Get the base font used in the workbook.

Usage

```r
wb_get_base_font(wb)
```

Arguments

- `wb` A `wbWorkbook` object

See Also

Other workbook wrappers: `wb_add_chartsheet()`, `wb_add_data_table()`, `wb_add_data()`, `wb_add_formula()`, `wb_add_worksheet()`, `wb_clone_worksheet()`, `wb_creators`, `wb_freeze_pane()`, `wb_save()`, `wb_set_col_widths()`, `wb_set_last_modified_by()`, `wb_set_row_heights()`, `wb_workbook()`, `workbook_grouping`, `ws_cell_merge`
wb_get_sheet_name

Examples

```r
## create a workbook
wb <- wb_workbook()
wb_get_base_font(wb)

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

wb_get_sheet_name  
Get sheet name

Description

Get sheet name

Usage

```r
wb_get_sheet_name(wb, index = NULL)
```

Arguments

- `wb` a `wbWorkbook` object
- `index` Sheet name index

Value

The sheet index

wb_get_sheet_names  
Get worksheet names for a workbook

Description

 Gets the worksheet names for a `wbWorkbook` object

Usage

```r
wb_get_sheet_names(wb)
```

Arguments

- `wb` A `wbWorkbook` object

Value

A named character vector of sheet names in their order. The names represent the original value of the worksheet prior to any character substitutions.
wb_get_worksheet

List Excel tables in a workbook

Description
List Excel tables in a workbook

Usage
wb_get_tables(wb, sheet = current_sheet())

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

Value
character vector of table names on the specified sheet

Examples
wb <- wb_workbook()
wbs$add_worksheet(sheet = "Sheet 1")
wbs$add_data_table(sheet = "Sheet 1", x = iris)
wbs$add_data_table(sheet = 1, x = mtcars, tableName = "mtcars", startCol = 10)
wbs$get_tables(sheet = "Sheet 1")

wb_get_worksheet
Get a worksheet from a wbWorkbook object

Description
Get a worksheet from a wbWorkbook object

Usage
wb_get_worksheet(wb, sheet)
wb_ws(wb, sheet)

Arguments
wb      a wbWorkbook object
sheet   A sheet name or index
**wb_grid_lines**

Set worksheet gridlines to show or hide.

**Value**

A `wbWorksheet` object

**Description**

Set worksheet gridlines to show or hide.

**Usage**

```r
wb_grid_lines(wb, sheet = current_sheet(), show = FALSE, print = show)
```

**Arguments**

- **wb**
  - A workbook object
- **sheet**
  - A name or index of a worksheet
- **show**
  - A logical. If `FALSE`, grid lines are hidden.
- **print**
  - A logical. If `FALSE`, grid lines are not printed.

**Examples**

```r
wb <- wb_load(file = system.file("extdata", "loadExample.xlsx", package = "openxlsx2"))
wb$get_sheet_names()  ## list worksheets in workbook
wb$grid_lines(1, show = FALSE)
wb$grid_lines("testing", show = FALSE)
```

---

**wb_hyperlink**

Create a new hyperlink object

**Description**

Create a new hyperlink object

**Usage**

```r
wb_hyperlink()
```
wb_load  

Load an existing .xlsx file

Description

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

Usage

wb_load(file, xlsxFile = NULL, sheet, data_only = FALSE, calc_chain = FALSE)

Arguments

- **file**: A path to an existing .xlsx or .xlsm file
- **xlsxFile**: alias for file
- **sheet**: optional sheet parameter. if this is applied, only the selected sheet will be loaded.
- **data_only**: mode to import if only a data frame should be returned. This strips the wbWorkbook to a bare minimum.
- **calc_chain**: optionally you can keep the calculation chain intact. This is used by spreadsheet software to identify the order in which formulas are evaluated. Removing the calculation chain is considered harmless. The calc chain will be created upon the next time the worksheet is loaded in spreadsheet software. Keeping it, might only speed loading time in said software.

Details

A warning is displayed if an xml namespace for main is found in the xlsx file. Certain xlsx files created by third-party applications contain a namespace (usually `x`). This namespace is not required for the file to work in spreadsheet software and is not expected by openxlsx2. Therefore it is removed when the file is loaded into a workbook. Removal is generally expected to be safe, but the feature is still experimental.

Value

Workbook object.

See Also

- wb_remove_worksheet()

Examples

```r
## load existing workbook from package folder
wb <- wb_load(file = system.file("extdata", "loadExample.xlsx", package = "openxlsx2"))
wb$get_sheet_names() # list worksheets
wb ## view object
## Add a worksheet
wb$add_worksheet("A new worksheet")
```
**wb_modify_basefont**  
*Modify the default font*

**Description**

Modify the default font for this workbook

**Usage**

```r
wb_set_base_font(
  wb,
  fontSize = 11,
  fontColour = wb_colour(theme = "1"),
  fontName = "Calibri"
)
```

**Arguments**

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

**Details**

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

**Examples**

```r
## create a workbook
wb <- wb_workbook()
bw$add_worksheet("S1")
## modify base font to size 10 Arial Narrow in red
wb$set_base_font(fontSize = 10, fontColour = "#FF0000", fontName = "Arial Narrow")

wb$add_data("S1", iris)
bw$add_data_table("S1", x = iris, startCol = 10) ## font colour does not affect tables
```
**wb_open**

*little worksheet opener*

**Description**

little worksheet opener

**Usage**

```python
wb_open(wb)
```

**Arguments**

- `wb` a workbook

---

**wb_order**

*Order of worksheets in xlsx file*

**Description**

Get/set order of worksheets in a Workbook object

**Usage**

```python
wb_get_order(wb)
```

```python
wb_set_order(wb, sheets)
```

**Arguments**

- `wb` A `wbWorkbook` object
- `sheets` Sheet order

**Details**

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

```r
## setup a workbook with 3 worksheets
wb <- wb_workbook()
wb$add_worksheet("Sheet 1", gridLines = FALSE)
wb$add_data_table(sheet = 1, x = iris)

wb$add_worksheet("mtcars (Sheet 2)", gridLines = FALSE)
wb$add_data(sheet = 2, x = mtcars)

wb$add_worksheet("Sheet 3", gridLines = FALSE)
wb$add_data(sheet = 3, x = Formaldehyde)

wb_get_order(wb)
wb$get_sheet_names() ## ordering within workbook is not changed

wb$set_order(c(1, 3, 2)) # switch position of sheets 2 & 3
wb$add_data(2, 'This is still the "mtcars" worksheet', startCol = 15)  
wb_get_order(wb)
wb$get_sheet_names() ## ordering within workbook is not changed

wb$set_order(3:1)
```

---

**wb_protect**  
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
wb_protect(  
wb,  
protect = TRUE,  
password = NULL,  
lockStructure = FALSE,  
lockWindows = FALSE,  
type = c("1", "2", "4", "8"),  
fileSharing = FALSE,  
username = unname(Sys.info()["user"]),  
readOnlyRecommended = FALSE  
)
```

**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 (see details)
fileSharing   Whether to enable a popup requesting the unlock password is prompted
username      The username for the fileSharing popup
readOnlyRecommended Whether or not a post unlock message appears stating that the workbook is recommended to be opened in readonly mode.

