Package 'openxlsx2'

July 7, 2024

Type Package

Title Read, Write and Edit 'xlsx' Files

Version 1.8

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://janmarvin.github.io/openxlsx2/,
https://github.com/JanMarvin/openxlsx2

BugReports https://github.com/JanMarvin/openxlsx2/issues

Depends R (>= 3.5.0)

Imports R6, Rcpp, grDevices, magrittr, stringi, utils, zip

LinkingTo Rcpp

Suggests ggplot2, knitr, mschart (>= 0.4), rmarkdown, rvg, testthat
(>= 3.0.0), waldo

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.2

Config/testthat/edition 3

Config/testthat/parallel true

NeedsCompilation yes

Author Jordan Mark Barbone [aut] (<https://orcid.org/0000-0001-9788-3628>),
Jan Marvin Garbuszus [aut, cre],
Oliver Roy [ctb],
openxlsx authors [cph] (openxlsx package),
Arseny Kapoulkine [ctb, cph] (Author of included pugixml code)

Maintainer Jan Marvin Garbuszus <jan.garbuszus@ruhr-uni-bochum.de>

Repository CRAN

Date/Publication 2024-07-07 10:30:28 UTC

Contents

`active_sheet-wb` *Modify the state of active and selected sheets in a workbook*

Description

Get and set table of sheets and their state as selected and active in a workbook

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

```
wb <- wb_load(file = system.file("extdata", "openxlsx2_example.xlsx", package = "openxlsx2"))
# testing is the selected sheet
wb_get_selected(wb)
# change the selected sheet to Sheet2
wb <- wb_set_selected(wb, "Sheet2")
```

```
# get the active sheet
wb_get_active_sheet(wb)
# change the selected sheet to Sheet2
wb <- wb_set_active_sheet(wb, sheet = "Sheet2")
```

---

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", "openxlsx2_example.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)
```

base_font-wb *Set the default font in a workbook*

Description

Modify / get the default font for the workbook. This will alter the latin major and minor font in the workbooks theme.

Usage

```
wb_set_base_font(
  wb,
  font_size = 11,
  font_color = wb_color(theme = "1"),
  font_name = "Aptos Narrow",
  ...
)

wb_get_base_font(wb)
```

Arguments

wb	A workbook object
font_size	Font size
font_color	Font color
font_name	Name of a font
...	Additional arguments

Details

The font name is not validated in anyway. Spreadsheet software replaces unknown font names with system defaults.

The default base font is Aptos Narrow, black, size 11. If font_name differs from the name in wb_get_base_font(), the theme is updated to use the newly selected font name.

See Also

Other workbook styling functions: wb_add_dxfs_style(), wb_add_style(), wb_base_colors

Other workbook wrappers: col_widths-wb, creators-wb, grouping-wb, row_heights-wb, wb_add_chartsheet(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_worksheet(), wb_base_colors, wb_clone_worksheet(), wb_copy_cells(), wb_freeze_pane(), wb_merge_cells(), wb_save(), wb_set_last_modified_by(), wb_workbook()

Examples

```
## create a workbook
wb <- wb_workbook(theme = "Office 2013 - 2022 Theme")
wb$add_worksheet("S1")
## modify base font to size 10 Aptos Narrow in red
wb$set_base_font(font_size = 10, font_color = wb_color("red"), font_name = "Aptos Narrow")

wb$add_data(x = iris)

## font color does not affect tables
wb$add_data_table(x = iris, dims = wb_dims(from_col = 10))

## get the base font
wb_get_base_font(wb)
```

`clean_worksheet_name` *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

col2int *Convert Excel column to integer*

Description

Converts an Excel column label to an integer.

Usage

```
col2int(x)
```

Arguments

x	A character vector

Value

An integer column label (or NULL if x is NULL)

Examples

```
col2int(LETTERS)
```

col_widths-wb *Modify column widths of a worksheet*

Description

Remove / set worksheet column widths to specified width or "auto".

Usage

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

wb_remove_col_widths(wb, sheet = current_sheet(), cols)
```

Arguments

wb	A wbWorkbook object.
sheet	A name or index of a worksheet, a vector in the case of remove_
cols	Indices of cols to set/remove column widths.
widths	Width to set cols to specified column width or "auto" for automatic sizing. widths is recycled to the length of cols. openxlsx2 sets the default width is 8.43, as this is the standard in some spreadsheet software. See **Details** for general information on column widths.
hidden	Logical vector recycled to the length of cols. If TRUE, the columns are 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) 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.

NOTE: The default column width varies by spreadsheet software, operating system, and DPI settings used. Setting widths to specific value also is no guarantee that the output will have consistent column widths.

For automatic text wrapping of columns use wb_add_cell_style(wrap_text = TRUE)

See Also

Other workbook wrappers: base_font-wb, creators-wb, grouping-wb, row_heights-wb, wb_add_chartsheet(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_worksheet(), wb_base_colors, wb_clone_worksheet(), wb_copy_cells(), wb_freeze_pane(), wb_merge_cells(), wb_save(), wb_set_last_modified_by(), wb_workbook()

Other worksheet content functions: filter-wb, grouping-wb, named_region-wb, row_heights-wb, wb_add_conditional_formatting(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_thread(), wb_freeze_pane(), wb_merge_cells()

Examples

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

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

## set col widths
wb$set_col_widths(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")

## removing column widths
## Create a new workbook
wb <- wb_load(file = system.file("extdata", "openxlsx2_example.xlsx", package = "openxlsx2"))

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

`convert_date` *Convert from Excel date, datetime or hms number to R Date type*

Description

Convert from Excel date number to R Date type

Usage

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

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

convert_hms(x)
```

Arguments

`x`	A vector of integers
`origin`	date. Default value is for Windows Excel 2010
`...`	Arguments passed on to `base::as.Date.character`
	`format` a `character` string. If not specified when converting from a character representation, it will try `tryFormats` one by one on the first non-`NA` element, and give an error if none works. Otherwise, the processing is via `strptime()` whose help page describes available conversion specifications.
	`tryFormats` `character` vector of `format` strings to try if `format` is not specified.
	`optional` `logical` indicating to return `NA` (instead of signalling an error) if the format guessing does not succeed.

Details

Excel stores dates as number of days from some origin day

Value

A date, datetime, or hms.

See Also

wb_add_data()

Examples

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

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

# hms ---
## 12:13:14
x <- 0.50918982
convert_hms(x)
```

---

convert_to_excel_date *convert back to an Excel Date*

Description

convert back to an Excel Date

Usage

```
convert_to_excel_date(df, date1904 = FALSE)
```

Arguments

df	dataframe
date1904	take different origin

Examples

```
xlsxFile <- system.file("extdata", "openxlsx2_example.xlsx", package = "openxlsx2")
wb1 <- wb_load(xlsxFile)
df <- wb_to_df(wb1)
# conversion is done on dataframes only
convert_to_excel_date(df = df["Var5"], date1904 = FALSE)
```

`create_border` *Helper to create a border*

Description

Border styles can any of the following: "thin", "thick", "slantDashDot", "none", "mediumDashed", "mediumDashDot", "medium", "hair", "double", "dotted", "dashed", "dashedDotDot", "dashDot" Border colors can be created with `wb_color()`

Usage

```
create_border(
  diagonal_down = "",
  diagonal_up = "",
  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

`diagonal_down`	x
`diagonal_up`	x
`outline`	x
`bottom`	X
`bottom_color`, `diagonal_color`, `left_color`, `right_color`, `top_color`	a color created with `wb_color()`
`diagonal`	X
`end`	x,
`horizontal`	x
`left`	x

right	x
start	x
top	x
vertical	x
...	x

See Also

wb_add_border()

Other style creating functions: create_cell_style(), create_colors_xml(), create_dxfs_style(), create_fill(), create_font(), create_numfmt(), create_tablestyle()

---

create_cell_style *Helper to create a cell style*

Description

Helper to create a cell style

Usage

```
create_cell_style(
  border_id = "",
  fill_id = "",
  font_id = "",
  num_fmt_id = "",
  pivot_button = "",
  quote_prefix = "",
  xf_id = "",
  horizontal = "",
  indent = "",
  justify_last_line = "",
  reading_order = "",
  relative_indent = "",
  shrink_to_fit = "",
  text_rotation = "",
  vertical = "",
  wrap_text = "",
  ext_lst = "",
  hidden = "",
  locked = "",
  ...
)
```

Arguments

border_id	dummy
fill_id	dummy
font_id	dummy
num_fmt_id	a numFmt ID for a builtin style
pivot_button	dummy
quote_prefix	dummy
xf_id	dummy
horizontal	alignment can be "", "center", "right"
indent	dummy
justify_last_line	dummy
reading_order	dummy
relative_indent	dummy
shrink_to_fit	dummy
text_rotation	dummy
vertical	alignment can be "", "center", "right"
wrap_text	dummy
ext_lst	dummy
hidden	dummy
locked	dummy
...	reserved for additional arguments

Details

ID	numFmt
0	General
1	0
2	0.00
3	#,##0
4	#,##0.00
9	0%
10	0.00%
11	0.00E+00
12	# ?/?
13	# ??/??
14	mm-dd-yy
15	d-mmm-yy
16	d-mmm
17	mmm-yy

"18"	"h:mm AM/PM"
"19"	"h:mm:ss AM/PM"
"20"	"h:mm"
"21"	"h:mm:ss"
"22"	"m/d/yy h:mm"
"37"	"#,##0 ;(#,##0)"
"38"	"#,##0 ;[Red](#,##0)"
"39"	"#,##0.00;(#,##0.00)"
"40"	"#,##0.00;[Red](#,##0.00)"
"45"	"mm:ss"
"46"	"[h]:mm:ss"
"47"	"mmss.0"
"48"	"##0.0E+0"
"49"	"@"

See Also

`wb_add_cell_style()`

Other style creating functions: `create_border()`, `create_colors_xml()`, `create_dxfs_style()`, `create_fill()`, `create_font()`, `create_numfmt()`, `create_tablestyle()`

`create_colors_xml` *Create custom color xml schemes*

Description

Create custom color themes that can be used with `wb_set_base_colors()`. The color input will be checked with `wb_color()`, so it must be either a color R from `grDevices::colors()` or a hex value. Default values for the dark argument are: `black`, `white`, `darkblue` and `lightgray`. For the accent argument, the six inner values of `grDevices::palette()`. The link argument uses `blue` and `purple` by default for active and visited links.

Usage

```
create_colors_xml(name = "Base R", dark = NULL, accent = NULL, link = NULL)
```

Arguments

`name`	the color name
`dark`	four colors: dark, light, brighter dark, darker light
`accent`	six accent colors
`link`	two link colors: link and visited link

See Also

Other style creating functions: create_border(), create_cell_style(), create_dxfs_style(), create_fill(), create_font(), create_numfmt(), create_tablestyle()

Examples

```
colors <- create_colors_xml()
wb <- wb_workbook()$add_worksheet()$set_base_colors(xml = colors)
```

---

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,
  num_fmt = NULL,
  border = NULL,
  border_color = wb_color(getOption("openxlsx2.borderColor", "black")),
  border_style = getOption("openxlsx2.borderStyle", "thin"),
  bg_fill = NULL,
  fg_color = NULL,
  gradient_fill = 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 font_name is NULL, the workbook base_font is used. (Defaults to Calibri), see wb_get_base_font()
font_size	Font size. A numeric greater than 0. By default, the workbook base font size is used. (Defaults to 11)
font_color	Color of text in cell. A valid hex color beginning with "#" or one of colors(). If font_color is NULL, the workbook base font colors is used. (Defaults to black)

num_fmt	Cell formatting. Some custom openxml format
border	NULL or TRUE
border_color	"black"
border_style	"thin"
bg_fill	Cell background fill color.
fg_color	Cell foreground fill color.
gradient_fill	An xml string beginning with <gradientFill> ...
text_bold	bold
text_strike	strikeout
text_italic	italic
text_underline	underline 1, true, single or double
...	Additional arguments

Details

It is possible to override border_color and border_style with {left, right, top, bottom}_color, {left, right, top, bottom}_style.

Value

A dxfs style node

See Also

wb_add_style() wb_add_dxfs_style()

Other style creating functions: create_border(), create_cell_style(), create_colors_xml(), create_fill(), create_font(), create_numfmt(), create_tablestyle()

Examples

```
# do not apply anything
style1 <- create_dxfs_style()

# change font color and background color
style2 <- create_dxfs_style(
  font_color = wb_color(hex = "FF9C0006"),
  bgFill = wb_color(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 = "Aptos Narrow",
  font_size = 11,
  font_color = wb_color(hex = "FF9C0006"),
  bgFill = wb_color(hex = "FFFFC7CE")
)

## See package vignettes for further examples
```

create_fill *Create fill pattern*

Description

Create fill pattern

Usage

```
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
...	...

See Also

wb_add_fill()

Other style creating functions: create_border(), create_cell_style(), create_colors_xml(), create_dxfs_style(), create_font(), create_numfmt(), create_tablestyle()

create_font *Create font format*

Description

Create font format

Usage

```
create_font(
  b = "",
  charset = "",
  color = wb_color(hex = "FF000000"),
  condense = "",
  extend = "",
  family = "2",
  i = "",
  name = "Aptos Narrow",
  outline = "",
  scheme = "minor",
  shadow = "",
  strike = "",
  sz = "11",
  u = "",
  vert_align = "",
  ...
)
```

Arguments

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

See Also

wb_add_font()

Other style creating functions: create_border(), create_cell_style(), create_colors_xml(), create_dxfs_style(), create_fill(), create_numfmt(), create_tablestyle()

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 `wb_add_formula()`. Either link to external URLs or local files or straight to cells of local Excel sheets.

Note that for an external URL, only `file` and `text` should be supplied. You can supply `dims` to `wb_add_formula()` to control the location of the link.

Usage

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

Arguments

`sheet`	Name of a worksheet
`row`	integer row number for hyperlink to link to
`col`	column number of letter for hyperlink to link to
`text`	Display text
`file`	Hyperlink or Excel file name to point to. If `NULL`, hyperlink is internal.

Examples

```
wb <- wb_workbook()$
  add_worksheet("Sheet1")$add_worksheet("Sheet2")$add_worksheet("Sheet3")

## Internal Hyperlink - create hyperlink formula manually
x <- '=HYPERLINK(\"#Sheet2!B3\", "Text to Display - Link to Sheet2")'
wb$add_formula(sheet = "Sheet1", x = x, dims = "A1")

## Internal - No text to display using create_hyperlink() function
x <- create_hyperlink(sheet = "Sheet3", row = 1, col = 2)
wb$add_formula(sheet = "Sheet1", x = x, dims = "A2")

## Internal - Text to display
x <- create_hyperlink(sheet = "Sheet3", row = 1, col = 2,text = "Link to Sheet 3")
```

```
wb$add_formula(sheet = "Sheet1", x = x, dims = "A3")

## Link to file - No text to display
fl <- system.file("extdata", "openxlsx2_example.xlsx", package = "openxlsx2")
x <- create_hyperlink(sheet = "Sheet1", row = 3, col = 10, file = fl)
wb$add_formula(sheet = "Sheet1", x = x, dims = "A4")

## Link to file - Text to display
fl <- system.file("extdata", "openxlsx2_example.xlsx", package = "openxlsx2")
x <- create_hyperlink(sheet = "Sheet2", row = 3, col = 10, file = fl, text = "Link to File.")
wb$add_formula(sheet = "Sheet1", x = x, dims = "A5")

## Link to external file - Text to display
x <- '=HYPERLINK("[C:/Users]", "Link to an external file")'
wb$add_formula(sheet = "Sheet1", x = x, dims = "A6")

x <- create_hyperlink(text = "test.png", file = "D:/somepath/somepicture.png")
wb$add_formula(x = x, dims = "A7")


## Link to an URL.
x <- create_hyperlink(text = "openxlsx2 website", file = "https://janmarvin.github.io/openxlsx2/")

wb$add_formula(x = x, dims = "A8")
# if (interactive()) wb$open()
```

`create_numfmt` *Create number format*

Description

Create number format

Usage

```
create_numfmt(numFmtId, formatCode)
```

Arguments

`numFmtId`	an id, the list can be found in the **Details** of `create_cell_style()`
`formatCode`	a format code

See Also

`wb_add_numfmt()`

Other style creating functions: `create_border()`, `create_cell_style()`, `create_colors_xml()`, `create_dxfs_style()`, `create_fill()`, `create_font()`, `create_tablestyle()`

`create_sparklines` *Create sparklines object*

Description

Create a sparkline to be added a workbook with `wb_add_sparklines()`

Usage

```
create_sparklines(
  sheet = current_sheet(),
  dims,
  sqref,
  type = NULL,
  negative = NULL,
  display_empty_cells_as = "gap",
  markers = NULL,
  high = NULL,
  low = NULL,
  first = NULL,
  last = NULL,
  color_series = wb_color(hex = "FF376092"),
  color_negative = wb_color(hex = "FFD00000"),
  color_axis = wb_color(hex = "FFD00000"),
  color_markers = wb_color(hex = "FFD00000"),
  color_first = wb_color(hex = "FFD00000"),
  color_last = wb_color(hex = "FFD00000"),
  color_high = wb_color(hex = "FFD00000"),
  color_low = wb_color(hex = "FFD00000"),
  manual_max = NULL,
  manual_min = NULL,
  line_weight = NULL,
  date_axis = NULL,
  display_x_axis = NULL,
  display_hidden = NULL,
  min_axis_type = NULL,
  max_axis_type = NULL,
  right_to_left = NULL,
  ...
)
```

Arguments

`sheet`	sheet
`dims`	Cell range of cells used to create the sparklines
`sqref`	Cell range of the destination of the sparklines.

type	Either NULL, stacked or column
negative	negative
display_empty_cells_as	Either gap, span or zero
markers	markers add marker to line
high	highlight highest value
low	highlight lowest value
first	highlight first value
last	highlight last value
color_series	colorSeries
color_negative	colorNegative
color_axis	colorAxis
color_markers	colorMarkers
color_first	colorFirst
color_last	colorLast
color_high	colorHigh
color_low	colorLow
manual_max	manualMax
manual_min	manualMin
line_weight	lineWeight
date_axis	dateAxis
display_x_axis	displayXAxis
display_hidden	displayHidden
min_axis_type	minAxisType
max_axis_type	maxAxisType
right_to_left	rightToLeft
...	additional arguments

Details

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

Value

A string containing XML code

Examples

```
# 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)
```

---

create_tablestyle *Create custom (pivot) table styles*

Description

Create a custom (pivot) table style. These functions are for expert use only. Use other styling functions instead.

Usage

```
create_tablestyle(
  name,
  whole_table = NULL,
  header_row = NULL,
  total_row = NULL,
  first_column = NULL,
  last_column = NULL,
  first_row_stripe = NULL,
  second_row_stripe = NULL,
  first_column_stripe = NULL,
  second_column_stripe = NULL,
  first_header_cell = NULL,
  last_header_cell = NULL,
  first_total_cell = NULL,
  last_total_cell = NULL,
  ...
)

create_pivottablestyle(
  name,
```

```
  whole_table = NULL,
  header_row = NULL,
  grand_total_row = NULL,
  first_column = NULL,
  grand_total_column = NULL,
  first_row_stripe = NULL,
  second_row_stripe = NULL,
  first_column_stripe = NULL,
  second_column_stripe = NULL,
  first_header_cell = NULL,
  first_subtotal_column = NULL,
  second_subtotal_column = NULL,
  third_subtotal_column = NULL,
  first_subtotal_row = NULL,
  second_subtotal_row = NULL,
  third_subtotal_row = NULL,
  blank_row = NULL,
  first_column_subheading = NULL,
  second_column_subheading = NULL,
  third_column_subheading = NULL,
  first_row_subheading = NULL,
  second_row_subheading = NULL,
  third_row_subheading = NULL,
  page_field_labels = NULL,
  page_field_values = NULL,
  ...
)
```

Arguments

`name`	name
`whole_table`	wholeTable
`header_row`, `total_row`	...Row
`first_column`, `last_column`	...Column
`first_row_stripe`, `second_row_stripe`	...RowStripe
`first_column_stripe`, `second_column_stripe`	...ColumnStripe
`first_header_cell`, `last_header_cell`	...HeaderCell
`first_total_cell`, `last_total_cell`	...TotalCell
`...`	additional arguments
`grand_total_row`	totalRow

grand_total_column
: lastColumn

first_subtotal_column, second_subtotal_column, third_subtotal_column
: ...SubtotalColumn

first_subtotal_row, second_subtotal_row, third_subtotal_row
: ...SubtotalRow

blank_row
: blankRow

first_column_subheading, second_column_subheading, third_column_subheading
: ...ColumnSubheading

first_row_subheading, second_row_subheading, third_row_subheading
: ...RowSubheading

page_field_labels
: pageFieldLabels

page_field_values
: pageFieldValues

See Also

Other style creating functions: create_border(), create_cell_style(), create_colors_xml(), create_dxfs_style(), create_fill(), create_font(), create_numfmt()

---

creators-wb *Modify creators of a workbook*

Description

Modify and get workbook creators

Usage

```
wb_add_creators(wb, creators)

wb_set_creators(wb, creators)

wb_remove_creators(wb, creators)

wb_get_creators(wb)
```

Arguments

wb	A wbWorkbook object
creators	A character vector of names

Value

- wb_set_creators(), wb_add_creators(), and wb_remove_creators() return the wbWorkbook object
- wb_get_creators() returns a character vector of creators

See Also

Other workbook wrappers: base_font-wb, col_widths-wb, grouping-wb, row_heights-wb, wb_add_chartsheet(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_worksheet(), wb_base_colors, wb_clone_worksheet(), wb_copy_cells(), wb_freeze_pane(), wb_merge_cells(), wb_save(), wb_set_last_modified_by(), wb_workbook()

Examples

```
# workbook made with default creator (see [wbWorkbook])
wb <- wb_workbook()
wb_get_creators(wb)

# add a new creator (assuming "test" isn't your default creator)
wb <- wb_add_creators(wb, "test")
wb_get_creators(wb)

# remove the creator (should be the same as before)
wb <- wb_remove_creators(wb, "test")
wb_get_creators(wb)
```

---

dims_helper *Helper functions to work with* dims

Description

Internal helpers to (de)construct a dims argument from/to a row and column vector. Exported for user convenience.

Usage

```
dims_to_rowcol(x, as_integer = FALSE)

rowcol_to_dims(row, col, single = TRUE)
```

Arguments

x	a dimension object "A1" or "A1:A1"
as_integer	If the output should be returned as integer, (defaults to string)
row	a numeric vector of rows
col	a numeric or character vector of cols
single	argument indicating if rowcol_to_dims() returns a single cell dimension

Value

- A dims string for _to_dim i.e "A1:A1"
- A list of rows and columns for to_rowcol

See Also

wb_dims()

Examples

```
dims_to_rowcol("A1:J10")
wb_dims(1:10, 1:10)
```

---

filter-wb *Add/remove column filters in a worksheet*

Description

Add or remove excel column filters to a worksheet

Usage

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

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

Arguments

wb	A workbook object
sheet	A worksheet name or index. In wb_remove_filter(), you may supply a vector of worksheets.
rows	A row number.
cols	columns to add filter to.

Details

Adds filters to worksheet columns, same as with_filter = TRUE in wb_add_data() wb_add_data_table() automatically adds filters to first row of a table.

NOTE Can only have a single filter per worksheet unless using tables.

See Also

wb_add_data(), wb_add_data_table()

Other worksheet content functions: col_widths-wb, grouping-wb, named_region-wb, row_heights-wb, wb_add_conditional_formatting(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_thread(), wb_freeze_pane(), wb_merge_cells()

Examples

```
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, with_filter = TRUE)

## Similarly
wb$add_data_table(3, iris)
wb <- wb_workbook()
wb$add_worksheet("Sheet 1")
wb$add_worksheet("Sheet 2")
wb$add_worksheet("Sheet 3")

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

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

## Similarly
wb$add_data_table(3, iris)

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

---

fmt_txt *format strings independent of the cell style.*

Description

format strings independent of the cell style.

Usage

```
fmt_txt(
  x,
  bold = FALSE,
  italic = FALSE,
  underline = FALSE,
  strike = FALSE,
  size = NULL,
  color = NULL,
```

```
  font = NULL,
  charset = NULL,
  outline = NULL,
  vert_align = NULL
)

## S3 method for class 'fmt_txt'
x + y

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

## S3 method for class 'fmt_txt'
print(x, ...)
```

Arguments

x, y	an openxlsx2 fmt_txt string
bold	bold
italic	italic
underline	underline
strike	strike
size	the font size
color	a wbColor color for the font
font	the font name
charset	integer value from the table below
outline	TRUE or FALSE
vert_align	baseline, superscript, or subscript
...	additional arguments for default print

Details

The result is an xml string. It is possible to paste multiple `fmt_txt()` strings together to create a string with differing styles.

Using `fmt_txt(charset = 161)` will give the Greek Character Set

charset	"Character Set"
0	"ANSI_CHARSET"
1	"DEFAULT_CHARSET"
2	"SYMBOL_CHARSET"
77	"MAC_CHARSET"
128	"SHIFTJIS_CHARSET"
129	"HANGUL_CHARSET"
130	"JOHAB_CHARSET"
134	"GB2312_CHARSET"

136	"CHINESEBIG5_CHARSET"
161	"GREEK_CHARSET"
162	"TURKISH_CHARSET"
163	"VIETNAMESE_CHARSET"
177	"HEBREW_CHARSET"
178	"ARABIC_CHARSET"
186	"BALTIC_CHARSET"
204	"RUSSIAN_CHARSET"
222	"THAI_CHARSET"
238	"EASTEUROPE_CHARSET"
255	"OEM_CHARSET"

You can join additional objects into fmt_txt() objects using "+". Though be aware that `fmt_txt("sum:") + (2 + 2)` is different to `fmt_txt("sum:") + 2 + 2`.

Examples

```
fmt_txt("bar", underline = TRUE)
fmt_txt("foo ", bold = TRUE) + fmt_txt("bar")
as.character(fmt_txt(2))
```

`grouping-wb` *Group rows and columns in a worksheet*

Description

Group a selection of rows or cols

Usage

```
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: base_font-wb, col_widths-wb, creators-wb, row_heights-wb, wb_add_chartsheet(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_worksheet(), wb_base_colors, wb_clone_worksheet(), wb_copy_cells(), wb_freeze_pane(), wb_merge_cells(), wb_save(), wb_set_last_modified_by(), wb_workbook()

Other worksheet content functions: col_widths-wb, filter-wb, named_region-wb, row_heights-wb, wb_add_conditional_formatting(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_thread(), wb_freeze_pane(), wb_merge_cells()

Other workbook wrappers: base_font-wb, col_widths-wb, creators-wb, row_heights-wb, wb_add_chartsheet(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_worksheet(), wb_base_colors, wb_clone_worksheet(), wb_copy_cells(), wb_freeze_pane(), wb_merge_cells(), wb_save(), wb_set_last_modified_by(), wb_workbook()

Examples

```
# 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, row_names = 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 <- 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)

### create grouping levels
grp_rows <- list(
  "1" = seq(2, 3),
  "2" = seq(4, 8),
  "3" = seq(9, 13)
)

grp_cols <- list(
  "1" = seq(2, 4),
  "2" = seq(5, 7),
  "3" = seq(8, 10),
  "4" = seq(11, 13)
)

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

wb$group_cols("AirPass", cols = grp_cols)
wb$group_rows("AirPass", rows = grp_rows)
```

`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)
```

named_region-wb *Modify named regions in a worksheet*

Description

Create / delete a named region. You can also specify a named region by using the name argument in wb_add_data(x = iris, name = "my-region"). It is important to note that named regions are not case-sensitive and must be unique.

Usage

```
wb_add_named_region(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  name,
  local_sheet = FALSE,
  overwrite = FALSE,
  comment = NULL,
  hidden = NULL,
  custom_menu = NULL,
  description = NULL,
  is_function = NULL,
  function_group_id = NULL,
  help = NULL,
  local_name = NULL,
  publish_to_server = NULL,
  status_bar = NULL,
  vb_procedure = NULL,
  workbook_parameter = NULL,
  xml = NULL,
  ...
)

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

wb_get_named_regions(wb, tables = FALSE, x = NULL)
```

Arguments

wb	A Workbook object
sheet	A name or index of a worksheet
dims	Worksheet cell range of the region ("A1:D4").
name	Name for region. A character vector of length 1. Note that region names must be case-insensitive unique.
local_sheet	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
hidden	Should the named region be hidden?
custom_menu, description, is_function, function_group_id, help, local_name, publish_to_server, status_bar, vb_procedure, workbook_parameter, xml	Unknown XML feature
...	additional arguments
tables	Should included both data tables and named regions in the result?
x	Deprecated. Use wb. For Excel input use wb_load() to first load the xlsx file as a workbook.

Details

You can use the wb_dims() helper to specify the cell range of the named region

Value

A workbook, invisibly.

A data frame with the all named regions in wb. Or NULL, if none are found.

See Also

wb_get_tables()

Other worksheet content functions: col_widths-wb, filter-wb, grouping-wb, row_heights-wb, wb_add_conditional_formatting(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_thread(), wb_freeze_pane(), wb_merge_cells()

Examples

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

## specify region
wb$add_data(x = iris, start_col = 1, start_row = 1)
wb$add_named_region(
  name = "iris",
  dims = wb_dims(x = iris)
)

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

## delete one
wb$remove_named_region(name = "iris2")
wb$get_named_regions()
## read named regions
```

```
df <- wb_to_df(wb, named_region = "iris")
head(df)


# Extract named regions from a file
out_file <- temp_xlsx()
wb_save(wb, out_file, overwrite = TRUE)

# Load the file as a workbook first, then get named regions.
wb1 <- wb_load(out_file)
wb1$get_named_regions()
```

`openxlsx2-deprecated` *Deprecated functions in package* openxlsx2

Description

These functions are provided for compatibility with older versions of `openxlsx2`, and may be defunct as soon as the next release. This guide helps you update your code to the latest standards.

As of openxlsx2 v1.0, API change should be minimal.

Internal functions

These functions are used internally by openxlsx2. It is no longer advertised to use them in scripts. They originate from openxlsx, but do not fit openxlsx2's API.

You should be able to modify

- `delete_data()` -> `wb_clean_sheet()`
- `write_data()` -> `wb_add_data()`
- `write_datatable()` -> `wb_add_data_table()`
- `write_comment()` -> `wb_add_comment()`
- `remove_comment()` -> `wb_remove_comment()`
- `write_formula()` -> `wb_add_formula()`

You should be able to change those with minimal changes

Deprecated functions

First of all, you can set an option that will add warnings when using deprecated functions.

```
options("openxlsx2.soon_deprecated" = TRUE)
```

Argument changes

For consistency, arguments were renamed to snake_case for the 0.8 release. It is now recommended to use `dims` (the cell range) in favor of `row`, `col`, `start_row`, `start_col`

See `wb_dims()` as it provides many options on how to provide cell range

Functions with a new name

These functions were renamed for consistency.

- convertToExcelDate() -> convert_to_excel_date()
- wb_grid_lines() -> wb_set_grid_lines()
- create_comment() -> wb_comment()

Deprecated usage

- wb_get_named_regions() will no longer allow providing a file.

```
## Before
wb_get_named_regions(file)

## Now
wb <- wb_load(file)
wb_get_named_regions(wb)
# also possible
wb_load(file)$get_named_regions()`
```

See Also

.Deprecated

---

openxlsx2_options *Options consulted by openxlsx2*

Description

The openxlsx2 package allows the user to set global options to simplify formatting:

If the built-in defaults don't suit you, set one or more of these options. Typically, this is done in the .Rprofile startup file

- options("openxlsx2.borderColor" = "black")
- options("openxlsx2.borderStyle" = "thin")
- options("openxlsx2.dateFormat" = "mm/dd/yyyy")
- options("openxlsx2.datetimeFormat" = "yyyy-mm-dd hh:mm:ss")
- options("openxlsx2.maxWidth" = NULL) (Maximum width allowed in Excel is 250)
- options("openxlsx2.minWidth" = NULL)
- options("openxlsx2.numFmt" = NULL)
- options("openxlsx2.paperSize" = 9) corresponds to a A4 paper size
- options("openxlsx2.orientation" = "portrait") page orientation
- options("openxlsx2.sheet.default_name" = "Sheet")

- `options("openxlsx2.rightToLeft" = NULL)`
- `options("openxlsx2.soon_deprecated" = FALSE)` Set to `TRUE` if you want a warning if using some functions deprecated recently in openxlsx2
- `options("openxlsx2.creator")` A default name for the creator of new `wbWorkbook` object with `wb_workbook()` or new comments with `wb_add_comment()`
- `options("openxlsx2.thread_id")` the default person id when adding a threaded comment to a cell with `wb_add_thread()`
- `options("openxlsx2.accountingFormat" = 4)`
- `options("openxlsx2.currencyFormat" = 4)`
- `options("openxlsx2.commaFormat" = 3)`
- `options("openxlsx2.percentageFormat" = 10)`
- `options("openxlsx2.scientificFormat" = 48)`
- `options("openxlsx2.string_nums" = TRUE)` numerics in character columns will be converted. `"1"` will be written as `1`
- `options("openxlsx2.na.strings" = "#N/A")` consulted by `write_xlsx()`, `wb_add_data()` and `wb_add_data_table()`.

`person-wb` *Helper for adding threaded comments*

Description

Adds a person to a workbook, so that they can be the author of threaded comments in a workbook with `wb_add_thread()`

Usage

```
wb_add_person(wb, name = NULL, id = NULL, user_id = NULL, provider_id = "None")

wb_get_person(wb, name = NULL)
```

Arguments

`wb`	a Workbook
`name`	the name of the person to display.
`id`	(optional) the display id
`user_id`	(optional) the user id
`provider_id`	(optional) the provider id

See Also

`wb_add_thread()`

print.pugi_xml *print pugi_xml*

Description

print pugi_xml

Usage

```
## 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)
```

properties-wb *Modify workbook properties*

Description

This function is useful for workbooks that are loaded. It can be used to set the workbook `title`, `subject` and `category` field. Use `wb_workbook()` to easily set these properties with a new workbook.

Usage

```
wb_get_properties(wb)

wb_set_properties(
  wb,
  creator = NULL,
  title = NULL,
  subject = NULL,
```

```
  category = NULL,
  datetime_created = Sys.time(),
  modifier = NULL,
  keywords = NULL,
  comments = NULL,
  manager = NULL,
  company = NULL,
  custom = NULL
)
```

Arguments

wb	A Workbook object
creator	Creator of the workbook (your name). Defaults to login username or `options("openxlsx2.creator")` if set.
title, subject, category, keywords, comments, manager, company	Workbook property, a string.
datetime_created	The time of the workbook is created
modifier	A character string indicating who was the last person to modify the workbook
custom	A named vector of custom properties added to the workbook

Details

To set properties, the following XML core properties are used.

- title = dc:title
- subject = dc:subject
- creator = dc:creator
- keywords = cp:keywords
- comments = dc:description
- modifier = cp:lastModifiedBy
- datetime_created = dcterms:created
- datetime_modified = dcterms:modified
- category = cp:category

In addition, manager and company are used.

Value

A wbWorkbook object, invisibly.