Details

Lock types:

1  xlsx with password (default)
2  xlsx recommends read-only
4  xlsx enforces read-only
8  xlsx is locked for annotation

Examples

wb <- wb_workbook()
wbsadd_worksheet("S1")
wb_protect(wb, protect = TRUE, password = "Password", lockStructure = TRUE)

# Remove the protection
wb_protect(wb, protect = FALSE)

wb <- wb_protect(  
  wb,  
  protect = TRUE,  
  password = "Password",  
  lockStructure = TRUE,  
  type = 2L,  
  fileSharing = TRUE,  
  username = "Test",  
  readOnlyRecommended = TRUE
)

wb_protect_worksheet  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.
wb_protect_worksheet(
    wb,
    sheet = current_sheet(),
    protect = TRUE,
    password = NULL,
    properties = 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
properties A character vector of properties to lock. Can be one or more of the following:
  "selectLockedCells", "selectUnlockedCells", "formatCells", "formatColumns",
  "formatRows", "insertColumns", "insertRows", "insertHyperlinks", "deleteColumns",
  "deleteRows", "sort", "autoFilter", "pivotTables", "objects", "scenarios"

Examples
wb <- wb_workbook()
w$b$add_worksheet("S1")
w$b$add_data_table(1, x = iris[1:30, ])
# Formatting cells / columns is allowed, but inserting / deleting columns is protected:
w$b$protect_worksheet("S1",
    protect = TRUE,
    properties = c("formatCells", "formatColumns", "insertColumns", "deleteColumns")
)

# Remove the protection
w$b$protect_worksheet("S1", protect = FALSE)

wb_read Read from an Excel file or Workbook object

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

Usage
wb_read(
    xlsxFile,
    sheet = 1,
startRow = 1,
startCol = NULL,
rowNames = FALSE,
colNames = TRUE,
skipEmptyRows = FALSE,
skipEmptyCols = FALSE,
rows = NULL,
cols = NULL,
detectDates = TRUE,
namedRegion,
na.strings = "NA",
na.numbers = NA,
...
)

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.
startCol  first column to begin looking for data.
rowNames  If TRUE, first column of data will be used as row names.
colNames  If TRUE, the first row of data will be used as column names.
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.
detectDates If TRUE, attempt to recognize dates and perform conversion.
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.
na.numbers A numeric vector of digits which are to be interpreted as NA. Blank cells will
          be returned as NA.
...      additional arguments passed to wb_to_df()

Details

Creates a data.frame of all data in worksheet.

Value

data.frame
wb_remove_col_widths

See Also

get_named_regions(), wb_to_df(), read_xlsx()

Examples

```r
xlsxFile <- system.file("extdata", "readTest.xlsx", package = "openxlsx2")
df1 <- wb_read(xlsxFile = xlsxFile, sheet = 1)

xlsxFile <- system.file("extdata", "readTest.xlsx", package = "openxlsx2")
df1 <- wb_read(xlsxFile = xlsxFile, sheet = 1, rows = c(1, 3, 5), cols = 1:3)
```

---

**wb_remove_col_widths**  
Remove column widths from a worksheet

**Description**

Remove column widths from a worksheet

**Usage**

```r
wb_remove_col_widths(wb, sheet = current_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.

**See Also**

`wb_set_col_widths()`

**Examples**

```r
## Create a new workbook
wb <- wb_load(file = system.file("extdata", "loadExample.xlsx", package = "openxlsx2"))

## remove column widths in columns 1 to 20
wb_remove_col_widths(wb, 1, cols = 1:20)
```
### wb_remove_filter

**Description**

Removes filters from `wb_add_filter()` and `write_data()`

**Usage**

```r
wb_remove_filter(wb, sheet = current_sheet())
```

**Arguments**

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

**Examples**

```r
wb <- wb_workbook()
w$b$add_worksheet("Sheet 1")
w$b$add_worksheet("Sheet 2")
w$b$add_worksheet("Sheet 3")

wb$b$add_data(1, iris)
w$b$add_filter(wb, 1, row = 1, cols = seq_along(iris))

## Equivalently
w$b$add_data(2, x = iris, withFilter = TRUE)

## Similarly
w$b$add_data_table(3, iris)

## remove filters
wb_remove_filter(wb, 1:2) ## remove filters
wb_remove_filter(wb, 3) ## Does not affect tables!
```

### wb_remove_row_heights

**Description**

Remove custom row heights from a worksheet

**Usage**

```r
wb_remove_row_heights(wb, sheet = current_sheet(), rows)
```
### wb_remove_tables

**Description**

List Excel tables in a workbook

**Usage**

```r
wb_remove_tables(wb, sheet = current_sheet(), table)
```

**Arguments**

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

**Value**

character vector of table names on the specified sheet

**Examples**