See Also

`wb_workbook()`

Examples

```
file <- system.file("extdata", "openxlsx2_example.xlsx", package = "openxlsx2")
wb <- wb_load(file)
wb$get_properties()

# Add a title to properties
wb$set_properties(title = "my title")
wb$get_properties()
```

---

pugixml *xml_node*

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>")
xml_value(x, "a")

x <- read_xml("<a><b r=\"1\">2</b></a>")
xml_value(x, "a", "b")

x <- read_xml("<a a=\"1\" b=\"2\">1</a>")
xml_attr(x, "a")

x <- read_xml("<a><b r=\"1\">2</b></a>")
xml_attr(x, "a", "b")
x <- read_xml("<a a=\"1\" b=\"2\">1</a>")
xml_attr(x, "a")

x <- read_xml("<b><a a=\"1\" b=\"2\"/></b>")
xml_attr(x, "b", "a")
```

`read_xml` *read xml file*

Description

read xml file

Usage

```
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 or be returned
`skip_control`	should whitespace character be exported

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/>"))

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)
```

---

row_heights-wb *Modify row heights of a worksheet*

Description

Set / remove custom worksheet row heights

Usage

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

```
wb_remove_row_heights(wb, sheet = current_sheet(), rows)
```

Arguments

wb	A wbWorkbook object
sheet	A name or index of a worksheet. (A vector is accepted for remove_row_heights())
rows	Indices of rows to set / remove (if any) custom height.
heights	Heights to set rows to specified in a spreadsheet column height units.
hidden	Option to hide rows. A logical vector of length 1 or length of rows

See Also

Other workbook wrappers: base_font-wb, col_widths-wb, creators-wb, grouping-wb, wb_add_chartsheet(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_worksheet(), wb_base_colors, wb_clone_worksheet(), wb_copy_cells(), wb_freeze_pane(), wb_merge_cells(), wb_save(), wb_set_last_modified_by(), wb_workbook()

Other worksheet content functions: col_widths-wb, filter-wb, grouping-wb, named_region-wb, wb_add_conditional_formatting(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_thread(), wb_freeze_pane(), wb_merge_cells()

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)
## remove any custom row heights in row 1
wb$remove_row_heights(sheet = 1, rows = 1)
```

---

sheet_names-wb *Get / Set worksheet names for a workbook*

Description

Gets / Sets the worksheet names for a wbWorkbook object.

Usage

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

wb_get_sheet_names(wb, escape = FALSE)
```

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
escape	Should the xml special characters be escaped?

Details

This only changes the sheet name as shown in spreadsheet software and will not alter it elsewhere. Not in formulas, chart references, named regions, pivot tables or anywhere else.

Value

- set_: The wbWorkbook object.
- get_: A named character vector of sheet names in order. The names represent the original value of the worksheet prior to any character substitutions.

---

sheet_visibility-wb *Get/set worksheet visible state in a workbook*

Description

Get and set worksheet visible state. This allows to hide worksheets from the workbook. The visibility of a worksheet can either be "visible", "hidden", or "veryHidden". You can set this when creating a worksheet with wb_add_worksheet(visible = FALSE)

Usage

```
wb_get_sheet_visibility(wb)

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, if providing a character vector, you can provide any of "hidden", "visible", or "veryHidden"

Value

- wb_set_sheet_visibility: The Workbook object, invisibly.
- wb_get_sheet_visibility(): A character vector of the worksheet visibility value

Examples

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

wb$get_sheet_visibility()
wb$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

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

Arguments

wb	workbook
sheet	worksheet

---

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

waivers openxlsx2 *waivers*

Description

Waiver functions for openxlsx2 functions.

- current_sheet() uses wb_get_active_sheet() by default if performing actions on a worksheet, for example when you add data.
- next_sheet() is used when you add a new worksheet, a new chartsheet or when you add a pivot table

Usage

```
current_sheet()

next_sheet()

na_strings()
```

Value

An object of class openxlsx2_waiver

wbWorkbook *Workbook class*

Description

This is the class used by openxlsx2 to modify workbooks from R. You can load an existing workbook with wb_load() and create a new one with wb_workbook().

After that, you can modify the wbWorkbook object through two primary methods:

Wrapper Function Method: Utilizes the wb family of functions that support piping to streamline operations.

```
wb <- wb_workbook(creator = "My name here") %>%
  wb_add_worksheet(sheet = "Expenditure", grid_lines = FALSE) %>%
  wb_add_data(x = USPersonalExpenditure, row_names = TRUE)
```

Chaining Method: Directly modifies the object through a series of chained function calls.

```
wb <- wb_workbook(creator = "My name here")$
  add_worksheet(sheet = "Expenditure", grid_lines = FALSE)$
  add_data(x = USPersonalExpenditure, row_names = TRUE)
```

While wrapper functions require explicit assignment of their output to reflect changes, chained functions inherently modify the input object. Both approaches are equally supported, offering flexibility to suit user preferences. The documentation mainly highlights the use of wrapper functions.

```
# Import workbooks
path <- system.file("extdata/openxlsx2_example.xlsx", package = "openxlsx2")
wb <- wb_load(path)

## or create one yourself
wb <- wb_workbook()
# add a worksheet
wb$add_worksheet("sheet")
# add some data
wb$add_data("sheet", cars)
# Add data with piping in a different location
wb <- wb %>% wb_add_data(x = cars, dims = wb_dims(from_dims = "D4"))
# open it in your default spreadsheet software
if (interactive()) wb$open()
```

Note that the documentation is more complete in each of the wrapper functions. (i.e. `?wb_add_data` rather than `?wbWorkbook`).

Public fields

`sheet_names` The names of the sheets

`calcChain` calcChain

`charts` charts

`is_chartsheet` A logical vector identifying if a sheet is a chartsheet.

`customXml` customXml

`connections` connections

`ctrlProps` ctrlProps

`Content_Types` Content_Types

`app` app

`core` The XML core

`custom` custom

`drawings` drawings

`drawings_rels` drawings_rels

`docMetadata` doc_meta_data

`embeddings` embeddings

`externalLinks` externalLinks

`externalLinksRels` externalLinksRels

`headFoot` The header and footer

`media` media

`metadata` contains cell/value metadata imported on load from xl/metadata.xml

`persons` Persons of the workbook. to be used with `wb_add_thread()`

`pivotTables` pivotTables

`pivotTables.xml.rels` pivotTables.xml.rels

`pivotDefinitions` pivotDefinitions

`pivotRecords` pivotRecords

`pivotDefinitionsRels` pivotDefinitionsRels

`queryTables` queryTables

`richData` richData

`slicers` slicers

`slicerCaches` slicerCaches

`sharedStrings` sharedStrings

`styles_mgr` styles_mgr

`tables` tables

`tables.xml.rels` tables.xml.rels

`theme` theme

`vbaProject` vbaProject

`vml` vml

`vml_rels` vml_rels

`comments` Comments (notes) present in the workbook.

`threadComments` Threaded comments

`timelines` timelines

`timelineCaches` timelineCaches

`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

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_pivot_table()
- wbWorkbook$add_slicer()
- wbWorkbook$remove_slicer()
- wbWorkbook$add_timeline()
- wbWorkbook$remove_timeline()
- wbWorkbook$add_formula()
- wbWorkbook$add_style()
- wbWorkbook$to_df()
- wbWorkbook$load()
- wbWorkbook$save()
- wbWorkbook$open()
- wbWorkbook$buildTable()
- wbWorkbook$update_table()
- wbWorkbook$copy_cells()
- wbWorkbook$get_base_font()
- wbWorkbook$set_base_font()
- wbWorkbook$get_base_colors()
- wbWorkbook$get_base_colours()
- wbWorkbook$set_base_colors()
- wbWorkbook$set_base_colours()
- wbWorkbook$set_bookview()
- wbWorkbook$get_sheet_names()
- wbWorkbook$set_sheet_names()
- 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$add_comment()
- wbWorkbook$get_comment()

- wbWorkbook$remove_comment()
- wbWorkbook$add_thread()
- wbWorkbook$get_thread()
- wbWorkbook$add_conditional_formatting()
- wbWorkbook$remove_conditional_formatting()
- wbWorkbook$add_image()
- wbWorkbook$add_plot()
- wbWorkbook$add_drawing()
- wbWorkbook$add_chart_xml()
- wbWorkbook$add_mschart()
- wbWorkbook$add_form_control()
- wbWorkbook$print()
- wbWorkbook$protect()
- wbWorkbook$protect_worksheet()
- wbWorkbook$get_properties()
- wbWorkbook$set_properties()
- wbWorkbook$add_mips()
- wbWorkbook$get_mips()
- wbWorkbook$set_creators()
- wbWorkbook$add_creators()
- wbWorkbook$remove_creators()
- wbWorkbook$set_last_modified_by()
- wbWorkbook$set_page_setup()
- wbWorkbook$page_setup()
- wbWorkbook$set_header_footer()
- wbWorkbook$get_tables()
- wbWorkbook$remove_tables()
- wbWorkbook$add_filter()
- wbWorkbook$remove_filter()
- wbWorkbook$set_grid_lines()
- wbWorkbook$grid_lines()
- wbWorkbook$add_named_region()
- wbWorkbook$get_named_regions()
- wbWorkbook$remove_named_region()
- wbWorkbook$set_order()
- wbWorkbook$get_sheet_visibility()
- wbWorkbook$set_sheet_visibility()
- wbWorkbook$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$set_cell_style_across()
- wbWorkbook$add_named_style()
- wbWorkbook$add_dxfs_style()
- wbWorkbook$clone_sheet_style()
- wbWorkbook$add_sparklines()
- wbWorkbook$add_ignore_error()
- wbWorkbook$set_sheetview()
- wbWorkbook$add_person()
- wbWorkbook$get_person()
- wbWorkbook$get_active_sheet()
- wbWorkbook$set_active_sheet()
- wbWorkbook$get_selected()
- wbWorkbook$set_selected()
- wbWorkbook$clone()

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

Usage:

```
wbWorkbook$new(
  creator = NULL,
  title = NULL,
  subject = NULL,
  category = NULL,
  datetime_created = Sys.time(),
  theme = NULL,
  keywords = NULL,
  comments = NULL,
  manager = NULL,
  company = NULL,
  ...
)
```

Arguments:

`creator` character vector of creators. Duplicated are ignored.

`title`, `subject`, `category`, `keywords`, `comments`, `manager`, `company` workbook properties

`datetime_created` 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.

`theme` Optional theme identified by string or number

`...` additional arguments

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

Returns: The integer position of the sheet

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

Usage:

```
wbWorkbook$add_chartsheet(
  sheet = next_sheet(),
  tab_color = NULL,
  zoom = 100,
  visible = c("true", "false", "hidden", "visible", "veryhidden"),
  ...
)
```

Arguments:

`sheet` The name of the sheet

`tab_color` tab_color

`zoom` zoom

`visible` visible

`...` additional arguments

Returns: The `wbWorkbook` object, invisibly

Method `add_worksheet()`**:** Add worksheet to the `wbWorkbook` object

Usage:

```
wbWorkbook$add_worksheet(
  sheet = next_sheet(),
  grid_lines = TRUE,
  row_col_headers = TRUE,
```

```
  tab_color = NULL,
  zoom = 100,
  header = NULL,
  footer = NULL,
  odd_header = header,
  odd_footer = footer,
  even_header = header,
  even_footer = footer,
  first_header = header,
  first_footer = footer,
  visible = c("true", "false", "hidden", "visible", "veryhidden"),
  has_drawing = FALSE,
  paper_size = 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` The name of the sheet

`grid_lines` gridLines

`row_col_headers` rowColHeaders

`tab_color` tabColor

`zoom` zoom

`header` header

`footer` footer

`odd_header` oddHeader

`odd_footer` oddFooter

`even_header` evenHeader

`even_footer` evenFooter

`first_header` firstHeader

`first_footer` firstFooter

`visible` visible

`has_drawing` hasDrawing

`paper_size` paperSize

`orientation` orientation

`hdpi` hdpi

`vdpi` vdpi

`...` additional arguments

Returns: The `wbWorkbook` object, invisibly

Method `clone_worksheet()`**:** Clone a workbooksheet to another workbook

Usage:

```
wbWorkbook$clone_worksheet(
  old = current_sheet(),
  new = next_sheet(),
  from = NULL
)
```

Arguments:

`old` name of worksheet to clone

`new` name of new worksheet to add

`from` name of new worksheet to add

Method `add_data()`**:** add data

Usage:

```
wbWorkbook$add_data(
  sheet = current_sheet(),
  x,
  dims = wb_dims(start_row, start_col),
  start_col = 1,
  start_row = 1,
  array = FALSE,
  col_names = TRUE,
  row_names = FALSE,
  with_filter = FALSE,
  name = NULL,
  sep = ", ",
  apply_cell_style = TRUE,
  remove_cell_style = FALSE,
  na.strings = na_strings(),
  inline_strings = TRUE,
  enforce = FALSE,
  ...
)
```

Arguments:

`sheet` The name of the sheet

`x` x

`dims` Cell range in a sheet

`start_col` startCol

`start_row` startRow

`array` array

`col_names` colNames

`row_names` rowNames

`with_filter` withFilter

`name` name

`sep` sep

`apply_cell_style` applyCellStyle

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

na.strings Value used for replacing NA values from x. Default na_strings() uses the special #N/A value within the workbook.

inline_strings write characters as inline strings

enforce enforce that selected dims is filled. For this to work, dims must match x

... additional arguments

return The wbWorkbook object

Method add_data_table(): add a data table

Usage:

```
wbWorkbook$add_data_table(
  sheet = current_sheet(),
  x,
  dims = wb_dims(start_row, start_col),
  start_col = 1,
  start_row = 1,
  col_names = TRUE,
  row_names = FALSE,
  table_style = "TableStyleLight9",
  table_name = NULL,
  with_filter = TRUE,
  sep = ", ",
  first_column = FALSE,
  last_column = FALSE,
  banded_rows = TRUE,
  banded_cols = FALSE,
  apply_cell_style = TRUE,
  remove_cell_style = FALSE,
  na.strings = na_strings(),
  inline_strings = TRUE,
  total_row = FALSE,
  ...
)
```

Arguments:

sheet The name of the sheet

x x

dims Cell range in a sheet

start_col startCol

start_row startRow

col_names colNames

row_names rowNames

table_style tableStyle

table_name tableName

with_filter withFilter

sep sep

first_column firstColumn

`last_column` lastColumn

`banded_rows` bandedRows

`banded_cols` bandedCols

`apply_cell_style` applyCellStyle

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

`na.strings` Value used for replacing `NA` values from `x`. Default `na_strings()` uses the special `#N/A` value within the workbook.

`inline_strings` write characters as inline strings

`total_row` write total rows to table

`...` additional arguments

Returns: The `wbWorkbook` object

Method `add_pivot_table()`**:** add pivot table

Usage:

```
wbWorkbook$add_pivot_table(
  x,
  sheet = next_sheet(),
  dims = "A3",
  filter,
  rows,
  cols,
  data,
  fun,
  params,
  pivot_table,
  slicer,
  timeline
)
```

Arguments:

`x` a wb_data object

`sheet` The name of the sheet

`dims` the worksheet cell where the pivot table is placed

`filter` a character object with names used to filter

`rows` a character object with names used as rows

`cols` a character object with names used as cols

`data` a character object with names used as data

`fun` a character object of functions to be used with the data

`params` a list of parameters to modify pivot table creation

`pivot_table` a character object with a name for the pivot table

`slicer` a character object with names used as slicer

`timeline` a character object with names used as timeline

Details: `fun` can be either of AVERAGE, COUNT, COUNTA, MAX, MIN, PRODUCT, STDEV, STDEVP, SUM, VAR, VARP

Returns: The `wbWorkbook` object

Method `add_slicer()`: add pivot table

Usage:

```
wbWorkbook$add_slicer(
  x,
  dims = "A1",
  sheet = current_sheet(),
  pivot_table,
  slicer,
  params
)
```

Arguments:

`x` a wb_data object

`dims` the worksheet cell where the pivot table is placed

`sheet` The name of the sheet

`pivot_table` the name of a pivot table on the selected sheet

`slicer` a variable used as slicer for the pivot table

`params` a list of parameters to modify pivot table creation

Returns: The `wbWorkbook` object

Method `remove_slicer()`: add pivot table

Usage:

```
wbWorkbook$remove_slicer(sheet = current_sheet())
```

Arguments:

`sheet` The name of the sheet

Returns: The `wbWorkbook` object

Method `add_timeline()`: add pivot table

Usage:

```
wbWorkbook$add_timeline(
  x,
  dims = "A1",
  sheet = current_sheet(),
  pivot_table,
  timeline,
  params
)
```

Arguments:

`x` a wb_data object

`dims` the worksheet cell where the pivot table is placed

`sheet` The name of the sheet

`pivot_table` the name of a pivot table on the selected sheet

`timeline` a variable used as timeline for the pivot table

`params` a list of parameters to modify pivot table creation

Returns: The wbWorkbook object

Method remove_timeline(): add pivot table

Usage:

```
wbWorkbook$remove_timeline(sheet = current_sheet())
```

Arguments:

sheet The name of the sheet

Returns: The wbWorkbook object

Method add_formula(): Add formula

Usage:

```
wbWorkbook$add_formula(
  sheet = current_sheet(),
  x,
  dims = wb_dims(start_row, start_col),
  start_col = 1,
  start_row = 1,
  array = FALSE,
  cm = FALSE,
  apply_cell_style = TRUE,
  remove_cell_style = FALSE,
  enforce = FALSE,
  ...
)
```

Arguments:

sheet The name of the sheet

x x

dims Cell range in a sheet

start_col startCol

start_row startRow

array array

cm cm

apply_cell_style applyCellStyle

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

enforce enforce dims

... additional arguments

Returns: The wbWorkbook object

Method add_style(): add style

Usage:

```
wbWorkbook$add_style(style = NULL, style_name = NULL)
```

Arguments:

style style

`style_name` style_name

Returns: The `wbWorkbook` object

Method `to_df()`**:** to_df

Usage:

```
wbWorkbook$to_df(
  sheet,
  start_row = 1,
  start_col = NULL,
  row_names = FALSE,
  col_names = TRUE,
  skip_empty_rows = FALSE,
  skip_empty_cols = FALSE,
  skip_hidden_rows = FALSE,
  skip_hidden_cols = FALSE,
  rows = NULL,
  cols = NULL,
  detect_dates = TRUE,
  na.strings = "#N/A",
  na.numbers = NA,
  fill_merged_cells = FALSE,
  dims,
  show_formula = FALSE,
  convert = TRUE,
  types,
  named_region,
  keep_attributes = FALSE,
  check_names = FALSE,
  ...
)
```

Arguments:

`sheet` Either sheet name or index. When missing the first sheet in the workbook is selected.

`start_row` first row to begin looking for data.

`start_col` first column to begin looking for data.

`row_names` If TRUE, the first col of data will be used as row names.

`col_names` If TRUE, the first row of data will be used as column names.

`skip_empty_rows` If TRUE, empty rows are skipped.

`skip_empty_cols` If TRUE, empty columns are skipped.

`skip_hidden_rows` If TRUE, hidden rows are skipped.

`skip_hidden_cols` If TRUE, hidden 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.

`detect_dates` 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.

`fill_merged_cells` 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.

`show_formula` 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, 3: posixt, 4:logical. Names must match the returned data

`named_region` Character string with a named_region (defined name or table). If no sheet is selected, the first appearance will be selected.

`keep_attributes` If TRUE additional attributes are returned. (These are used internally to define a cell type.)

`check_names` If TRUE then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names.

`...` additional arguments

Returns: a data frame

Method `load()`**:** load workbook

Usage:

```
wbWorkbook$load(file, sheet, data_only = FALSE, ...)
```

Arguments:

`file` file

`sheet` The name of the sheet

`data_only` data_only

`...` additional arguments

Returns: The `wbWorkbook` object invisibly

Method `save()`**:** Save the workbook

Usage:

```
wbWorkbook$save(file = self$path, overwrite = TRUE, path = NULL)
```

Arguments:

`file` The path to save the workbook to

`overwrite` If `FALSE`, will not overwrite when `path` exists

`path` Deprecated argument previously used for file. Please use file in new code.

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()`

Details: minor helper wrapping xl_open which does the entire same thing

Returns: The `wbWorkbook`, invisibly

Method `buildTable()`**:** Build table

Usage:

```
wbWorkbook$buildTable(
  sheet = current_sheet(),
  colNames,
  ref,
  showColNames,
  tableStyle,
  tableName,
  withFilter = TRUE,
  totalsRowCount = 0,
  totalLabel = FALSE,
  showFirstColumn = 0,
  showLastColumn = 0,
  showRowStripes = 1,
  showColumnStripes = 0
)
```

Arguments:

`sheet` The name of the sheet

`colNames` colNames

`ref` ref

`showColNames` showColNames

`tableStyle` tableStyle

`tableName` tableName

`withFilter` withFilter

`totalsRowCount` totalsRowCount

`totalLabel` totalLabel

`showFirstColumn` showFirstColumn

`showLastColumn` showLastColumn

`showRowStripes` showRowStripes

`showColumnStripes` showColumnStripes

Returns: The `wbWorksheet` object, invisibly

Method `update_table()`**:** update a data_table

Usage:

```
wbWorkbook$update_table(sheet = current_sheet(), dims = "A1", tabname)
```

Arguments:

`sheet` The name of the sheet

`dims` Cell range in a sheet

tabname a tablename

Returns: The wbWorksheet object, invisibly

Method copy_cells(): copy cells around in a workbook

Usage:

```
wbWorkbook$copy_cells(
  sheet = current_sheet(),
  dims = "A1",
  data,
  as_value = FALSE,
  as_ref = FALSE,
  transpose = FALSE,
  ...
)
```

Arguments:

sheet The name of the sheet

dims Cell range in a sheet

data a wb_data object

as_value should a copy of the value be written

as_ref should references to the cell be written

transpose should the data be written transposed

... additional arguments passed to add_data() if used with as_value

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(): Set the base font

Usage:

```
wbWorkbook$set_base_font(
  font_size = 11,
  font_color = wb_color(theme = "1"),
  font_name = "Aptos Narrow",
  ...
)
```

Arguments:

font_size fontSize

font_color font_color

font_name font_name

... additional arguments

Returns: The wbWorkbook object

Method `get_base_colors()`**:** Get the base color

Usage:

```
wbWorkbook$get_base_colors(xml = FALSE, plot = TRUE)
```

Arguments:

`xml` xml

`plot` plot

Method `get_base_colours()`**:** Get the base colour

Usage:

```
wbWorkbook$get_base_colours(xml = FALSE, plot = TRUE)
```

Arguments:

`xml` xml

`plot` plot

Method `set_base_colors()`**:** Set the base color

Usage:

```
wbWorkbook$set_base_colors(theme = "Office", ...)
```

Arguments:

`theme` theme

`...` ...

Returns: The `wbWorkbook` object

Method `set_base_colours()`**:** Set the base colour

Usage:

```
wbWorkbook$set_base_colours(theme = "Office", ...)
```

Arguments:

`theme` theme

`...` ...

Returns: The `wbWorkbook` object

Method `set_bookview()`**:** Set the book views

Usage:

```
wbWorkbook$set_bookview(
  active_tab = NULL,
  auto_filter_date_grouping = NULL,
  first_sheet = NULL,
  minimized = NULL,
  show_horizontal_scroll = NULL,
  show_sheet_tabs = NULL,
  show_vertical_scroll = NULL,
  tab_ratio = NULL,
  visibility = NULL,
  window_height = NULL,
```

```
  window_width = NULL,
  x_window = NULL,
  y_window = NULL,
  ...
)
```

Arguments:

`active_tab` activeTab

`auto_filter_date_grouping` autoFilterDateGrouping

`first_sheet` firstSheet

`minimized` minimized

`show_horizontal_scroll` showHorizontalScroll

`show_sheet_tabs` showSheetTabs

`show_vertical_scroll` showVerticalScroll

`tab_ratio` tabRatio

`visibility` visibility

`window_height` windowHeight

`window_width` windowWidth

`x_window` xWindow

`y_window` yWindow

`...` additional arguments

Returns: The `wbWorkbook` object

Method `get_sheet_names()`**:** Get sheet names

Usage:

```
wbWorkbook$get_sheet_names(escape = FALSE)
```

Arguments:

`escape` Logical if the xml special characters are escaped

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

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 `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` The name of the sheet

`rows` rows

`heights` heights

`hidden` hidden

Returns: The `wbWorkbook` object, invisibly

Method `remove_row_heights()`**:** Removes a row height for a sheet

Usage:

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

Arguments:

`sheet` The name of the sheet

`rows` rows

Returns: The `wbWorkbook` object, invisibly

Method `createCols()`**:** creates column object for worksheet

Usage:

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

Arguments:

`sheet` The name of the sheet

`n` n

`beg` beg

`end` end

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

Usage:

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

Arguments:

`sheet` The name of the 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 The name of the sheet

cols columns

Returns: The wbWorkbook object

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(): Set column widths

Usage:

```
wbWorkbook$set_col_widths(
  sheet = current_sheet(),
  cols,
  widths = 8.43,
  hidden = FALSE
)
```

Arguments:

sheet The name of the 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` The name of the 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` The name of the 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(),
  dims = "A1",
  type,
  operator,
  value,
  allow_blank = TRUE,
  show_input_msg = TRUE,
  show_error_msg = TRUE,
  error_style = NULL,
  error_title = NULL,
  error = NULL,
  prompt_title = NULL,
  prompt = NULL,
  ...
)
```

Arguments:

`sheet` The name of the sheet

`dims` Cell range in a sheet

`type` type

`operator` operator

`value` value

`allow_blank` allowBlank

`show_input_msg` showInputMsg

`show_error_msg` showErrorMsg

`error_style` 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)

`error_title` The error title

`error` The error text

`prompt_title` The prompt title

`prompt` The prompt text

`...` additional arguments

Returns: The `wbWorkbook` object

Method `merge_cells()`**:** Set cell merging for a sheet

Usage:

```
wbWorkbook$merge_cells(
  sheet = current_sheet(),
  dims = NULL,
  solve = FALSE,
  ...
)
```

Arguments:

`sheet` The name of the sheet

`dims` Cell range in a sheet

`solve` logical if intersecting cells should be solved

`...` additional arguments

Returns: The `wbWorkbook` object, invisibly

Method `unmerge_cells()`**:** Removes cell merging for a sheet

Usage:

```
wbWorkbook$unmerge_cells(sheet = current_sheet(), dims = NULL, ...)
```

Arguments:

`sheet` The name of the sheet

`dims` Cell range in a sheet

`...` additional arguments

Returns: The `wbWorkbook` object, invisibly

Method `freeze_pane()`**:** Set freeze panes for a sheet

Usage:

```
wbWorkbook$freeze_pane(
  sheet = current_sheet(),
  first_active_row = NULL,
  first_active_col = NULL,
```

```
  first_row = FALSE,
  first_col = FALSE,
  ...
)
```

Arguments:

`sheet` The name of the sheet

`first_active_row` first_active_row

`first_active_col` first_active_col

`first_row` first_row

`first_col` first_col

`...` additional arguments

Returns: The `wbWorkbook` object, invisibly

Method `add_comment()`**:** Add comment

Usage:

```
wbWorkbook$add_comment(sheet = current_sheet(), dims = "A1", comment, ...)
```

Arguments:

`sheet` The name of the sheet

`dims` row and column as spreadsheet dimension, e.g. "A1"

`comment` a comment to apply to the worksheet

`...` additional arguments

Returns: The `wbWorkbook` object

Method `get_comment()`**:** Get comments

Usage:

```
wbWorkbook$get_comment(sheet = current_sheet(), dims = NULL)
```

Arguments:

`sheet` sheet

`dims` dims

Returns: A data frame containing comments

Method `remove_comment()`**:** Remove comment

Usage:

```
wbWorkbook$remove_comment(sheet = current_sheet(), dims = "A1", ...)
```

Arguments:

`sheet` The name of the sheet

`dims` row and column as spreadsheet dimension, e.g. "A1"

`...` additional arguments

Returns: The `wbWorkbook` object

Method `add_thread()`**:** add threaded comment to worksheet

Usage:

```
wbWorkbook$add_thread(
  sheet = current_sheet(),
  dims = "A1",
  comment = NULL,
  person_id,
  reply = FALSE,
  resolve = FALSE
)
```

Arguments:

`sheet` The name of the sheet

`dims` Cell range in a sheet

`comment` the comment to add

`person_id` the person Id this should be added for

`reply` logical if the comment is a reply

`resolve` logical if the comment should be marked as resolved

Method `get_thread()`**:** Get threads

Usage:

```
wbWorkbook$get_thread(sheet = current_sheet(), dims = NULL)
```

Arguments:

`sheet` sheet

`dims` dims

Returns: A data frame containing threads

Method `add_conditional_formatting()`**:** Add conditional formatting

Usage:

```
wbWorkbook$add_conditional_formatting(
  sheet = current_sheet(),
  dims = NULL,
  rule = NULL,
  style = NULL,
 type = c("expression", "colorScale", "dataBar", "iconSet", "duplicatedValues",
    "uniqueValues", "containsErrors", "notContainsErrors", "containsBlanks",
   "notContainsBlanks", "containsText", "notContainsText", "beginsWith", "endsWith",
    "between", "topN", "bottomN"),
 params = list(showValue = TRUE, gradient = TRUE, border = TRUE, percent = FALSE, rank =
    5L),
  ...
)
```

Arguments:

`sheet` The name of the sheet

`dims` Cell range in a sheet

`rule` rule

`style` style

`type` type

`params` Additional parameters

`...` additional arguments

Returns: The `wbWorkbook` object

Method `remove_conditional_formatting()`**:** Remove conditional formatting

Usage:

```
wbWorkbook$remove_conditional_formatting(
  sheet = current_sheet(),
  dims = NULL,
  first = FALSE,
  last = FALSE
)
```

Arguments:

`sheet` sheet

`dims` dims

`first` first

`last` last

Returns: The `wbWorkbook` object

Method `add_image()`**:** Insert an image into a sheet

Usage:

```
wbWorkbook$add_image(
  sheet = current_sheet(),
  dims = "A1",
  file,
  width = 6,
  height = 3,
  row_offset = 0,
  col_offset = 0,
  units = "in",
  dpi = 300,
  ...
)
```

Arguments:

`sheet` The name of the sheet

`dims` Cell range in a sheet

`file` file

`width` width

`height` height

`row_offset`, `col_offset` offsets

`units` units

`dpi` dpi

`...` additional arguments

Returns: The wbWorkbook object, invisibly

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

Usage:

```
wbWorkbook$add_plot(
  sheet = current_sheet(),
  dims = "A1",
  width = 6,
  height = 4,
  row_offset = 0,
  col_offset = 0,
  file_type = "png",
  units = "in",
  dpi = 300,
  ...
)
```

Arguments:

sheet The name of the sheet

dims Cell range in a sheet

width width

height height

row_offset, col_offset offsets

file_type fileType

units units

dpi dpi

... additional arguments

Returns: The wbWorkbook object

Method add_drawing(): Add xml drawing

Usage:

```
wbWorkbook$add_drawing(
  sheet = current_sheet(),
  dims = "A1",
  xml,
  col_offset = 0,
  row_offset = 0,
  ...
)
```

Arguments:

sheet The name of the sheet

dims Cell range in a sheet

xml xml

col_offset, row_offset offsets for column and row

... additional arguments

Returns: The wbWorkbook object

Method add_chart_xml(): Add xml chart

Usage:

```
wbWorkbook$add_chart_xml(
  sheet = current_sheet(),
  dims = NULL,
  xml,
  col_offset = 0,
  row_offset = 0,
  ...
)
```

Arguments:

sheet The name of the sheet

dims Cell range in a sheet

xml xml

col_offset, row_offset positioning parameters

... additional arguments

Returns: The wbWorkbook object

Method add_mschart(): Add mschart chart to the workbook

Usage:

```
wbWorkbook$add_mschart(
  sheet = current_sheet(),
  dims = NULL,
  graph,
  col_offset = 0,
  row_offset = 0,
  ...
)
```

Arguments:

sheet The name of the sheet

dims the dimensions where the sheet will appear

graph mschart graph

col_offset, row_offset offsets for column and row

... additional arguments

Returns: The wbWorkbook object

Method add_form_control(): Add form control to workbook

Usage:

```
wbWorkbook$add_form_control(
  sheet = current_sheet(),
  dims = "A1",
  type = c("Checkbox", "Radio", "Drop"),
```

```
  text = NULL,
  link = NULL,
  range = NULL,
  checked = FALSE
)
```

Arguments:

`sheet` The name of the sheet

`dims` Cell range in a sheet

`type` type

`text` text

`link` link

`range` range

`checked` checked

Returns: The `wbWorkbook` object, invisibly

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,
  lock_structure = FALSE,
  lock_windows = FALSE,
  type = 1,
  file_sharing = FALSE,
  username = unname(Sys.info()["user"]),
  read_only_recommended = FALSE,
  ...
)
```

Arguments:

`protect` protect

`password` password

`lock_structure` lock_structure

`lock_windows` lock_windows

`type` type

`file_sharing` file_sharing

`username` username

`read_only_recommended` read_only_recommended

`...` additional arguments

Returns: The `wbWorkbook` object, invisibly

Method `protect_worksheet()`**:** protect worksheet

Usage:

```
wbWorkbook$protect_worksheet(
  sheet = current_sheet(),
  protect = TRUE,
  password = NULL,
  properties = NULL
)
```

Arguments:

`sheet` The name of the sheet

`protect` protect

`password` password

`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"`

Returns: The `wbWorkbook` object

Method `get_properties()`**:** Get properties of a workbook

Usage:

```
wbWorkbook$get_properties()
```

Method `set_properties()`**:** Set a property of a workbook

Usage:

```
wbWorkbook$set_properties(
  creator = NULL,
  title = NULL,
  subject = NULL,
  category = NULL,
  datetime_created = Sys.time(),
  modifier = NULL,
  keywords = NULL,
  comments = NULL,
  manager = NULL,
  company = NULL,
  custom = NULL
)
```

Arguments:

`creator` character vector of creators. Duplicated are ignored.

`title`, `subject`, `category`, `datetime_created`, `modifier`, `keywords`, `comments`, `manager`, `company`, `custom` A workbook property to set

Method `add_mips()`**:** add mips string

Usage:

```
wbWorkbook$add_mips(xml = NULL)
```

Arguments:

`xml` A mips string added to self$custom

Method `get_mips()`**:** get mips string

Usage:

```
wbWorkbook$get_mips(single_xml = TRUE, quiet = TRUE)
```

Arguments:

`single_xml` single_xml

`quiet` quiet

Method `set_creators()`**:** Set creator(s)

Usage:

```
wbWorkbook$set_creators(creators)
```

Arguments:

`creators` A character vector of creators to set. Duplicates are ignored.

Method `add_creators()`**:** Add creator(s)

Usage:

```
wbWorkbook$add_creators(creators)
```

Arguments:

`creators` A character vector of creators to add. Duplicates are ignored.

Method `remove_creators()`**:** Remove creator(s)

Usage:

```
wbWorkbook$remove_creators(creators)
```

Arguments:

`creators` A character vector of creators to remove. All duplicated are removed.

Method `set_last_modified_by()`**:** Change the last modified by

Usage:

```
wbWorkbook$set_last_modified_by(name, ...)
```

Arguments:

`name` A new value

`...` additional arguments

Returns: The `wbWorkbook` object, invisibly

Method `set_page_setup()`**:** set_page_setup() this function is intended to supersede page_setup(), but is not yet stable

Usage:

```
wbWorkbook$set_page_setup(
  sheet = current_sheet(),
  black_and_white = NULL,
  cell_comments = NULL,
  copies = NULL,
  draft = NULL,
  errors = NULL,
  first_page_number = NULL,
  id = NULL,
  page_order = NULL,
  paper_height = NULL,
  paper_width = NULL,
  hdpi = NULL,
  vdpi = NULL,
  use_first_page_number = NULL,
  use_printer_defaults = NULL,
  orientation = NULL,
  scale = NULL,
  left = 0.7,
  right = 0.7,
  top = 0.75,
  bottom = 0.75,
  header = 0.3,
  footer = 0.3,
  fit_to_width = FALSE,
  fit_to_height = FALSE,
  paper_size = NULL,
  print_title_rows = NULL,
  print_title_cols = NULL,
  summary_row = NULL,
  summary_col = NULL,
  tab_color = NULL,
  ...
)
```

Arguments:

`sheet` The name of the sheet

`black_and_white` black_and_white

`cell_comments` cell_comment

`copies` copies

`draft` draft

`errors` errors

`first_page_number` first_page_number

`id` id

`page_order` page_order

`paper_height`, `paper_width` paper size

`hdpi`, `vdpi` horizontal and vertical dpi

`use_first_page_number` use_first_page_number

`use_printer_defaults` use_printer_defaults

`orientation` orientation

`scale` scale

`left` left

`right` right

`top` top

`bottom` bottom

`header` header

`footer` footer

`fit_to_width` fitToWidth

`fit_to_height` fitToHeight

`paper_size` paperSize

`print_title_rows` printTitleRows

`print_title_cols` printTitleCols

`summary_row` summaryRow

`summary_col` summaryCol

`tab_color` tabColor

`...` additional arguments

Returns: The `wbWorkbook` object, invisibly

Method `page_setup()`**:** page_setup()

Usage:

```
wbWorkbook$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,
  fit_to_width = FALSE,
  fit_to_height = FALSE,
  paper_size = NULL,
  print_title_rows = NULL,
  print_title_cols = NULL,
  summary_row = NULL,
  summary_col = NULL,
  ...
)
```

Arguments:

`sheet` The name of the sheet

`orientation` orientation

`scale` scale

`left` left

`right` right

`top` top

`bottom` bottom

`header` header

`footer` footer

`fit_to_width` fitToWidth

`fit_to_height` fitToHeight

`paper_size` paperSize

`print_title_rows` printTitleRows

`print_title_cols` printTitleCols

`summary_row` summaryRow

`summary_col` summaryCol

`...` additional arguments

Returns: The `wbWorkbook` object, invisibly

Method `set_header_footer()`**:** Sets headers and footers

Usage:

```
wbWorkbook$set_header_footer(
  sheet = current_sheet(),
  header = NULL,
  footer = NULL,
  even_header = NULL,
  even_footer = NULL,
  first_header = NULL,
  first_footer = NULL,
  ...
)
```

Arguments:

`sheet` The name of the sheet

`header` header

`footer` footer

`even_header` evenHeader

`even_footer` evenFooter

`first_header` firstHeader

`first_footer` firstFooter

`...` additional arguments

Returns: The `wbWorkbook` object, invisibly

Method `get_tables()`**:** get tables

Usage:

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

Arguments:

`sheet` The name of the sheet

Returns: The sheet tables. `character()` if empty

Method `remove_tables()`**:** remove tables

Usage:

```
wbWorkbook$remove_tables(sheet = current_sheet(), table, remove_data = TRUE)
```

Arguments:

`sheet` The name of the sheet
`table` table
`remove_data` removes the data as well

Returns: The `wbWorkbook` object

Method `add_filter()`**:** add filters

Usage:

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

Arguments:

`sheet` The name of the sheet
`rows` rows
`cols` cols

Returns: The `wbWorkbook` object

Method `remove_filter()`**:** remove filters

Usage:

```
wbWorkbook$remove_filter(sheet = current_sheet())
```

Arguments:

`sheet` The name of the sheet

Returns: The `wbWorkbook` object

Method `set_grid_lines()`**:** grid lines

Usage:

```
wbWorkbook$set_grid_lines(sheet = current_sheet(), show = FALSE, print = show)
```

Arguments:

`sheet` The name of the sheet
`show` show
`print` print

Returns: The `wbWorkbook` object

Method `grid_lines()`**:** grid lines

Usage:

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

Arguments:

`sheet` The name of the sheet

`show` show

`print` print

Returns: The `wbWorkbook` object

Method `add_named_region()`**:** add a named region

Usage:

```
wbWorkbook$add_named_region(
  sheet = current_sheet(),
  dims = "A1",
  name,
  local_sheet = FALSE,
  overwrite = FALSE,
  comment = NULL,
  hidden = NULL,
  custom_menu = NULL,
  description = NULL,
  is_function = NULL,
  function_group_id = NULL,
  help = NULL,
  local_name = NULL,
  publish_to_server = NULL,
  status_bar = NULL,
  vb_procedure = NULL,
  workbook_parameter = NULL,
  xml = NULL,
  ...
)
```

Arguments:

`sheet` The name of the sheet

`dims` Cell range in a sheet

`name` name

`local_sheet` local_sheet

`overwrite` overwrite

`comment` comment

`hidden` hidden

`custom_menu` custom_menu

`description` description

`is_function` function

`function_group_id` function group id

`help` help

`local_name` localName

publish_to_server publish to server
status_bar status bar
vb_procedure vb procedure
workbook_parameter workbookParameter
xml xml
... additional arguments

Returns: The wbWorkbook object

Method get_named_regions(): get named regions in a workbook

Usage:

```
wbWorkbook$get_named_regions(tables = FALSE, x = NULL)
```

Arguments:

tables Return tables as well?
x Not used.

Returns: A data.frame of named regions

Method remove_named_region(): remove a named region

Usage:

```
wbWorkbook$remove_named_region(sheet = current_sheet(), name = NULL)
```

Arguments:

sheet The name of the sheet
name name

Returns: The wbWorkbook object

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()
```

Returns: Returns sheet visibility

Method set_sheet_visibility(): Set sheet visibility

Usage:

```
wbWorkbook$set_sheet_visibility(sheet = current_sheet(), value)
```

Arguments:

sheet The name of the sheet
value value

Returns: The wbWorkbook object

Method add_page_break(): Add a page break

Usage:

```
wbWorkbook$add_page_break(sheet = current_sheet(), row = NULL, col = NULL)
```

Arguments:

sheet The name of the sheet

row row

col col

Returns: The wbWorkbook object

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

Usage:

```
wbWorkbook$clean_sheet(
  sheet = current_sheet(),
  dims = NULL,
  numbers = TRUE,
  characters = TRUE,
  styles = TRUE,
  merged_cells = TRUE
)
```

Arguments:

sheet The name of the sheet

dims Cell range in a 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_color(hex = "FF000000"),
  left_color = wb_color(hex = "FF000000"),
  right_color = wb_color(hex = "FF000000"),
  top_color = wb_color(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` The name of the sheet

`dims` dimensions on the worksheet e.g. "A1", "A1:A5", "A1:H5"

`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

`...` additional arguments

Returns: The `wbWorkbook`, invisibly

Method `add_fill()`**:** provide simple fill function

Usage:

```
wbWorkbook$add_fill(
  sheet = current_sheet(),
  dims = "A1",
  color = wb_color(hex = "FFFFFF00"),
  pattern = "solid",
  gradient_fill = "",
  every_nth_col = 1,
  every_nth_row = 1,
  ...
)
```

Arguments:

`sheet` The name of the sheet

`dims` Cell range in a sheet

`color` the colors to apply, e.g. yellow: wb_color(hex = "FFFFFF00")

`pattern` various default "none" but others are possible: "solid", "mediumGray", "darkGray", "lightGray", "darkHorizontal", "darkVertical", "darkDown", "darkUp", "darkGrid", "darkTrellis", "lightHorizontal", "lightVertical", "lightDown", "lightUp", "lightGrid", "lightTrellis", "gray125", "gray0625"

`gradient_fill` a gradient fill xml pattern.

`every_nth_col` which col should be filled

`every_nth_row` which row should be filled

`...` additional arguments

Returns: The `wbWorksheetObject`, invisibly

Method `add_font()`**:** provide simple font function

Usage:

```
wbWorkbook$add_font(
  sheet = current_sheet(),
  dims = "A1",
  name = "Aptos Narrow",
  color = wb_color(hex = "FF000000"),
  size = "11",
  bold = "",
  italic = "",
  outline = "",
  strike = "",
  underline = "",
  charset = "",
  condense = "",
  extend = "",
  family = "",
  scheme = "",
  shadow = "",
  vert_align = "",
  ...
)
```

Arguments:

`sheet` The name of the sheet

`dims` Cell range in a sheet

`name` font name: default "Aptos Narrow"

`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

`vert_align` vertical alignment

`...` additional arguments

Returns: The `wbWorkbook`, invisibly

Method `add_numfmt()`**:** provide simple number format function

Usage:

```
wbWorkbook$add_numfmt(sheet = current_sheet(), dims = "A1", numfmt)
```

Arguments:

`sheet` The name of the sheet

`dims` Cell range in a sheet

`numfmt` number format id or a character of the format

Returns: The `wbWorksheetObject`, invisibly

Method `add_cell_style()`**:** provide simple cell style format function

Usage:

```
wbWorkbook$add_cell_style(
  sheet = current_sheet(),
  dims = "A1",
  apply_alignment = NULL,
  apply_border = NULL,
  apply_fill = NULL,
  apply_font = NULL,
  apply_number_format = NULL,
  apply_protection = NULL,
  border_id = NULL,
  ext_lst = NULL,
  fill_id = NULL,
  font_id = NULL,
  hidden = NULL,
  horizontal = NULL,
  indent = NULL,
  justify_last_line = NULL,
  locked = NULL,
  num_fmt_id = NULL,
  pivot_button = NULL,
  quote_prefix = NULL,
  reading_order = NULL,
  relative_indent = NULL,
  shrink_to_fit = NULL,
  text_rotation = NULL,
  vertical = NULL,
  wrap_text = NULL,
  xf_id = NULL,
  ...
)
```

Arguments:

`sheet` The name of the sheet

`dims` Cell range in a sheet

`apply_alignment` logical apply alignment

`apply_border` logical apply border

`apply_fill` logical apply fill

`apply_font` logical apply font

`apply_number_format` logical apply number format

`apply_protection` logical apply protection
`border_id` border ID to apply
`ext_lst` extension list something like `<extLst>...</extLst>`
`fill_id` fill ID to apply
`font_id` font ID to apply
`hidden` logical cell is hidden
`horizontal` align content horizontal ('left', 'center', 'right')
`indent` logical indent content
`justify_last_line` logical justify last line
`locked` logical cell is locked
`num_fmt_id` number format ID to apply
`pivot_button` unknown
`quote_prefix` unknown
`reading_order` reading order left to right
`relative_indent` relative indentation
`shrink_to_fit` logical shrink to fit
`text_rotation` degrees of text rotation
`vertical` vertical alignment of content ('top', 'center', 'bottom')
`wrap_text` wrap text in cell
`xf_id` xf ID to apply
`...` additional arguments

Returns: The `wbWorkbook` object, invisibly

Method `get_cell_style()`**:** get sheet style

Usage:

```
wbWorkbook$get_cell_style(sheet = current_sheet(), dims)
```

Arguments:

`sheet` The name of the sheet
`dims` Cell range in a sheet

Returns: a character vector of cell styles

Method `set_cell_style()`**:** set sheet style

Usage:

```
wbWorkbook$set_cell_style(sheet = current_sheet(), dims, style)
```

Arguments:

`sheet` The name of the sheet
`dims` Cell range in a sheet
`style` style

Returns: The `wbWorksheetObject`, invisibly

Method `set_cell_style_across()`**:** set style across columns and/or rows

Usage:

```
wbWorkbook$set_cell_style_across(
  sheet = current_sheet(),
  style,
  cols = NULL,
  rows = NULL
)
```

Arguments:

`sheet` sheet

`style` style

`cols` cols

`rows` rows

Returns: The `wbWorkbook` object

Method `add_named_style()`**:** set sheet style

Usage:

```
wbWorkbook$add_named_style(
  sheet = current_sheet(),
  dims = "A1",
  name = "Normal",
  font_name = NULL,
  font_size = NULL
)
```

Arguments:

`sheet` The name of the sheet

`dims` Cell range in a sheet

`name` name

`font_name`, `font_size` optional else the default of the theme

Returns: The `wbWorkbook`, invisibly

Method `add_dxfs_style()`**:** create dxfs style These styles are used with conditional formatting and custom table styles

Usage:

```
wbWorkbook$add_dxfs_style(
  name,
  font_name = NULL,
  font_size = NULL,
  font_color = NULL,
  num_fmt = NULL,
  border = NULL,
  border_color = wb_color(getOption("openxlsx2.borderColor", "black")),
  border_style = getOption("openxlsx2.borderStyle", "thin"),
  bg_fill = NULL,
  gradient_fill = NULL,
  text_bold = NULL,
  text_italic = NULL,
```

```
  text_underline = NULL,
  ...
)
```

Arguments:

`name` the style name

`font_name` the font name

`font_size` the font size

`font_color` the font color (a `wb_color()` object)

`num_fmt` the number format

`border` logical if borders are applied

`border_color` the border color

`border_style` the border style

`bg_fill` any background fill

`gradient_fill` any gradient fill

`text_bold` logical if text is bold

`text_italic` logical if text is italic

`text_underline` logical if text is underlined

`...` additional arguments passed to `create_dxfs_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 name of the sheet

`sparklines` sparkline created by `create_sparkline()`

Method `add_ignore_error()`**:** Ignore error on worksheet

Usage:

```
wbWorkbook$add_ignore_error(
  sheet = current_sheet(),
  dims = "A1",
  calculated_column = FALSE,
  empty_cell_reference = FALSE,
  eval_error = FALSE,
  formula = FALSE,
```

```
  formula_range = FALSE,
  list_data_validation = FALSE,
  number_stored_as_text = FALSE,
  two_digit_text_year = FALSE,
  unlocked_formula = FALSE,
  ...
)
```

Arguments:

`sheet` The name of the sheet

`dims` Cell range in a sheet

`calculated_column` calculatedColumn

`empty_cell_reference` emptyCellReference

`eval_error` evalError

`formula` formula

`formula_range` formulaRange

`list_data_validation` listDataValidation

`number_stored_as_text` numberStoredAsText

`two_digit_text_year` twoDigitTextYear

`unlocked_formula` unlockedFormula

`...` additional arguments

Method `set_sheetview()`**:** add sheetview

Usage:

```
wbWorkbook$set_sheetview(
  sheet = current_sheet(),
  color_id = NULL,
  default_grid_color = NULL,
  right_to_left = NULL,
  show_formulas = NULL,
  show_grid_lines = NULL,
  show_outline_symbols = NULL,
  show_row_col_headers = NULL,
  show_ruler = NULL,
  show_white_space = NULL,
  show_zeros = NULL,
  tab_selected = NULL,
  top_left_cell = NULL,
  view = NULL,
  window_protection = NULL,
  workbook_view_id = NULL,
  zoom_scale = NULL,
  zoom_scale_normal = NULL,
  zoom_scale_page_layout_view = NULL,
  zoom_scale_sheet_layout_view = NULL,
  ...
)
```

Arguments:

`sheet` The name of the sheet

`color_id, default_grid_color` Integer: A color, default is 64

`right_to_left` Logical: if TRUE column ordering is right to left

`show_formulas` Logical: if TRUE cell formulas are shown

`show_grid_lines` Logical: if TRUE the worksheet grid is shown

`show_outline_symbols` Logical: if TRUE outline symbols are shown

`show_row_col_headers` Logical: if TRUE row and column headers are shown

`show_ruler` Logical: if TRUE a ruler is shown in page layout view

`show_white_space` Logical: if TRUE margins are shown in page layout view

`show_zeros` Logical: if FALSE cells containing zero are shown blank if !showFormulas

`tab_selected` Integer: zero vector indicating the selected tab

`top_left_cell` Cell: the cell shown in the top left corner / or top right with rightToLeft

`view` View: "normal", "pageBreakPreview" or "pageLayout"

`window_protection` Logical: if TRUE the panes are protected

`workbook_view_id` integer: Pointing to some other view inside the workbook

`zoom_scale, zoom_scale_normal, zoom_scale_page_layout_view, zoom_scale_sheet_layout_view` Integer: the zoom scale should be between 10 and 400. These are values for current, normal etc.

`...` additional arguments

Returns: The `wbWorksheetObject`, invisibly

Method `add_person()`**:** add person to workbook

Usage:

```
wbWorkbook$add_person(
  name = NULL,
  id = NULL,
  user_id = NULL,
  provider_id = "None"
)
```

Arguments:

`name` name

`id` id

`user_id` user_id

`provider_id` provider_id

Method `get_person()`**:** description get person

Usage:

```
wbWorkbook$get_person(name = NULL)
```

Arguments:

`name` name

Method `get_active_sheet()`**:** description get active sheet

Usage:

```
wbWorkbook$get_active_sheet()
```

Method `set_active_sheet()`**:** description set active sheet

Usage:

```
wbWorkbook$set_active_sheet(sheet = current_sheet())
```

Arguments:

`sheet` The name of the sheet

Method `get_selected()`**:** description get selected sheets

Usage:

```
wbWorkbook$get_selected()
```

Method `set_selected()`**:** set selected sheet

Usage:

```
wbWorkbook$set_selected(sheet = current_sheet())
```

Arguments:

`sheet` The name of the sheet

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.

`wb_add_border` *Modify borders in a cell region of a worksheet*

Description

wb wrapper to create borders for cell regions.