```r
wb <- wb_workbook()
wbs$add_worksheet(sheet = "Sheet 1")
wbs$add_worksheet(sheet = "Sheet 2")
wbs$add_data_table(sheet = "Sheet 1", x = iris, tableName = "iris")
wbs$add_data_table(sheet = 1, x = mtcars, tableName = "mtcars", startCol = 10)
```
wb <- wb_remove_worksheet(wb, sheet = 1) ## delete worksheet removes table objects

wb$add_data_table(sheet = 1, x = iris, tableName = "iris")
wb$add_data_table(sheet = 1, x = mtcars, tableName = "mtcars", startCol = 10)

## wb_remove_tables() deletes table object and all data
wb$remove_tables(sheet = 1, table = "iris")
wb$add_data_table(sheet = 1, x = iris, tableName = "iris", startCol = 1)

wb$remove_tables(sheet = 1, table = "iris")
wb$add_data_table(sheet = 1, x = iris, tableName = "iris", startCol = 1)
wb_save

Save Workbook to file

Description

Save Workbook to file

Usage

wb_save(wb, path = NULL, overwrite = TRUE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wb</td>
<td>A wbWorkbook object to write to file</td>
</tr>
<tr>
<td>path</td>
<td>A path to save the workbook to</td>
</tr>
<tr>
<td>overwrite</td>
<td>If FALSE, will not overwrite when path exists</td>
</tr>
</tbody>
</table>

Value

the wbWorkbook object, invisibly

See Also

Other workbook wrappers: wb_add_chartsheet(), wb_add_data_table(), wb_add_data(), wb_add_formula(), wb_add_worksheet(), wb_clone_worksheet(), wb_creators, wb_freeze_pane(), wb_get_base_font(), wb_set_col_widths(), wb_set_last_modified_by(), wb_set_row_heights(), wb_workbook(), workbook_grouping, ws_cell_merge

Examples

```r
## Create a new workbook and add a worksheet
wb <- wb_workbook("Creator of workbook")
wb$add_worksheet(sheet = "My first worksheet")

## Save workbook to working directory

wb_save(wb, path = temp_xlsx(), overwrite = TRUE)
```
wb_set_bookview

Set the workbook position, size and filter

Description

Get the base font used in the workbook.

Usage

```r
wb_set_bookview(
  wb,
  activeTab = NULL,
  autoFilterDateGrouping = NULL,
  firstSheet = NULL,
  minimized = NULL,
  showHorizontalScroll = NULL,
  showSheetTabs = NULL,
  showVerticalScroll = NULL,
  tabRatio = NULL,
  visibility = NULL,
  windowHeight = NULL,
  windowWidth = NULL,
  xWindow = NULL,
  yWindow = NULL
)
```

Arguments

- **wb**: A `wbWorkbook` object
- **activeTab**: activeTab
- **autoFilterDateGrouping**: autoFilterDateGrouping
- **firstSheet**: firstSheet
- **minimized**: minimized
- **showHorizontalScroll**: showHorizontalScroll
- **showSheetTabs**: showSheetTabs
- **showVerticalScroll**: showVerticalScroll
- **tabRatio**: tabRatio
- **visibility**: visibility
- **windowHeight**: windowHeight
- **windowWidth**: windowWidth
- **xWindow**: xWindow
- **yWindow**: yWindow
**wb_set_col_widths**

**Value**

The `wbWorkbook` object

---

**wb_set_col_widths**  
*Set worksheet column widths*

---

**Description**

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

**Usage**

```r
wb_set_col_widths(
    wb,
    sheet = current_sheet(),
    cols,
    widths = 8.43,
    hidden = FALSE
)
```

**Arguments**

- **wb**: A `wbWorkbook` object
- **sheet**: A name or index of a worksheet
- **cols**: Indices of cols to set width
- **widths**: width 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. The default width is 8.43. Though there is no specific default width for Excel, it depends on Excel version, operating system and DPI settings used. Setting it to specific value also is no guarantee that the output will be of the selected width.
- **hidden**: Logical vector. If TRUE the column is hidden.

**Details**

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

- `options("openxlsx2.minWidth" = 3)`
- `options("openxlsx2.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 `wb_group_cols`; changing one will update the other.
See Also

wb_remove_col_widths()

Other workbook wrappers: wb_add_chartsheet(), wb_add_data_table(), wb_add_data(), wb_add_formula(), wb_add_worksheet(), wb_clone_worksheet(), wb_creators, wb_freeze_pane(), wb_get_base_font(), wb_save(), wb_set_last_modified_by(), wb_set_row_heights(), wb_workbook(), workbook_grouping, ws_cell_merge

Examples

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

## Add a worksheet
wb$add_worksheet("Sheet 1")

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

## auto columns
wb$add_worksheet("Sheet 2")
wb$add_data(sheet = 2, x = iris)
wb$set_col_widths(sheet = 2, cols = 1:5, widths = "auto")
```

---

**wb_set_header_footer**  
*Set document headers and footers*

**Description**

Set document headers and footers

**Usage**

```r
wb_set_header_footer(
  wb,
  sheet = current_sheet(),
  header = NULL,
  footer = NULL,
  evenHeader = NULL,
  evenFooter = NULL,
  firstHeader = NULL,
  firstFooter = NULL
)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wb</td>
<td>A workbook object</td>
</tr>
<tr>
<td>sheet</td>
<td>A name or index of a worksheet</td>
</tr>
</tbody>
</table>
document header. Character vector of length 3 corresponding to positions left, center, right. Use NA to skip a position.

document footer. Character vector of length 3 corresponding to positions left, center, right. Use NA to skip a position.

document header for even pages.
document footer for even pages.
document header for first page only.
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

See Also

`wb_add_worksheet()` to set headers and footers when adding a worksheet

Examples

```r
wb <- wb_workbook()

wb$add_worksheet("S1")
wb$add_worksheet("S2")
wb$add_worksheet("S3")
wb$add_worksheet("S4")

wb$add_data(1, 1:400)
wb$add_data(2, 1:400)
wb$add_data(3, 3:400)
wb$add_data(4, 3:400)

wb$set_header_footer(
  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")
)
```
Add another author to the meta data of the file.

**Description**

Just a wrapper of `wb$set_last_modified_by()`

**Usage**

```
wb_set_last_modified_by(wb, LastModifiedBy)
```

**Arguments**

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

**See Also**

Other workbook wrappers: `wb_add_chartsheet()`, `wb_add_data_table()`, `wb_add_data()`, `wb_add_formula()`, `wb_add_worksheet()`, `wb_clone_worksheet()`, `wb_creators`, `wb_freeze-pane()`, `wb_get_base_font()`, `wb_save()`, `wb_set_col_widths()`, `wb_set_row_heights()`, `wb_workbook()`, `workbook_grouping`, `ws_cell_merge`

**Examples**

```r
wb <- wb_workbook()
wbs_set_last_modified_by(wb, "test")
```
wb_set_row_heights

Set worksheet row heights

Description
Set worksheet row heights

Usage
wb_set_row_heights(
  wb, 
  sheet = current_sheet(), 
  rows, 
  heights = NULL, 
  hidden = FALSE 
)

Arguments
wb          A wbWorkbook 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.
hidden      Option to hide rows.

See Also
wb_remove_row_heights()

Other workbook wrappers: wb_add_chartsheet(), wb_add_data_table(), wb_add_data(), wb_add_formula(), 
wb_add_worksheet(), wb_clone_worksheet(), wb_creators, wb_freeze_pane(), wb_get_base_font(), 
wb_save(), wb_set_col_widths(), wb_set_last_modified_by(), wb_workbook(), workbook_grouping, 
ws_cell_merge

Examples
## Create a new workbook
wb <- wb_workbook()

## Add a worksheet
wb$add_worksheet("Sheet 1")

## set row heights
wb <- wb_set_row_heights(
  wb, 1,
  rows = c(1, 4, 22, 2, 19),
  heights = c(24, 28, 32, 42, 33)
)
## overwrite row 1 height
wb <- wb_set_row_heights(wb, 1, rows = 1, heights = 40)

---

### wb_set_sheet_names

Set worksheet names for a workbook

**Description**

Sets the worksheet names for a `wbWorkbook` object

**Usage**

```r
wb_set_sheet_names(wb, old = NULL, new)
```

**Arguments**

- `wb` A `wbWorkbook` object
- `old` The name (or index) of the old sheet name. If `NULL` will assume all worksheets are to be renamed.
- `new` The name of the new sheet

**Value**

The `wbWorkbook` object

---

### wb_to_df

Create Dataframe from Workbook

**Description**

Simple function to create a dataframe from a workbook. Simple as in simply written down and not optimized etc. The goal was to have something working.

**Usage**

```r
wb_to_df(
  xlsxFile, # required
  sheet, # required
  startRow = 1,
  startCol = NULL,
  rowNames = FALSE,
  colNames = TRUE,
  skipEmptyRows = FALSE,
  skipEmptyCols = FALSE,
)```


```r
wb_to_df = function(xlsxFile, sheet = NULL, startRow = 1, startCol = 1, 
                    rowNames = FALSE, colNames = FALSE, skipEmptyRows = FALSE, 
                    skipEmptyCols = FALSE, rows = NULL, cols = NULL, 
                    detectDates = TRUE, 
                    na.strings = "#N/A", na.numbers = NA, 
                    fillMergedCells = FALSE, 
                    dims = NULL, 
                    showFormula = FALSE, convert = TRUE, 
                    types = c(0:character, 1:numeric, 2:date), definedName = NULL, 
                    named_region = NULL)
}
```

**Arguments**

- `xlsxFile` : An xlsx file, Workbook object or URL to xlsx file.
- `sheet` : Either sheet name or index. When missing the first sheet in the workbook is selected.
- `startRow` : first row to begin looking for data.
- `startCol` : first column to begin looking for data.
- `rowNames` : If TRUE, the first col of data will be used as row names.
- `colNames` : If TRUE, the first row of data will be used as column names.
- `skipEmptyRows` : If TRUE, empty rows are skipped.
- `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.
- `detectDates` : If TRUE, attempt to recognize dates and perform conversion.
- `na.strings` : A character vector of strings which are to be interpreted as NA. Blank cells will be returned as NA.
- `na.numbers` : A numeric vector of digits 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.
- `dims` : Character string of type "A1:B2" as optional dimensions to be imported.
- `showFormula` : If TRUE, the underlying Excel formulas are shown.
- `convert` : If TRUE, a conversion to dates and numerics is attempted.
- `types` : A named numeric indicating, the type of the data. 0: character, 1: numeric, 2: date. Names must match the created
- `definedName` : (deprecated) Character string with a definedName. If no sheet is selected, the first appearance will be selected.
- `named_region` : Character string with a named_region (defined name or table). If no sheet is selected, the first appearance will be selected.
### Examples

```r
# numerics, dates, missings, bool and string
xlsxFile <- system.file("extdata", "readTest.xlsx", package = "openxlsx2")
w1 <- wb_load(xlsxFile)

# import workbook
wb_to_df(w1)

# do not convert first row to colNames
wb_to_df(w1, colNames = FALSE)

# do not try to identify dates in the data
wb_to_df(w1, detectDates = FALSE)

# return the underlying Excel formula instead of their values
wb_to_df(w1, showFormula = TRUE)

# read dimension without colNames
wb_to_df(w1, dims = "A2:C5", colNames = FALSE)

# read selected cols
wb_to_df(w1, cols = c(1:2, 7))

# read selected rows
wb_to_df(w1, rows = c(1, 4, 6))

# convert characters to numerics and date (logical too?)
wb_to_df(w1, convert = FALSE)

# erase empty Rows from dataset
wb_to_df(w1, sheet = 3, skipEmptyRows = TRUE)

# erase emptyCols from dataset
wb_to_df(w1, skipEmptyCols = TRUE)

# convert first row to rownames
wb_to_df(w1, sheet = 3, dims = "C6:G9", rowNames = TRUE)

# define type of the data.frame
wb_to_df(w1, cols = c(1, 4), types = c("Var1" = 0, "Var3" = 1))

# start in row 5
wb_to_df(w1, startRow = 5, colNames = FALSE)

# na string
wb_to_df(w1, na.strings = "")

# read_xlsx(w1)
```
# inlinestr
xlsxFile <- system.file("extdata", "inline_str.xlsx", package = "openxlsx2")
wb2 <- wb_load(xlsxFile)

# read dataset with inlinestr
wb_to_df(wb2)
wb_to_xlsx(wb2)

###########################################################################
# named_region // namedRegion
xlsxFile <- system.file("extdata", "namedRegions3.xlsx", package = "openxlsx2")
wb3 <- wb_load(xlsxFile)

# read dataset with named_region (returns global first)
wb_to_df(wb3, named_region = "MyRange", colNames = FALSE)

# read named_region from sheet
wb_to_df(wb3, named_region = "MyRange", sheet = 4, colNames = FALSE)

---

**wb_workbook**

Create a new Workbook object

**Description**

Create a new Workbook object

**Usage**

```r
wb_workbook(
    creator = NULL,
    title = NULL,
    subject = NULL,
    category = NULL,
    datetimeCreated = Sys.time()
)
```