Usage

```
wb_add_border(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  bottom_color = wb_color(hex = "FF000000"),
  left_color = wb_color(hex = "FF000000"),
  right_color = wb_color(hex = "FF000000"),
  top_color = wb_color(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	A wbWorkbook
sheet	A worksheet
dims	Cell range in the worksheet e.g. "A1", "A1:A5", "A1:H5"
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
...	additional arguments

See Also

create_border()

Other styles: wb_add_cell_style(), wb_add_fill(), wb_add_font(), wb_add_named_style(), wb_add_numfmt(), wb_cell_style

Examples

```
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_color(hex = "FF9400D3"), right_color = wb_color(hex = "FF4B0082"),
 top_color = wb_color(hex = "FF0000FF"), bottom_color = wb_color(hex = "FF00FF00"))
wb <- wb_add_border(wb, 1, dims = "A20:C23")
wb <- wb_add_border(wb, 1, dims = "B12:D14",
 left_color = wb_color(hex = "FFFFFF00"), right_color = wb_color(hex = "FFFF7F00"),
 bottom_color = wb_color(hex = "FFFF0000"))
wb <- wb_add_border(wb, 1, dims = "D28:E28")
```

```
# With chaining

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 = "G2:H3")
wb$add_border(1, dims = "G12:H13",
 left_color = wb_color(hex = "FF9400D3"), right_color = wb_color(hex = "FF4B0082"),
 top_color = wb_color(hex = "FF0000FF"), bottom_color = wb_color(hex = "FF00FF00"))
wb$add_border(1, dims = "A20:C23")
wb$add_border(1, dims = "B12:D14",
 left_color = wb_color(hex = "FFFFFF00"), right_color = wb_color(hex = "FFFF7F00"),
 bottom_color = wb_color(hex = "FFFF0000"))
wb$add_border(1, dims = "D28:E28")
# if (interactive()) wb$open()

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

---

wb_add_cell_style *Modify the style in a cell region*

Description

Add cell style to a cell region

Usage

```
wb_add_cell_style(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  apply_alignment = NULL,
  apply_border = NULL,
  apply_fill = NULL,
  apply_font = NULL,
  apply_number_format = NULL,
  apply_protection = NULL,
  border_id = NULL,
  ext_lst = NULL,
  fill_id = NULL,
```

```
  font_id = NULL,
  hidden = NULL,
  horizontal = NULL,
  indent = NULL,
  justify_last_line = NULL,
  locked = NULL,
  num_fmt_id = NULL,
  pivot_button = NULL,
  quote_prefix = NULL,
  reading_order = NULL,
  relative_indent = NULL,
  shrink_to_fit = NULL,
  text_rotation = NULL,
  vertical = NULL,
  wrap_text = NULL,
  xf_id = NULL,
  ...
)
```

Arguments

wb	a workbook
sheet	the worksheet
dims	the cell range
apply_alignment	logical apply alignment
apply_border	logical apply border
apply_fill	logical apply fill
apply_font	logical apply font
apply_number_format	logical apply number format
apply_protection	logical apply protection
border_id	border ID to apply
ext_lst	extension list something like <extLst>...</extLst>
fill_id	fill ID to apply
font_id	font ID to apply
hidden	logical cell is hidden
horizontal	align content horizontal ('left', 'center', 'right')
indent	logical indent content
justify_last_line	logical justify last line
locked	logical cell is locked
num_fmt_id	number format ID to apply

pivot_button	unknown
quote_prefix	unknown
reading_order	reading order left to right
relative_indent	relative indentation
shrink_to_fit	logical shrink to fit
text_rotation	degrees of text rotation
vertical	vertical alignment of content ('top', 'center', 'bottom')
wrap_text	wrap text in cell
xf_id	xf ID to apply
...	additional arguments

Value

The wbWorkbook object, invisibly

See Also

Other styles: wb_add_border(), wb_add_fill(), wb_add_font(), wb_add_named_style(), wb_add_numfmt(), wb_cell_style

Examples

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

wb %>%
  wb_add_cell_style(
    dims = "A1:K1",
    text_rotation = "45",
    horizontal = "center",
    vertical = "center",
    wrap_text = "1"
)
# Chaining
wb <- wb_workbook()$add_worksheet("S1")$add_data(x = mtcars)
wb$add_cell_style(dims = "A1:K1",
                  text_rotation = "45",
                  horizontal = "center",
                  vertical = "center",
                  wrap_text = "1")
```

wb_add_chartsheet *Add a chartsheet to a workbook*

Description

A chartsheet is a special type of sheet that handles charts output. You must add a chart to the sheet. Otherwise, this will break the workbook.

Usage

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

Arguments

wb	A Workbook object to attach the new chartsheet
sheet	A name for the new chartsheet
tab_color	Color of the sheet tab. A wb_color(), a valid color (belonging to grDevices::colors()) or a valid hex color beginning with "#".
zoom	The sheet zoom level, a numeric between 10 and 400 as a percentage. (A zoom value smaller than 10 will default to 10.)
visible	If FALSE, sheet is hidden else visible.
...	Additional arguments

See Also

wb_add_mschart()

Other workbook wrappers: base_font-wb, col_widths-wb, creators-wb, grouping-wb, row_heights-wb, wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_worksheet(), wb_base_colors, wb_clone_worksheet(), wb_copy_cells(), wb_freeze_pane(), wb_merge_cells(), wb_save(), wb_set_last_modified_by(), wb_workbook()

wb_add_chart_xml *Add a chart XML to a worksheet*

Description

Add a chart XML to a worksheet

Usage

```
wb_add_chart_xml(
  wb,
  sheet = current_sheet(),
  dims = NULL,
  xml,
  col_offset = 0,
  row_offset = 0,
  ...
)
```

Arguments

wb	a workbook
sheet	the sheet on which the graph will appear
dims	the dimensions where the sheet will appear
xml	chart xml
col_offset, row_offset	positioning
...	additional arguments

See Also

wb_add_drawing() wb_add_image() wb_add_mschart() wb_add_plot()

wb_add_comment *Add comment to worksheet*

Description

Add comment to worksheet

Usage

```
wb_add_comment(wb, sheet = current_sheet(), dims = "A1", comment, ...)

wb_get_comment(wb, sheet = current_sheet(), dims = NULL)

wb_remove_comment(wb, sheet = current_sheet(), dims = "A1", ...)
```

Arguments

wb	A workbook object
sheet	A worksheet of the workbook
dims	Optional row and column as spreadsheet dimension, e.g. "A1"
comment	A comment to apply to dims created by wb_comment(), a string or a fmt_txt() object
...	additional arguments

Details

If applying a comment with a string, it will use wb_comment() default values. If additional background colors are applied, RGB colors should be provided, either as hex code or with builtin R colors. The alpha channel is ignored.

Value

The Workbook object, invisibly.

See Also

wb_comment(), wb_add_thread()

Examples

```
wb <- wb_workbook()
wb$add_worksheet("Sheet 1")
# add a comment without author
c1 <- wb_comment(text = "this is a comment", author = "")
wb$add_comment(dims = "B10", comment = c1)
#' # Remove comment
wb$remove_comment(sheet = "Sheet 1", dims = "B10")
# Write another comment with author information
c2 <- wb_comment(text = "this is another comment", author = "Marco Polo", visible = TRUE)
wb$add_comment(sheet = 1, dims = "C10", comment = c2)
# Works with formatted text also.
formatted_text <- fmt_txt("bar", underline = TRUE)
wb$add_comment(dims = "B5", comment = formatted_text)
# With background color
wb$add_comment(dims = "B7", comment = formatted_text, color = wb_color("green"))
# With background image. File extension must be png or jpeg, not jpg?
tmp <- tempfile(fileext = ".png")
png(file = tmp, bg = "transparent")
plot(1:10)
rect(1, 5, 3, 7, col = "white")
dev.off()

c1 <- wb_comment(text = "this is a comment", author = "", visible = TRUE)
wb$add_comment(dims = "B12", comment = c1, file = tmp)
```

wb_add_conditional_formatting *Add conditional formatting to cells in a worksheet*

Description

Add conditional formatting to cells. You can find more details in `vignette("conditional-formatting")`.

Usage

```
wb_add_conditional_formatting(
  wb,
  sheet = current_sheet(),
  dims = NULL,
  rule = NULL,
  style = NULL,
  type = c("expression", "colorScale", "dataBar", "iconSet", "duplicatedValues",
    "uniqueValues", "containsErrors", "notContainsErrors", "containsBlanks",
   "notContainsBlanks", "containsText", "notContainsText", "beginsWith", "endsWith",
    "between", "topN", "bottomN"),
 params = list(showValue = TRUE, gradient = TRUE, border = TRUE, percent = FALSE, rank =
    5L),
  ...
)

wb_remove_conditional_formatting(
  wb,
  sheet = current_sheet(),
  dims = NULL,
  first = FALSE,
  last = FALSE
)
```

Arguments

wb	A Workbook object
sheet	A name or index of a worksheet
dims	A cell or cell range like "A1" or "A1:B2"
rule	The condition under which to apply the formatting. See **Examples**.
style	A name of a style to apply to those cells that satisfy the rule. See `wb_add_dxfs_style()` how to create one. The default style has `font_color = "FF9C0006"` and `bg_fill = "FFFFC7CE"`
type	The type of conditional formatting rule to apply. One of `"expression"`, `"colorScale"` or others mentioned in **Details**.
params	A list of additional parameters passed. See **Details** for more.

...	additional arguments
first	remove the first conditional formatting
last	remove the last conditional formatting

Details

openxml uses the alpha channel first then RGB, whereas the usual default is RGBA.

Conditional formatting type 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`

`[rule]`
: `NULL` or a `character` vector of valid colors of equal length to `styles`

`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` color gradient is removed. Default `TRUE`

`[params$border]`
: If `FALSE` the border around the database is hidden. Default `TRUE`

`duplicatedValues` / `uniqueValues` / `containsErrors [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

[params$percent]
If TRUE, uses percentage

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

[params$reverse]
If TRUE, the order is reversed. Default FALSE

[params$percent]
If TRUE, uses percentage

[params$iconSet]
Uses one of the implemented icon sets. Values must match the length of the icons in the set 3Arrows, 3ArrowsGray, 3Flags, 3Signs, 3Stars, 3Symbols, 3Symbols2, 3TrafficLights1, 3TrafficLights2, 3Triangles, 4Arrows, 4ArrowsGray, 4Rating, 4RedToBlack, 4TrafficLights, 5Arrows, 5ArrowsGray, 5Boxes, 5Quarters, 5Rating. The default is 3TrafficLights1.

See Also

Other worksheet content functions: col_widths-wb, filter-wb, grouping-wb, named_region-wb, row_heights-wb, wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_thread(), wb_freeze_pane(), wb_merge_cells()

Examples

```
wb <- wb_workbook()
wb$add_worksheet("a")
wb$add_data(x = 1:4, col_names = FALSE)
wb$add_conditional_formatting(dims = wb_dims(cols = "A", rows = 1:4), rule = ">2")
```

---

wb_add_data *Add data to a worksheet*

Description

Add data to worksheet with optional styling.

Usage

```
wb_add_data(
  wb,
  sheet = current_sheet(),
  x,
  dims = wb_dims(start_row, start_col),
  start_col = 1,
  start_row = 1,
  array = FALSE,
  col_names = TRUE,
  row_names = FALSE,
  with_filter = FALSE,
  name = NULL,
  sep = ", ",
  apply_cell_style = TRUE,
  remove_cell_style = FALSE,
  na.strings = na_strings(),
  inline_strings = TRUE,
  enforce = FALSE,
  ...
)
```

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.
dims	Spreadsheet cell range that will determine start_col and start_row: "A1", "A1:B2", "A:B"
start_col	A vector specifying the starting column to write x to.
start_row	A vector specifying the starting row to write x to.
array	A bool if the function written is of type array
col_names	If TRUE, column names of x are written.
row_names	If TRUE, the row names of x are written.
with_filter	If TRUE, add filters to the column name row. NOTE: can only have one filter per worksheet.
name	The name of a named region if specified.
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).
apply_cell_style	Should we write cell styles to the workbook
remove_cell_style	keep the cell style?

na.strings	Value used for replacing NA values from x. Default looks if options(openxlsx2.na.strings) is set. Otherwise na_strings() uses the special #N/A value within the workbook.
inline_strings	write characters as inline strings
enforce	enforce that selected dims is filled. For this to work, dims must match x
...	additional arguments

Details

Formulae written using wb_add_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.

Supported classes are data frames, matrices and vectors of various types and everything that can be converted into a data frame with as.data.frame(). Everything else that the user wants to write should either be converted into a vector or data frame or written in vector or data frame segments. This includes base classes such as table, which were coerced internally in the predecessor of this package.

Even vectors and data frames can consist of different classes. Many base classes are covered, though not all and far from all third-party classes. When data of an unknown class is written, it is handled with as.character(). It is not possible to write character nodes beginning with <r> or <r/>. Both are reserved for internal functions. If you need these. You have to wrap the input string in fmt_txt().

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

Functions wb_add_data() and wb_add_data_table() behave quite similar. The distinction is that the latter creates a table in the worksheet that can be used for different kind of formulas and can be sorted independently, though is less flexible than basic cell regions.

Value

A wbWorkbook, invisibly.

See Also

Other workbook wrappers: base_font-wb, col_widths-wb, creators-wb, grouping-wb, row_heights-wb, wb_add_chartsheet(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_worksheet(), wb_base_colors, wb_clone_worksheet(), wb_copy_cells(), wb_freeze_pane(), wb_merge_cells(), wb_save(), wb_set_last_modified_by(), wb_workbook()

Other worksheet content functions: col_widths-wb, filter-wb, grouping-wb, named_region-wb, row_heights-wb, wb_add_conditional_formatting(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_thread(), wb_freeze_pane(), wb_merge_cells()

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, start_col = 2, start_row = 3, row_names = 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, dims = "B32")

################################################################################
## 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", "openxlsx2_example.xlsx", package = "openxlsx2")
wb2 <- wb_load(xlsxFile)

# read dataset with inlinestr
wb_to_df(wb2)
wb2 <- wb2 %>% wb_add_data(sheet = 1, mtcars, dims = wb_dims(4, 4))
wb_to_df(wb2)
```

---

wb_add_data_table *Add a data table to a worksheet*

Description

Add data to a worksheet and format as an Excel table.

Usage

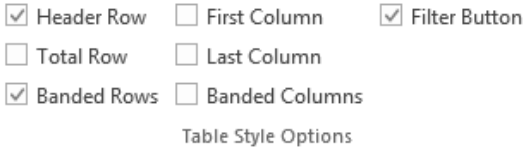
```
wb_add_data_table(
  wb,
  sheet = current_sheet(),
  x,
  dims = wb_dims(start_row, start_col),
  start_col = 1,
  start_row = 1,
  col_names = TRUE,
  row_names = FALSE,
  table_style = "TableStyleLight9",
  table_name = NULL,
  with_filter = TRUE,
  sep = ", ",
  first_column = FALSE,
  last_column = FALSE,
  banded_rows = TRUE,
  banded_cols = FALSE,
  apply_cell_style = TRUE,
  remove_cell_style = FALSE,
  na.strings = na_strings(),
  inline_strings = TRUE,
  total_row = FALSE,
  ...
)
```

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
dims	Spreadsheet cell range that will determine start_col and start_row: "A1", "A1:B2", "A:B"
start_col	A vector specifying the starting column to write x to.
start_row	A vector specifying the starting row to write x to.
col_names	If TRUE, column names of x are written.
row_names	If TRUE, the row names of x are written.
table_style	Any table style name or "none" (see vignette("openxlsx2_style_manual"))
table_name	Name of table in workbook. The table name must be unique.
with_filter	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:



first_column logical. If TRUE, the first column is bold.

last_column logical. If TRUE, the last column is bold.

banded_rows logical. If TRUE, rows are color banded.

banded_cols logical. If TRUE, the columns are color banded.

apply_cell_style
Should we write cell styles to the workbook

remove_cell_style
keep the cell style?

na.strings Value used for replacing NA values from x. Default looks if `options(openxlsx2.na.strings)` is set. Otherwise `na_strings()` uses the special `#N/A` value within the workbook.

inline_strings write characters as inline strings

total_row logical. With the default FALSE no total row is added.

... additional arguments

Details

Formulae written using `wb_add_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.

Supported classes are data frames, matrices and vectors of various types and everything that can be converted into a data frame with `as.data.frame()`. Everything else that the user wants to write should either be converted into a vector or data frame or written in vector or data frame segments. This includes base classes such as `table`, which were coerced internally in the predecessor of this package.

Even vectors and data frames can consist of different classes. Many base classes are covered, though not all and far from all third-party classes. When data of an unknown class is written, it is handled with `as.character()`. It is not possible to write character nodes beginning with `<r>` or `<r/>`. Both are reserved for internal functions. If you need these. You have to wrap the input string in `fmt_txt()`.

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

Functions wb_add_data() and wb_add_data_table() behave quite similar. The distinction is that the latter creates a table in the worksheet that can be used for different kind of formulas and can be sorted independently, though is less flexible than basic cell regions.

Modify total row argument

It is possible to further tweak the total row. In addition to the default FALSE possible values are TRUE (the xlsx file will create column sums each variable).

In addition it is possible to tweak this further using a character string with one of the following functions for each variable: "average", "count", "countNums", "max", "min", "stdDev", "sum", "var". It is possible to leave the cell empty "none" or to create a text input using a named character with name text like: c(text = "Total"). It's also possible to pass other spreadsheet software functions if they return a single value and hence "SUM" would work too.

See Also

Other worksheet content functions: col_widths-wb, filter-wb, grouping-wb, named_region-wb, row_heights-wb, wb_add_conditional_formatting(), wb_add_data(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_thread(), wb_freeze_pane(), wb_merge_cells()

Other workbook wrappers: base_font-wb, col_widths-wb, creators-wb, grouping-wb, row_heights-wb, wb_add_chartsheet(), wb_add_data(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_worksheet(), wb_base_colors, wb_clone_worksheet(), wb_copy_cells(), wb_freeze_pane(), wb_merge_cells(), wb_save(), wb_set_last_modified_by(), wb_workbook()

Examples

```
wb <- wb_workbook()$add_worksheet()$
  add_data_table(
    x = as.data.frame(USPersonalExpenditure),
    row_names = TRUE,
    total_row = c(text = "Total", "none", "sum", "sum", "sum", "SUM")
  )
```

---

wb_add_data_validation

Add data validation to cells in a worksheet

Description

Add Excel data validation to cells

Usage

```
wb_add_data_validation(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  type,
```

```
  operator,
  value,
  allow_blank = TRUE,
  show_input_msg = TRUE,
  show_error_msg = TRUE,
  error_style = NULL,
  error_title = NULL,
  error = NULL,
  prompt_title = NULL,
  prompt = NULL,
  ...
)
```

Arguments

wb	A Workbook object
sheet	A name or index of a worksheet
dims	A cell dimension ("A1" or "A1:B2")
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)
allow_blank	logical
show_input_msg	logical
show_error_msg	logical
error_style	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)
error_title	The error title
error	The error text
prompt_title	The prompt title
prompt	The prompt text
...	additional arguments

Examples

```
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,
  dims = "A2:C31", type = "whole",
  operator = "between", value = c(1, 9)
)
```

```
wb$add_data_validation(1,
  dims = "E2:E31", 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, dims = "A2:A12", type = "date",
  operator = "greaterThanOrEqual", value = as.Date("2016-01-01")
)
wb$add_data_validation(2,
  dims = "B2:B12", type = "time",
  operator = "between", value = df$t[c(4, 8)]
)


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

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$add_data_validation(1, dims = "A2:A31", type = "list", value = "'Sheet 2'!$A$1:$A$10")
```

---

wb_add_drawing *Add drawings to a worksheet*

Description

Add drawings to a worksheet. This requires the rvg package.

Usage

```
wb_add_drawing(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  xml,
  col_offset = 0,
  row_offset = 0,
  ...
)
```

Arguments

wb	A wbWorkbook
sheet	A sheet in the workbook
dims	The dimension where the drawing is added.
xml	the drawing xml as character or file
col_offset, row_offset	offsets for column and row
...	additional arguments

See Also

wb_add_chart_xml() wb_add_image() wb_add_mschart() wb_add_plot()

Examples

```
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_dxfs_style *Set a dxfs styling for the workbook*

Description

These styles are used with conditional formatting and custom table styles.

Usage

```
wb_add_dxfs_style(
  wb,
  name,
  font_name = NULL,
  font_size = NULL,
  font_color = NULL,
  num_fmt = NULL,
```

```
  border = NULL,
  border_color = wb_color(getOption("openxlsx2.borderColor", "black")),
  border_style = getOption("openxlsx2.borderStyle", "thin"),
  bg_fill = NULL,
  gradient_fill = NULL,
  text_bold = NULL,
  text_italic = NULL,
  text_underline = NULL,
  ...
)
```

Arguments

wb	A Workbook object.
name	the style name
font_name	the font name
font_size	the font size
font_color	the font color (a wb_color() object)
num_fmt	the number format
border	logical if borders are applied
border_color	the border color
border_style	the border style
bg_fill	any background fill
gradient_fill	any gradient fill
text_bold	logical if text is bold
text_italic	logical if text is italic
text_underline	logical if text is underlined
...	additional arguments passed to create_dxfs_style()

Value

The Workbook object, invisibly

See Also

Other workbook styling functions: base_font-wb, wb_add_style(), wb_base_colors

Examples

```
wb <- wb_workbook() %>%
  wb_add_worksheet() %>%
  wb_add_dxfs_style(
   name = "nay",
   font_color = wb_color(hex = "FF9C0006"),
   bg_fill = wb_color(hex = "FFFFC7CE")
  )
```

wb_add_fill *Modify the background fill color in a cell region*

Description

Add fill to a cell region.

Usage

```
wb_add_fill(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  color = wb_color(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_color(hex = "FFFFFF00")
pattern	various default "none" but others are possible: "solid", "mediumGray", "darkGray", "lightGray", "darkHorizontal", "darkVertical", "darkDown", "darkUp", "darkGrid", "darkTrellis", "lightHorizontal", "lightVertical", "lightDown", "lightUp", "lightGrid", "lightTrellis", "gray125", "gray0625"
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 wbWorkbook object, invisibly

See Also

Other styles: wb_add_border(), wb_add_cell_style(), wb_add_font(), wb_add_named_style(), wb_add_numfmt(), wb_cell_style

Examples

```
wb <- wb_workbook() %>% wb_add_worksheet("S1") %>% wb_add_data("S1", mtcars)
wb <- wb %>% wb_add_fill("S1", dims = "D5:J23", color = wb_color(hex = "FFFFFF00"))
wb <- wb %>% wb_add_fill("S1", dims = "B22:D27", color = wb_color(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_font *Modify font in a cell region*

Description

Modify the font in a cell region with more precision You can specify the font in a cell with other cell styling functions, but wb_add_font() gives you more control.

Usage

```
wb_add_font(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  name = "Aptos Narrow",
  color = wb_color(hex = "FF000000"),
  size = "11",
  bold = "",
  italic = "",
  outline = "",
  strike = "",
  underline = "",
  charset = "",
  condense = "",
  extend = "",
  family = "",
  scheme = "",
  shadow = "",
  vert_align = "",
```

```
  ...
)
```

Arguments

wb	A Workbook object
sheet	the worksheet
dims	the cell range
name	Font name: default "Aptos Narrow"
color	An object created by wb_color()
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
vert_align	vertical alignment
...	...

Details

wb_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

A wbWorkbook, invisibly

See Also

Other styles: wb_add_border(), wb_add_cell_style(), wb_add_fill(), wb_add_named_style(), wb_add_numfmt(), wb_cell_style

Examples

```
 wb <- wb_workbook() %>% wb_add_worksheet("S1") %>% wb_add_data("S1", mtcars)
 wb %>% wb_add_font("S1", "A1:K1", name = "Arial", color = wb_color(theme = "4"))
# With chaining
 wb <- wb_workbook()$add_worksheet("S1")$add_data("S1", mtcars)
 wb$add_font("S1", "A1:K1", name = "Arial", color = wb_color(theme = "4"))
```

wb_add_formula *Add a formula to a cell range in a worksheet*

Description

This function can be used to add a formula to a worksheet. In `wb_add_formula()`, you can provide the formula as a character vector.

Usage

```
wb_add_formula(
  wb,
  sheet = current_sheet(),
  x,
  dims = wb_dims(start_row, start_col),
  start_col = 1,
  start_row = 1,
  array = FALSE,
  cm = FALSE,
  apply_cell_style = TRUE,
  remove_cell_style = FALSE,
  enforce = FALSE,
  ...
)
```

Arguments

`wb`	A Workbook object containing a worksheet.
`sheet`	The worksheet to write to. (either as index or name)
`x`	A formula as character vector.
`dims`	Spreadsheet dimensions that will determine where `x` spans: "A1", "A1:B2", "A:B"
`start_col`	A vector specifying the starting column to write to.
`start_row`	A vector specifying the starting row to write to.
`array`	A bool if the function written is of type array
`cm`	A special kind of array function that hides the curly braces in the cell. Add this, if you see "@" inserted into your formulas.
`apply_cell_style`	Should we write cell styles to the workbook?
`remove_cell_style`	Should we keep the cell style?
`enforce`	enforce dims
`...`	additional arguments

Details

Currently, the local translations of formulas are not supported. Only the English functions work.

The examples below show a small list of possible formulas:

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

It is possible to pass vectors to `x`. If `x` is an array formula, it will take `dims` as a reference. For some formulas, the result will span multiple cells (see the `MMULT()` example below). For this type of formula, the output range must be known a priori and passed to `dims`, otherwise only the value of the first cell will be returned. This type of formula, whose result extends over several cells, is only possible with single strings. If a vector is passed, it is only possible to return individual cells.

Value

The workbook, invisibly.

See Also

Other workbook wrappers: `base_font-wb`, `col_widths-wb`, `creators-wb`, `grouping-wb`, `row_heights-wb`, `wb_add_chartsheet()`, `wb_add_data()`, `wb_add_data_table()`, `wb_add_pivot_table()`, `wb_add_slicer()`, `wb_add_worksheet()`, `wb_base_colors`, `wb_clone_worksheet()`, `wb_copy_cells()`, `wb_freeze_pane()`, `wb_merge_cells()`, `wb_save()`, `wb_set_last_modified_by()`, `wb_workbook()`

Other worksheet content functions: `col_widths-wb`, `filter-wb`, `grouping-wb`, `named_region-wb`, `row_heights-wb`, `wb_add_conditional_formatting()`, `wb_add_data()`, `wb_add_data_table()`, `wb_add_pivot_table()`, `wb_add_slicer()`, `wb_add_thread()`, `wb_freeze_pane()`, `wb_merge_cells()`

Examples

```
wb <- wb_workbook()$add_worksheet()
wb$add_data(dims = wb_dims(rows = 1, cols = 1:3), x = c(4, 5, 8))

# calculate the sum of elements.
wb$add_formula(dims = "D1", x = "SUM(A1:C1)")

# array formula with result spanning over multiple cells
mm <- matrix(1:4, 2, 2)

wb$add_worksheet()$
 add_data(x = mm, dims = "A1:B2", col_names = FALSE)$
 add_data(x = mm, dims = "A4:B5", col_names = FALSE)$
 add_formula(x = "MMULT(A1:B2, A4:B5)", dims = "A7:B8", array = TRUE)
```

wb_add_form_control *Add a checkbox, radio button or drop menu to a cell in a worksheet*

Description

You can add Form Control to a cell. The three supported types are a Checkbox, a Radio button, or a Drop menu.

Usage

```
wb_add_form_control(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  type = c("Checkbox", "Radio", "Drop"),
  text = NULL,
  link = NULL,
  range = NULL,
  checked = FALSE
)
```

Arguments

wb	A Workbook object
sheet	A worksheet of the workbook
dims	A single cell as spreadsheet dimension, e.g. "A1".
type	A type "Checkbox" (the default), "Radio" a radio button or "Drop" a drop down menu
text	A text to be shown next to the Checkbox or radio button (optional)
link	A cell range to link to
range	A cell range used as input
checked	A logical indicating if the Checkbox or Radio button is checked

Value

The wbWorkbook object, invisibly.

Examples

```
wb <- wb_workbook() %>% wb_add_worksheet() %>%
  wb_add_form_control()
# Add
wb$add_form_control(dims = "C5", type = "Radio", checked = TRUE)
```

`wb_add_ignore_error` *Ignore error types on worksheet*

Description

This function allows to hide / ignore certain types of errors shown in a worksheet.

Usage

```
wb_add_ignore_error(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  calculated_column = FALSE,
  empty_cell_reference = FALSE,
  eval_error = FALSE,
  formula = FALSE,
  formula_range = FALSE,
  list_data_validation = FALSE,
  number_stored_as_text = FALSE,
  two_digit_text_year = FALSE,
  unlocked_formula = FALSE,
  ...
)
```

Arguments

`wb`	A workbook
`sheet`	A sheet name or index.
`dims`	Cell range to ignore the error
`calculated_column`	calculatedColumn
`empty_cell_reference`	emptyCellReference
`eval_error`	evalError
`formula`	formula
`formula_range`	formulaRange
`list_data_validation`	listDataValidation
`number_stored_as_text`	If `TRUE`, will not display the error if numbers are stored as text.
`two_digit_text_year`	twoDigitTextYear
`unlocked_formula`	unlockedFormula
`...`	additional arguments

Value

The wbWorkbook object, invisibly.

---

wb_add_image *Insert an image into a worksheet*

Description

Insert an image into a worksheet

Usage

```
wb_add_image(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  file,
  width = 6,
  height = 3,
  row_offset = 0,
  col_offset = 0,
  units = "in",
  dpi = 300,
  ...
)
```

Arguments

wb	A workbook object
sheet	A name or index of a worksheet
dims	Dimensions where to plot. Default absolute anchor, single cell (eg. "A1") oneCellAnchor, cell range (eg. "A1:D4") twoCellAnchor
file	An image file. Valid file types are: "jpeg", "png", "bmp"
width	Width of figure.
height	Height of figure.
row_offset	offset vector for one or two cell anchor within cell (row)
col_offset	offset vector for one or two cell anchor within cell (column)
units	Units of width and height. Can be "in", "cm" or "px"
dpi	Image resolution used for conversion between units.
...	additional arguments

See Also

wb_add_chart_xml() wb_add_drawing() wb_add_mschart() wb_add_plot()

Examples

```
img <- system.file("extdata", "einstein.jpg", package = "openxlsx2")

wb <- wb_workbook()$
  add_worksheet()$
  add_image("Sheet 1", dims = "C5", file = img, width = 6, height = 5)$
  add_worksheet()$
  add_image(dims = "B2", file = img)$
  add_worksheet()$
  add_image(dims = "G3", file = img, width = 15, height = 12, units = "cm")
```

---

wb_add_mips *wb get and apply MIP section*

Description

Read sensitivity labels from files and apply them to workbooks

Usage

```
wb_add_mips(wb, xml = NULL)

wb_get_mips(wb, single_xml = TRUE, quiet = TRUE)
```

Arguments

wb	a workbook
xml	a mips string obtained from wb_get_mips() or a global option "openxlsx2.mips_xml_string"
single_xml	option to define if the string should be exported as single string. helpful if storing as option is desired.
quiet	option to print a MIP section name. This is not always a human readable string.

Details

The MIP section is a special user-defined XML section that is used to create sensitivity labels in workbooks. It consists of a series of XML property nodes that define the sensitivity label. This XML string cannot be created and it is necessary to first load a workbook with a suitable sensitivity label. Once the workbook is loaded, the string `fmips <- wb_get_mips(wb)` can be extracted. This xml string can later be assigned to an `options("openxlsx2.mips_xml_string" = fmips)` option.

The sensitivity label can then be assigned with `wb_add_mips(wb)`. If no xml string is passed, the MIP section is taken from the option. This should make it easier for users to read the section from a specific workbook, save it to a file or string and copy it to an option via the .Rprofile.

Value

the workbook invisible (wb_add_mips()) or the xml string (wb_get_mips())

wb_add_mschart *Add mschart object to a worksheet*

Description

Add mschart object to a worksheet

Usage

```
wb_add_mschart(
  wb,
  sheet = current_sheet(),
  dims = NULL,
  graph,
  col_offset = 0,
  row_offset = 0,
  ...
)
```

Arguments

wb	a workbook
sheet	the sheet on which the graph will appear
dims	the dimensions where the sheet will appear
graph	mschart object
col_offset, row_offset	offsets for column and row
...	additional arguments

See Also

wb_data() wb_add_chart_xml() wb_add_image wb_add_mschart() wb_add_plot

Examples

```
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_named_style` *Apply styling to a cell region with a named style*

Description

Set the styling to a named style for a cell region. Use `wb_add_cell_style()` to style a cell region with custom parameters. A named style is the one in spreadsheet software, like "Normal", "Warning".

Usage

```
wb_add_named_style(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  name = "Normal",
  font_name = NULL,
  font_size = NULL
)
```

Arguments

`wb`	A `wbWorkbook` object
`sheet`	A worksheet
`dims`	A cell range
`name`	The named style name.
`font_name`, `font_size`	optional else the default of the theme

Value

The wbWorkbook, invisibly

See Also

Other styles: wb_add_border(), wb_add_cell_style(), wb_add_fill(), wb_add_font(), wb_add_numfmt(), wb_cell_style

---

wb_add_numfmt *Modify number formatting in a cell region*

Description

Add number formatting to a cell region. You can use a number format created by create_numfmt().

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

Details

The list of number formats ID is located in the **Details** section of create_cell_style().

Value

The wbWorkbook object, invisibly.

See Also

Other styles: wb_add_border(), wb_add_cell_style(), wb_add_fill(), wb_add_font(), wb_add_named_style(), wb_cell_style

Examples

```
wb <- wb_workbook() %>% wb_add_worksheet("S1") %>% wb_add_data("S1", mtcars)
wb %>% wb_add_numfmt("S1", dims = "F1:F33", numfmt = "#.0")
# Chaining
wb <- wb_workbook()$add_worksheet("S1")$add_data("S1", mtcars)
wb$add_numfmt("S1", "A1:A33", numfmt = 1)
```

wb_add_page_break *Add a page break to a worksheet*

Description

Insert page breaks into a worksheet

Usage

```
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 of column number. One must be NULL

See Also

wb_add_worksheet()

Examples

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

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

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

wb_add_pivot_table *Add a pivot table to a worksheet*

Description

The data must be specified using wb_data() to ensure the function works. The sheet will be empty unless it is opened in spreadsheet software. Find more details in the section about pivot tables in the openxlsx2 book.

Usage

```
wb_add_pivot_table(
  wb,
  x,
  sheet = next_sheet(),
  dims = "A3",
  filter,
  rows,
  cols,
  data,
  fun,
  params,
  pivot_table,
  slicer,
  timeline
)
```

Arguments

wb	A Workbook object containing a #' worksheet.
x	A data.frame that inherits the wb_data class.
sheet	A worksheet containing a #'
dims	The worksheet cell where the pivot table is placed
filter	The column name(s) of x used for filter.
rows	The column name(s) of x used as rows
cols	The column names(s) of x used as cols
data	The column name(s) of x used as data
fun	A vector of functions to be used with data. See **Details** for the list of available options.
params	A list of parameters to modify pivot table creation. See **Details** for available options.
pivot_table	An optional name for the pivot table
slicer, timeline	Any additional column name(s) of x used as slicer/timeline

Details

The pivot table is not actually written to the worksheet, therefore the cell region has to remain empty. What is written to the workbook is something like a recipe how the spreadsheet software has to construct the pivot table when opening the file.

It is possible to add slicers to the pivot table. For this the pivot table has to be named and the variable used as slicer, must be part of the selected pivot table names (cols, rows, filter, or slicer). If these criteria are matched, a slicer can be added using wb_add_slicer().

Be aware that you should always test on a copy if a `param` argument works with a pivot table. Not only to check if the desired effect appears, but first and foremost if the file loads. Wildly mixing params might brick the output file and cause spreadsheet software to crash.

`fun` can be any of `AVERAGE`, `COUNT`, `COUNTA`, `MAX`, `MIN`, `PRODUCT`, `STDEV`, `STDEVP`, `SUM`, `VAR`, `VARP`.

`show_data_as` can be any of `normal`, `difference`, `percent`, `percentDiff`, `runTotal`, `percentOfRow`, `percentOfCol`, `percentOfTotal`, `index`.

It is possible to calculate data fields if the formula is assigned as a variable name for the field to calculate. This would look like this: `data = c("am", "disp/cyl" = "New")`

Possible `params` arguments are listed below. Pivot tables accepts more parameters, but they were either not tested or misbehaved (probably because we misunderstood how the parameter should be used).

Boolean arguments:

- apply_alignment_formats
- apply_number_formats
- apply_border_formats
- apply_font_formats
- apply_pattern_formats
- apply_width_height_formats
- no_style
- compact
- outline
- compact_data
- row_grand_totals
- col_grand_totals

Table styles accepting character strings:

- auto_format_id: style id as character in the range of 4096 to 4117
- table_style: a predefined (pivot) table style `"TableStyleMedium23"`
- show_data_as: accepts character strings as listed above

Miscellaneous:

- numfmt: accepts vectors of the form `c(formatCode = "0.0%")`
- choose: select variables in the form of a named logical vector like `c(agegp = 'x > "25-34"')` for the `esoph` dataset.
- sort_item: named list of index or character vectors

See Also

wb_data()

Other workbook wrappers: base_font-wb, col_widths-wb, creators-wb, grouping-wb, row_heights-wb, wb_add_chartsheet(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_slicer(), wb_add_worksheet(), wb_base_colors, wb_clone_worksheet(), wb_copy_cells(), wb_freeze_pane(), wb_merge_cells(), wb_save(), wb_set_last_modified_by(), wb_workbook()

Other worksheet content functions: col_widths-wb, filter-wb, grouping-wb, named_region-wb, row_heights-wb, wb_add_conditional_formatting(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_slicer(), wb_add_thread(), wb_freeze_pane(), wb_merge_cells()

Examples