**Arguments**

- `creator` Creator of the workbook (your name). Defaults to login username
- `title` Workbook properties title
- `subject` Workbook properties subject
- `category` Workbook properties category
- `datetimeCreated` The time of the workbook is created

**Value**

A `wbWorkbook` object
See Also

Other workbook wrappers: `wb_add_chartsheet()`, `wb_add_data_table()`, `wb_add_data()`, `wb_add_formula()`, `wb_add_worksheet()`, `wb_clone_worksheet()`, `wbCreators`, `wb_freeze-pane()`, `wb_get_base_font()`, `wb_save()`, `wb_set_col_widths()`, `wb_set_row_heights()`, `workbook_grouping`, `ws_cell_merge`

Examples

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

## Set Workbook properties
wb <- wb_workbook(
  creator = "Me",
  title = "Expense Report",
  subject = "Expense Report - 2022 Q1",
  category = "sales"
)
```

---

workbook_grouping  

Group Rows and Columns

Description

Group a selection of rows or cols

Usage

```r
wb_group_cols(
  wb,
  sheet = current_sheet(),
  cols,
  collapsed = FALSE,
  levels = NULL
)

wb_ungroup_cols(wb, sheet = current_sheet(), cols)

wb_group_rows(
  wb,
  sheet = current_sheet(),
  rows,
  collapsed = FALSE,
  levels = NULL
)

wb_ungroup_rows(wb, sheet = current_sheet(), rows)
```
Arguments

wb A `wbWorkbook` object

sheet A name or index of a worksheet

collapsed If TRUE the grouped columns are collapsed

levels levels

rows, cols Indices of rows and columns to group

Details

If row was previously hidden, it will now be shown

See Also

Other workbook wrappers: `wb_add_chartsheet()`, `wb_add_data_table()`, `wb_add_data()`, `wb_add_formula()`, `wb_add_worksheet()`, `wb_clone_worksheet()`, `wb_creators()`, `wb_freeze_pane()`, `wb_get_base_font()`, `wb_save()`, `wb_set_col_widths()`, `wb_set_last_modified_by()`, `wb_set_row_heights()`, `wb_workbook()`, `ws_cell_merge`

Examples

```r
# create matrix
t1 <- AirPassengers
t2 <- do.call(cbind, split(t1, cycle(t1)))
dimnames(t2) <- dimnames(.preformat.ts(t1))

wb <- wb_workbook()
wb$add_worksheet("AirPass")
wb$add_data("AirPass", t2, rowNames = TRUE)

# groups will always end on/show the last row. in the example 1950, 1955, and 1960
wb <- wb_group_rows(wb, "AirPass", 2:3, collapsed = TRUE) # group years < 1950
wb <- wb_group_rows(wb, "AirPass", 4:8, collapsed = TRUE) # group years 1951-1955
wb <- wb_group_rows(wb, "AirPass", 9:13) # group years 1956-1960

wb$createCols("AirPass", 13)

wb <- wb_group_cols(wb, "AirPass", 2:4, collapsed = TRUE)
wb <- wb_group_cols(wb, "AirPass", 5:7, collapsed = TRUE)
wb <- wb_group_cols(wb, "AirPass", 8:10, collapsed = TRUE)
wb <- wb_group_cols(wb, "AirPass", 11:13)
```
write_data2

dummy function to write data

Description

dummy function to write data

Usage

write_data2(
  wb,
  sheet,
  data,
  name = NULL,
  colNames = TRUE,
  rowNames = FALSE,
  startRow = 1,
  startCol = 1,
  applyCellStyle = TRUE,
  removeCellStyle = FALSE,
  na.strings,
  data_table = FALSE
)

Arguments

wb             workbook
sheet          sheet
data            data to export
name            If not NULL, a named region is defined.
colNames        include colnames?
rowNames        include rownames?
startRow        row to place it
startCol        col to place it
applyCellStyle  apply styles when writing on the sheet
removeCellStyle keep the cell style?
na.strings      optional na.strings argument. if missing #N/A is used. If NULL no cell value is written, if character or numeric this is written (even if NA is part of numeric data)
data_table      logical. if TRUE and rowNames = TRUE, do not write the cell containing "_rowNames_"
Details

The string "_openxlsx_NA" is reserved for openxlsx2. If the data frame contains this string, the output will be broken.

Examples

# create a workbook and add some sheets
wb <- wb_workbook()

wb$add_worksheet("sheet1")
write_data2(wb, "sheet1", mtcars, colNames = TRUE, rowNames = TRUE)

wb$add_worksheet("sheet2")
write_data2(wb, "sheet2", cars, colNames = FALSE)

wb$add_worksheet("sheet3")
write_data2(wb, "sheet3", letters)

wb$add_worksheet("sheet4")
write_data2(wb, "sheet4", as.data.frame(Titanic), startRow = 2, startCol = 2)

write_datatable

Write to a worksheet as an Excel table

Description

Write to a worksheet and format as an Excel table

Usage

write_datatable(
  wb,
  sheet,
  x,
  startCol = 1,
  startRow = 1,
  dims = rowcol_to_dims(startRow, startCol),
  xy = NULL,
  colNames = TRUE,
  rowNames = FALSE,
  tableStyle = "TableStyleLight9",
  tableName = NULL,
  withFilter = TRUE,
  sep = ",";
  firstColumn = FALSE,
  lastColumn = FALSE,
  bandedRows = TRUE,
write_datatable

bandedCols = FALSE,
applyCellStyle = TRUE,
removeCellStyle = FALSE,
na.strings
)

Arguments

wb A Workbook object containing a worksheet.
sheet The worksheet to write to. Can be the worksheet index or name.
x A data frame.
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.
withFilter If TRUE, columns with have filters in the first row.
sep Only applies to list columns. The separator used to collapse list columns to a character vector e.g. sapply(x$list_column, paste, collapse = sep).

The below options correspond to Excel table options:

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

Table Style Options

firstColumn logical. If TRUE, the first column is bold
lastColumn logical. If TRUE, the last column is bold
bandedRows logical. If TRUE, rows are colour banded
bandedCols logical. If TRUE, the columns are colour banded
applyCellStyle apply styles when writing on the sheet
removeCellStyle if writing into existing cells, should the cell style be removed?
nna.strings optional na.strings argument. if missing #N/A is used. If NULL no cell value is written, if character or numeric this is written (even if NA is part of numeric data)
Details

Columns of x with class Date.POSIXt, currency, accounting, hyperlink, percentage are automatically styled as dates, currency, accounting, hyperlinks, percentages respectively. The string "_openxlsx_NA" is reserved for openxlsx2. If the data frame contains this string, the output will be broken.

See Also

write_data()
write_worksheet()
wb_remove_tables()
wb_get_tables()

Examples

## see package vignettes for further examples.

### Create Workbook object and add worksheets

wb <- wb_workbook()
wb$add_worksheet("S1")
wb$add_worksheet("S2")
wb$add_worksheet("S3")

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

wb$add_data_table("S1", x = iris)

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

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

### openxlsx will apply default Excel styling for these classes
class(df$Cash) <- c(class(df$Cash), "currency")
class(df$Cash2) <- c(class(df$Cash2), "accounting")
class(df$hLink) <- "hyperlink"
class(df$Percentage) <- c(class(df$Percentage), "percentage")
class(df$TinyNumbers) <- c(class(df$TinyNumbers), "scientific")
wb$add_data_table("S3", x = df, startRow = 4, rowNames = TRUE, tableStyle = "TableStyleMedium9")

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

write_datatable(wb,
    sheet = 1,
    x = iris,
    startCol = 7,
    withFilter = FALSE,
    firstColumn = TRUE,
    lastColumn = TRUE,
    bandedRows = TRUE,
    bandedCols = TRUE)

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

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

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

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

write_file write xml file

Description

brings the added benefit of xml checking
write_formula

Usage

write_file(head = "", body = "", tail = "", fl = "", escapes = FALSE)

Arguments

head        head part of xml
body        body part of xml
tail        tail part of xml
fl           file name with full path
escapes     bool if characters like "&" should be escaped. The default is no escape, assuming that xml to export is already valid.

write_formula Write a character vector as an Excel Formula

Description

Write a character vector containing Excel formula to a worksheet.

Usage

write_formula(
  wb,
  sheet,
  x,
  startCol = 1,
  startRow = 1,
  dims = rowcol_to_dims(startRow, startCol),
  array = FALSE,
  xy = NULL,
  applyCellStyle = TRUE,
  removeCellStyle = FALSE
)

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
An alternative to specifying `startCol` and `startRow` individually. A vector of the form `c(startCol, startRow)`.

applyCellStyle  
apply styles when writing on the sheet

removeCellStyle  
if writing into existing cells, should the cell style be removed?

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)`
- ...

See Also

- `write_data()`

Examples

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

wb <- wb_workbook()
wbs$add_worksheet("Sheet 1")
wbs$add_data("Sheet 1", x = iris)

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

## 1. - As a character vector using write_formula

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

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

df <- data.frame(
  x = 1:3,
  y = 1:3,
  z = paste(paste0("A", 1:3 + 1L), paste0("B", 1:3 + 1L), sep = " + "),
  z2 = sprintf("ADDRESS(1,\%s", 1:3),
    stringsAsFactors = FALSE)
)
class(df$z) <- c(class(df$z), "formula")
```
```r
class(df$z2) <- c(class(df$z2), "formula")
wb$add_worksheet("Sheet 2")
wb$add_data(sheet = 2, x = df)

## 3. - As a vector with class "formula" using write_data
v2 <- c("SUM(A2:A4)", "AVERAGE(B2:B4)", "MEDIAN(C2:C4)"
class(v2) <- c(class(v2), "formula")
wb$add_data(sheet = 2, x = v2, startCol = 10, startRow = 2)

## 4. - Writing internal hyperlinks
wb <- wb_workbook()
wb$add_worksheet("Sheet1")
wb$add_worksheet("Sheet2")
write_formula(wb, "Sheet1", x = '=HYPERLINK("#Sheet2!B3", "Text to Display - Link to Sheet2")')

## 5. - Writing array formulas
set.seed(123)
df <- data.frame(C = rnorm(10), D = rnorm(10))
wb <- wb_workbook()
wb <- wb_add_worksheet(wb, "df")
wb$add_data("df", df, startCol = "C")
write_formula(wb, "df", startCol = "E", startRow = 2,
x = "SUM(C2:C11*D2:D11)",
array = TRUE)
```

---

**write_xlsx**  
write data to an xlsx file

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

**Usage**
write_xlsx(x, file, asTable = FALSE, ...)

Arguments

- **x**: object or a list of objects that can be handled by `write_data()` to write to file
- **file**: xlsx file name
- **asTable**: write using `write_datatable` as opposed to `write_data`
- **...**: optional parameters to pass to functions:
  - `wb_workbook()`
  - `wb_add_worksheet()`
  - `wb_add_data()`
  - `wb_freeze_pane`
  - `wb_save()`
  - see details.

Details

Optional parameters are:

**wb_workbook Parameters**

- **creator**: A string specifying the workbook author

**wb_add_worksheet() 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.

**write_data/write_datatable 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.
- **na.string**: If not NULL, 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)

**wb_save Parameters**

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

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

**Value**

A workbook object

**See Also**

`wb_add_worksheet()`, `write_data()`

**Examples**

```r
## write to working directory
write_xlsx(iris, file = temp_xlsx(), colNames = TRUE)

write_xlsx(iris,
    file = temp_xlsx(),
    colNames = TRUE
)

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

## different sheets can be given different parameters
write_xlsx(l, temp_xlsx(),
    startCol = c(1, 2, 3), startRow = 2,
    asTable = c(TRUE, TRUE, FALSE), withFilter = c(TRUE, FALSE, FALSE)
)

# specify column widths for multiple sheets
write_xlsx(l, temp_xlsx(), colWidths = 20)
write_xlsx(l, temp_xlsx(), colWidths = list(100, 200, 300))
write_xlsx(l, temp_xlsx(), colWidths = list(rep(10, 5), rep(8, 11), rep(5, 5)))
```

---

**ws_cell_merge**

**Worksheet cell merging**

**Description**

Merge cells within a worksheet
Usage

```
wb_merge_cells(wb, sheet = current_sheet(), rows = NULL, cols = NULL)
wb_unmerge_cells(wb, sheet = current_sheet(), rows = NULL, cols = NULL)
```

Arguments

- **wb**: A workbook object
- **sheet**: A name or index of a worksheet
- **cols, rows**: Column and row specifications to merge on. Note: min() and max() of each vector are provided for specs. Skipping rows or columns is not recognized.

Details

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

See Also

Other workbook wrappers: `wb_add_chartsheet()`, `wb_add_data_table()`, `wb_add_data()`, `wb_add_formula()`, `wb_add_worksheet()`, `wb_clone_worksheet()`, `wb_creators()`, `wb_freeze_pane()`, `wb_get_base_font()`, `wb_save()`, `wb_set_col_widths()`, `wb_set_last_modified_by()`, `wb_set_row_heights()`, `wb_workbook()`, `workbook_grouping`

Examples

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

# Add a worksheets
wb$add_worksheet("Sheet 1")
wb$add_worksheet("Sheet 2")

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

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

# Intersecting merges
wb <- wb_merge_cells(wb, 2, cols = 1:10, rows = 1)
wb <- wb_merge_cells(wb, 2, cols = 5:10, rows = 2)
wb <- wb_merge_cells(wb, 2, cols = c(1, 10), rows = 12) # equivalent to 1:10
try(wb_merge_cells(wb, 2, cols = 1, rows = c(1,10))) # intersects existing merge

# remove merged cells
wb <- wb_unmerge_cells(wb, 2, cols = 1, rows = 1) # removes any intersecting merges
wb <- wb_merge_cells(wb, 2, cols = 1, rows = 1:10) # Now this works
```
ws_page_setup

Set page margins, orientation and print scaling

Description

Set page margins, orientation and print scaling

Usage

```r
wb_page_setup(
  wb,
  sheet = current_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)
Examples

```r
wb <- wb_workbook()
wb$add_worksheet("S1")
wb$add_worksheet("S2")
wb$add_data_table(1, x = iris[1:30, ])
wb$add_data_table(2, x = iris[1:30, ], xy = c("C", 5))

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

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

## print titles
wb$add_worksheet("print_title_rows")
wb$add_worksheet("print_title_cols")

wb$add_data("print_title_rows", rbind(iris, iris, iris, iris))
wb$add_data("print_title_cols", x = rbind(mtcars, mtcars, mtcars), rowNames = TRUE)

wb$page_setup(sheet = "print_title_rows", printTitleRows = 1) ## first row
wb$page_setup(sheet = "print_title_cols", printTitleCols = 1, printTitleRows = 1)
```

xl_open

Open a Microsoft Excel file (xls/xlsx) or an openxlsx2 wbWorkbook

Description

This function tries to open a Microsoft Excel (xls/xlsx) file or an openxlsx2 wbWorkbook with the proper application, in a portable manner.

In Windows it uses `base::shell.exec()` (Windows only function) to determine the appropriate program.

In 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('openxlsx2.excelApp')` is set to the app bin path), and if it finds anything, sets `options('openxlsx2.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
xl_open(x, file, interactive = NA)
```

## S3 method for class 'wbWorkbook'

`xl_open(x, file, interactive = NA)`

## Default S3 method:

`xl_open(x, file, interactive = NA)`
xml_add_child

Arguments

x  A path to the Excel (xls/xlsx) file or Workbook object.
file  Deprecated
interactive  If FALSE will throw a warning and not open the path. This can be manually set to TRUE, otherwise when NA (default) uses the value returned from base::interactive()

Examples

if (interactive()) {
  xlsxFile <- system.file("extdata", "readTest.xlsx", package = "openxlsx2")
  xl_open(xlsxFile)

  # (not yet saved) Workbook example
  wb <- wb_workbook()
  x <- mtcars[1:6, ]
  wb$add_worksheet("Cars")
  wb$add_data("Cars", x, startCol = 2, startRow = 3, rowNames = TRUE)
  xl_open(wb)
}

xml_add_child  append xml child to node

Description

append xml child to node

Usage

xml_add_child(xml_node, xml_child, level, pointer = FALSE, ...)

Arguments

xml_node  xml_node
xml_child  xml_child
level  optional level, if missing the first child is picked
pointer  pointer
...  additional arguments passed to read_xml()
Examples

```r
xml_node <- "<a><b/></a>"
xml_child <- "<c/>"

# add child to first level node
xml_add_child(xml_node, xml_child)

# add child to second level node as request
xml_node <- xml_add_child(xml_node, xml_child, level = c("b"))

# add child to third level node as request
xml_node <- xml_add_child(xml_node, "<d/>", level = c("b", "c"))
```

---

**xml_attr_mod**

**Adds or updates attribute(s) in existing xml node**

Description

Needs xml node and named character vector as input. Modifies the arguments of each first child found in the xml node and adds or updates the attribute vector.

Usage

```r
xml_attr_mod(xml_content, xml_attributes, escapes = FALSE, declaration = FALSE)
```

Arguments

- `xml_content`: some valid xml_node
- `xml_attributes`: R vector of named attributes
- `escapes`: bool if escapes should be used
- `declaration`: bool if declaration should be imported

Details

If a named attribute in `xml_attributes` is "" remove the attribute from the node. If `xml_attributes` contains a named entry found in the xml node, it is updated else it is added as attribute.

Examples

```r
# add single node
xml_node <- "<a foo="bar">openxlsx2</a><b />
xml_attr <- c(qux = "quux")
# "<a foo="bar" qux="quux">openxlsx2</a><b qux="quux"/>
xml_attr_mod(xml_node, xml_attr)

# update node and add node
xml_node <- "<a foo="bar">openxlsx2</a><b />
```
xml_node_create

create xml_node from R objects

description
takes xml_name, xml_children and xml_attributes to create a new xml_node.

usage
xml_node_create(
  xml_name,
  xml_children = NULL,
  xml_attributes = NULL,
  escapes = FALSE,
  declaration = FALSE
)

arguments
xml_name        the name of the new xml_node
xml_children    character vector children attached to the xml_node
xml_attributes  named character vector of attributes for the xml_node
escapes         bool if escapes should be used
declaration     bool if declaration should be imported

details
if xml_children or xml_attributes should be empty, use NULL

examples
xml_name <- "a"
# "<a/>"
xml_node_create(xml_name)

xml_child <- "openxlsx"
# "<a>openxlsx</a>"
xml_node_create(xml_name, xml_children = xml_child)
xml_attr <- c(foo = "baz", qux = "quux")
# "<a foo="baz" qux="quux"/>"
xml_node_create(xml_name, xml_attributes = xml_attr)
# "<a foo="baz" qux="quux">openxlsx</a>
xml_node_create(xml_name, xml_children = xml_child, xml_attributes = xml_attr)

xml_rm_child remove xml child to node

Description

remove xml child to node

Usage

xml_rm_child(xml_node, xml_child, level, which = 0, pointer = FALSE, ...)

Arguments

xml_node xml_node
xml_child xml_child
level optional level, if missing the first child is picked
which optional index which node to remove, if multiple are available. Default disabled
all will be removed
pointer pointer
... additional arguments passed to read_xml()

Examples

xml_node <- "<a><b><c><d/></c></b><c/></a>"
xml_child <- "c"
xml_rm_child(xml_node, xml_child)
xml_rm_child(xml_node, xml_child, level = c("b"))
xml_rm_child(xml_node, "d", level = c("b", "c"))
Index

* styles
  - wb_add_border, 90  
  - wb_add_cell_style, 91  
  - wb_add_fill, 105  
  - wb_add_font, 107  
  - wb_add_numfmt, 112  
  - wb_clone_sheet_style, 119
* workbook wrappers
  - wb_add_chartsheet, 93  
  - wb_add_data, 97  
  - wb_add_data_table, 100  
  - wb_add_formula, 109  
  - wb_add_worksheet, 117  
  - wb_clone_worksheet, 120  
  - wb_creators, 121  
  - wb_freeze_pane, 123  
  - wb_get_base_font, 124  
  - wb_save, 139  
  - wb_set_col_widths, 141  
  - wb_set_last_modified_by, 144  
  - wb_set_row_heights, 145  
  - wb_workbook, 149  
  - workbook_grouping, 150  
  - ws_cell_merge, 161

as_xml, 4
base::interactive(), 61, 167

cell_style, 5  
clean_worksheet_name, 7  
cleanup, 6  
cloneSheetStyle, 7  
col2int, 8  
convert_date, 9  
convert_date(), 24, 25  
convert_datetime, 9  
convertToExcelDate, 8  
create_border, 10  
create_border(), 90, 116  
create_cell_style, 11  
create_cell_style(), 116  
create_comment, 13  
create_dxf_style, 15  
create_dxf_style(), 116  
create_fill, 16  
create_fill(), 116  
create_font, 17  
create_font(), 116  
create_hyperlink, 18  
create_numfmt, 20  
create_numfmt(), 116  
create_sparklines, 21  
create_sparklines(), 115

dataframe_to_dims, 22  
delete_data(cleanup), 6  
dims_to_dataframe, 23

get_cell_refs, 23  
get_cell_style, 24  
get_date_origin, 24  
get_named_regions(NamedRegions), 26  
get_named_regions(), 35, 135  
grDevices::dev.copy(), 114  
guess_col_type, 25

int2col, 26  
named_region, 27  
NamedRegions, 26  
numfmt_is_date, 29  
numfmt_is_posix, 30

openxlsx2, 30

print.pugi_xml, 31  
pugixml, 32

read_sheet_names, 33  
read_xlsx, 34

171
read_xlsx(), 31, 135
read_xml, 36
remove_comment(create_comment), 13
select_active_sheet, 37
set_cell_style, 38
sheet_visibility, 39
style_is_date, 40
style_is_posix, 41
style_mgr, 41
styles_on_sheet, 40
temp_xlsx, 46
wb_add_border, 90, 93, 105, 108, 113, 119
wb_add_cell_style, 90, 91, 105, 108, 113, 119
wb_add_chart_xml, 94
wb_add_chartsheet, 93, 99, 102, 110, 118, 120, 122, 124, 139, 142, 144, 145, 150, 151, 162
wb_add_comment(create_comment), 13
wb_add_conditional_formatting, 94
wb_add_creators(wb_creators), 121
wb_add_data, 94, 97, 102, 110, 118, 120, 122, 124, 139, 142, 144, 145, 150, 151, 162
wb_add_data(), 160
wb_add_data_table, 94, 99, 100, 110, 118, 120, 122, 124, 139, 142, 144, 145, 150, 151, 162
wb_add_data_validation, 102
wb_add_draw, 104
wb_add_fill, 90, 93, 105, 108, 113, 119
wb_add_filter, 106
wb_add_filter(), 106
wb_add_font, 90, 93, 105, 107, 113, 119
wb_add_formula, 94, 99, 102, 109, 118, 120, 122, 124, 139, 142, 144, 145, 150, 151, 162
wb_add_image, 110
wb_add_image(), 114
wb_add_mschart, 111
wb_add_mschart(), 94, 123
wb_add_named_region(named_region), 27
wb_add_named_region(), 26
wb_add_numfmt, 90, 93, 105, 108, 112, 119
wb_add_page_break, 113
wb_add_plot, 114
wb_add_plot(), 111
wb_add_sparklines, 115
wb_add_style, 116
wb_add_style(), 16
wb_add_worksheet, 94, 99, 102, 110, 117, 120, 122, 124, 139, 142, 144, 145, 150, 151, 162
wb_add_worksheet(), 113, 143, 155, 160, 161
wb_clean_sheet (cleanup), 6
wb_clone_sheet_style, 90, 93, 105, 108, 113, 119
wb_clone_worksheet, 94, 99, 102, 110, 118, 120, 122, 124, 139, 142, 144, 145, 150, 151, 162
wb_colour, 121
wb_conditional_formatting
(wb_add_conditional_formatting), 94
wb_creators, 94, 99, 102, 110, 118, 120, 121, 124, 139, 142, 144, 145, 150, 151, 162
wb_data, 122
wb_data(), 111
wb_freeze_pane, 94, 99, 102, 110, 118, 120, 122, 123, 124, 139, 142, 144, 145, 150, 151, 162
wb_get_active_sheet
(select_active_sheet), 37
wb_get_base_font, 94, 99, 102, 110, 118, 120, 122, 124, 139, 142, 144, 145, 150, 151, 162
wb_get_cell_style(cell_style), 5
wb_get_creators(wb_creators), 121
wb_get_order(wb_order), 130
wb_get_selected(select_active_sheet), 37
wb_get_sheet_name, 125
wb_get_sheet_names, 125
wb_get_sheet_visibility
(sheet_visibility), 39
wb_get_tables, 126
wb_get_tables(), 137, 155
wb_get_worksheet, 126
wb_grid_lines, 127
wb_group_cols, 141
wb_group_cols(workbook_grouping), 150
wb_group_rows(workbook_grouping), 150
xml_value (pugixml), 32