```
wb <- wb_workbook() %>% wb_add_worksheet() %>% wb_add_data(x = mtcars)

df <- wb_data(wb, sheet = 1)

wb <- wb %>%
  # default pivot table
  wb_add_pivot_table(df, dims = "A3",
    filter = "am", rows = "cyl", cols = "gear", data = "disp"
  ) %>%
  # with parameters
  wb_add_pivot_table(df,
    filter = "am", rows = "cyl", cols = "gear", data = "disp",
    params = list(no_style = TRUE, numfmt = c(formatCode = "##0.0"))
  )
```

---

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(),
  dims = "A1",
  width = 6,
  height = 4,
  row_offset = 0,
  col_offset = 0,
  file_type = "png",
  units = "in",
```

```
  dpi = 300,
  ...
)
```

Arguments

wb	A workbook object
sheet	A name or index of a worksheet
dims	Worksheet dimension, single cell ("A1") or cell range ("A1:D4")
width	Width of figure. Defaults to 6 in.
height	Height of figure . Defaults to 4 in.
row_offset, col_offset	Offset for column and row
file_type	File type of image
units	Units of width and height. Can be "in", "cm" or "px"
dpi	Image resolution
...	additional arguments

See Also

wb_add_chart_xml() wb_add_drawing() wb_add_image() wb_add_mschart()

Examples

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

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

## create plot objects
require(ggplot2)
p1 <- ggplot(mtcars, aes(x = mpg, fill = as.factor(gear))) +
  ggtitle("Distribution of Gas Mileage") +
  geom_density(alpha = 0.5)
p2 <- ggplot(Orange, aes(x = age, y = circumference, color = 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, file_type = "png", units = "in")

## Insert plot 2
print(p2)
wb$add_plot(1, dims = "J2", width = 16, height = 10, file_type = "png", units = "cm")

}
```

wb_add_slicer *Add a slicer/timeline to a pivot table*

Description

Add a slicer/timeline to a previously created pivot table. This function is still experimental and might be changed/improved in upcoming releases.

Usage

```
wb_add_slicer(
  wb,
  x,
  dims = "A1",
  sheet = current_sheet(),
  pivot_table,
  slicer,
  params
)

wb_remove_slicer(wb, sheet = current_sheet())

wb_add_timeline(
  wb,
  x,
  dims = "A1",
  sheet = current_sheet(),
  pivot_table,
  timeline,
  params
)

wb_remove_timeline(wb, sheet = current_sheet())
```

Arguments

wb	A Workbook object containing a worksheet.
x	A data.frame that inherits the wb_data class.
dims	The worksheet cell where the pivot table is placed
sheet	A worksheet
pivot_table	The name of a pivot table
slicer, timeline	A variable used as slicer/timeline for the pivot table
params	A list of parameters to modify pivot table creation. See **Details** for available options.

Details

This assumes that the slicer/timeline variable initialization has happened before. Unfortunately, it is unlikely that we can guarantee this for loaded workbooks, and we *strictly* discourage users from attempting this. If the variable has not been initialized properly, this may cause the spreadsheet software to crash. Although it is documented that slicers should use "TimelineStyleLight[1-6]" and "TimelineStyleDark[1-6]" they use slicer styles.

Possible `params` arguments for slicers are listed below.

- edit_as: "twoCell" to place the slicer into the cells
- column_count: integer used as column count
- sort_order: "descending" / "ascending"
- choose: select variables in the form of a named logical vector like `c(agegp = 'x > "25-34"')` for the `esoph` dataset.
- locked_position
- start_item

Possible `params` arguments for timelines are listed below.

- beg_date/end_date: dates when the timeline should begin or end
- choose_beg/choose_end: dates when the selection should begin or end
- scroll_position
- show_selection_label
- show_time_level
- show_horizontal_scrollbar

Possible common `params`:

- caption: string used for a caption
- style: "SlicerStyleLight[1-6]", "SlicerStyleDark[1-6]" only for slicer "SlicerStyleOther[1-2]"
- level: the granularity of the slicer (for timeline 0 = year, 1 = quarter, 2 = month)
- show_caption: logical if caption should be shown or not

Removing works on the spreadsheet level. Therefore all slicers/timelines are removed from a worksheet. At the moment the drawing reference remains on the spreadsheet. Therefore spreadsheet software that does not handle slicers/timelines will still show the drawing.

See Also

Other workbook wrappers: `base_font-wb`, `col_widths-wb`, `creators-wb`, `grouping-wb`, `row_heights-wb`, `wb_add_chartsheet()`, `wb_add_data()`, `wb_add_data_table()`, `wb_add_formula()`, `wb_add_pivot_table()`, `wb_add_worksheet()`, `wb_base_colors`, `wb_clone_worksheet()`, `wb_copy_cells()`, `wb_freeze_pane()`, `wb_merge_cells()`, `wb_save()`, `wb_set_last_modified_by()`, `wb_workbook()`

Other worksheet content functions: `col_widths-wb`, `filter-wb`, `grouping-wb`, `named_region-wb`, `row_heights-wb`, `wb_add_conditional_formatting()`, `wb_add_data()`, `wb_add_data_table()`, `wb_add_formula()`, `wb_add_pivot_table()`, `wb_add_thread()`, `wb_freeze_pane()`, `wb_merge_cells()`

Examples

```
# prepare data
df <- data.frame(
  AirPassengers = c(AirPassengers),
  time = seq(from = as.Date("1949-01-01"), to = as.Date("1960-12-01"), by = "month"),
  letters = letters[1:4]
)

# create workbook
wb <- wb_workbook()$
  add_worksheet("pivot")$
  add_worksheet("data")$
  add_data(x = df)

# get pivot table data source
df <- wb_data(wb, sheet = "data")

# create pivot table
wb$add_pivot_table(
  df,
  sheet = "pivot",
  rows = "time",
  cols = "letters",
  data = "AirPassengers",
  pivot_table = "airpassengers",
  params = list(
    compact = FALSE, outline = FALSE, compact_data = FALSE,
    row_grand_totals = FALSE, col_grand_totals = FALSE)
)

# add slicer
wb$add_slicer(
  df,
  dims = "E1:I7",
  sheet = "pivot",
  slicer = "letters",
  pivot_table = "airpassengers",
  params = list(choose = c(letters = 'x %in% c("a", "b")'))
)

# add timeline
wb$add_timeline(
  df,
  dims = "E9:I14",
  sheet = "pivot",
  timeline = "time",
  pivot_table = "airpassengers",
  params = list(
    beg_date = as.Date("1954-01-01"),
    end_date = as.Date("1961-01-01"),
    choose_beg = as.Date("1957-01-01"),
    choose_end = as.Date("1958-01-01"),
```

```
    level = 0,
    style = "TimeSlicerStyleLight2"
  )
)
```

---

wb_add_sparklines *Add sparklines to a worksheet*

Description

Add sparklines to a worksheet

Usage

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

Arguments

wb	A wbWorkbook
sheet	sheet to add the sparklines to
sparklines	sparklines object created with create_sparklines()

See Also

create_sparklines()

Examples

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

---

wb_add_style *Set the default style in a workbook*

Description

wb wrapper to add style to workbook

Usage

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

Arguments

wb	A workbook
style	style xml character, created by a create_*() function.
style_name	style name used optional argument

Value

The wbWorkbook object, invisibly.

See Also

- create_border()
- create_cell_style()
- create_dxfs_style()
- create_fill()
- create_font()
- create_numfmt()

Other workbook styling functions: base_font-wb, wb_add_dxfs_style(), wb_base_colors

Examples

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

yellow <- create_dxfs_style(font_color = yellow_f, bg_fill = yellow_b)
wb <- wb_workbook() %>% wb_add_style(yellow)
```

---

wb_add_thread *Add threaded comments to a cell in a worksheet*

Description

These functions allow adding thread comments to spreadsheets. This is not yet supported by all spreadsheet software. A threaded comment must be tied to a person created by wb_add_person().

Usage

```
wb_add_thread(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  comment = NULL,
  person_id,
  reply = FALSE,
  resolve = FALSE
)

wb_get_thread(wb, sheet = current_sheet(), dims = NULL)
```

Arguments

wb	A workbook
sheet	A worksheet
dims	A cell
comment	The text to add, a character vector.
person_id	the person Id this should be added. The default is `getOption("openxlsx2.thread_id")` if set.
reply	Is the comment a reply? (default FALSE)
resolve	Should the comment be resolved? (default FALSE)

Details

If a threaded comment is added, it needs a person attached to it. The default is to create a person with provider id "None". Other providers are possible with specific values for id and user_id. If you require the following, create a workbook via spreadsheet software load it and get the values with wb_get_person()

See Also

wb_add_comment() person-wb

Other worksheet content functions: col_widths-wb, filter-wb, grouping-wb, named_region-wb, row_heights-wb, wb_add_conditional_formatting(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_freeze_pane(), wb_merge_cells()

Examples

```
wb <- wb_workbook()$add_worksheet()
# Add a person to the workbook.
wb$add_person(name = "someone who likes to edit workbooks")

pid <- wb$get_person(name = "someone who likes to edit workbooks")$id

# write a comment to a thread, reply to one and solve some
wb <- wb %>%
  wb_add_thread(dims = "A1", comment = "wow it works!", person_id = pid) %>%
  wb_add_thread(dims = "A2", comment = "indeed", person_id = pid, resolve = TRUE) %>%
  wb_add_thread(dims = "A1", comment = "so cool", person_id = pid, reply = TRUE)
```

---

wb_add_worksheet *Add a worksheet to a workbook*

Description

Add a worksheet to a wbWorkbook is the first step to build a workbook. With the function, you can also set the sheet view with zoom, set headers and footers as well as other features. See the function arguments.

Usage

```
wb_add_worksheet(
  wb,
  sheet = next_sheet(),
  grid_lines = TRUE,
  row_col_headers = TRUE,
  tab_color = NULL,
  zoom = 100,
  header = NULL,
  footer = NULL,
  odd_header = header,
  odd_footer = footer,
  even_header = header,
  even_footer = footer,
  first_header = header,
  first_footer = footer,
  visible = c("true", "false", "hidden", "visible", "veryhidden"),
  has_drawing = FALSE,
  paper_size = 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 wbWorkbook object to attach the new worksheet
sheet	A name for the new worksheet
grid_lines	A logical. If FALSE, the worksheet grid lines will be hidden.
row_col_headers	A logical. If FALSE, the worksheet colname and rowname will be hidden.
tab_color	Color of the sheet tab. A wb_color(), a valid color (belonging to grDevices::colors()) or a valid hex color beginning with "#".
zoom	The sheet zoom level, a numeric between 10 and 400 as a percentage. (A zoom value smaller than 10 will default to 10.)
header, odd_header, even_header, first_header, footer, odd_footer, even_footer, first_footer	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.
has_drawing	If TRUE prepare a drawing output (TODO does this work?)
paper_size	An integer corresponding to a paper size. See wb_page_setup() for details.
orientation	One of "portrait" or "landscape"

hdpi, vdpi	Horizontal and vertical DPI. Can be set with `options("openxlsx2.dpi" = X)`, `options("openxlsx2.hdpi" = X)` or `options("openxlsx2.vdpi" = X)`
...	Additional arguments

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 `wbWorkbook` object, invisibly.

See Also

Other workbook wrappers: `base_font-wb`, `col_widths-wb`, `creators-wb`, `grouping-wb`, `row_heights-wb`, `wb_add_chartsheet()`, `wb_add_data()`, `wb_add_data_table()`, `wb_add_formula()`, `wb_add_pivot_table()`, `wb_add_slicer()`, `wb_base_colors`, `wb_clone_worksheet()`, `wb_copy_cells()`, `wb_freeze_pane()`, `wb_merge_cells()`, `wb_save()`, `wb_set_last_modified_by()`, `wb_workbook()`

Examples

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

## Add a worksheet
wb$add_worksheet("Sheet 1")
## No grid lines
wb$add_worksheet("Sheet 2", grid_lines = FALSE)
## A red tab color
wb$add_worksheet("Sheet 3", tab_color = wb_color("red"))
## All options combined with a zoom of 40%
wb$add_worksheet("Sheet 4", grid_lines = FALSE, tab_color = wb_color(hex = "#4F81BD"), zoom = 40)

## 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"),
  even_header = c("EVEN HEAD LEFT", "EVEN HEAD CENTER", "EVEN HEAD RIGHT"),
  even_footer = c("EVEN FOOT RIGHT", "EVEN FOOT CENTER", "EVEN FOOT RIGHT"),
  first_header = c("TOP", "OF FIRST", "PAGE"),
  first_footer = 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]"),
  first_header = c(NA, "Center Header of First Page", NA),
  first_footer = 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",
  first_header = c("FIRST ONLY L", NA, "FIRST ONLY R"),
  first_footer = 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)
```

`wb_base_colors` *Set the default colors in a workbook*

Description

Modify / get the default colors of the workbook.

Usage

```
wb_set_base_colors(wb, theme = "Office", ...)

wb_get_base_colors(wb, xml = FALSE, plot = TRUE)
```

Arguments

`wb`	A workbook object
`theme`	a predefined color theme
`...`	optional parameters
`xml`	Logical if xml string should be returned
`plot`	Logical if a barplot of the colors should be returned

Details

Theme must be any of the following: "Aspect", "Blue", "Blue II", "Blue Green", "Blue Warm", "Greyscale", "Green", "Green Yellow", "Marquee", "Median", "Office", "Office 2007 - 2010", "Office 2013 - 2022", "Orange", "Orange Red", "Paper", "Red", "Red Orange", "Red Violet", "Slipstream", "Violet", "Violet II", "Yellow", "Yellow Orange"

See Also

Other workbook styling functions: base_font-wb, wb_add_dxfs_style(), wb_add_style()

Other workbook wrappers: base_font-wb, col_widths-wb, creators-wb, grouping-wb, row_heights-wb, wb_add_chartsheet(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_worksheet(), wb_clone_worksheet(), wb_copy_cells(), wb_freeze_pane(), wb_merge_cells(), wb_save(), wb_set_last_modified_by(), wb_workbook()

Examples

```
wb <- wb_workbook()
wb$get_base_colors()
wb$set_base_colors(theme = 3)
wb$set_base_colors(theme = "Violet II")
wb$get_base_colours()
```

---

wb_cell_style *Apply styling to a cell region*

Description

Setting a style across only impacts cells that are not yet part of a workbook. The effect is similar to setting the cell style for all cells in a row independently, though much quicker and less memory consuming.

Usage

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

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

wb_set_cell_style_across(
  wb,
  sheet = current_sheet(),
  style,
  cols = NULL,
  rows = NULL
)
```

Arguments

wb	A wbWorkbook object
sheet	sheet
dims	A cell range in the worksheet
style	A style or a cell with a certain style
cols	The columns the style will be applied to, either "A:D" or 1:4
rows	The rows the style will be applied to

Value

A Workbook object

See Also

Other styles: wb_add_border(), wb_add_cell_style(), wb_add_fill(), wb_add_font(), wb_add_named_style(), wb_add_numfmt()

Examples

```
# 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)

# set style across a workbook
wb <- wb_workbook() %>%
  wb_add_worksheet() %>%
  wb_add_fill(dims = "C3", color = wb_color("yellow")) %>%
  wb_set_cell_style_across(style = "C3", cols = "C:D", rows = 3:4)
```

---

wb_clean_sheet *Remove all values in a worksheet*

Description

Remove content of a worksheet completely, or a region if specifying dims.

Usage

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

Arguments

wb	A Workbook object
sheet	sheet to clean
dims	spreadsheet dimensions (optional)
numbers	remove all numbers
characters	remove all characters
styles	remove all styles
merged_cells	remove all merged_cells

Value

A Workbook object

---

wb_clone_sheet_style *Apply styling from a sheet to another within a workbook*

Description

This function can be used to apply styling from a cell range, and apply it to another cell range.

Usage

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

Arguments

wb	A workbook
from	sheet we select the style from
to	sheet to apply the style to

wb_clone_worksheet *Create copies of a worksheet within a workbook*

Description

Create a copy of a worksheet in the same wbWorkbook object.

Cloning is possible only to a limited extent. References to sheet names in formulas, charts, pivot tables, etc. may not be updated. Some elements like named ranges and slicers cannot be cloned yet.

Cloning from another workbook is still an experimental feature and might not work reliably. Cloning data, media, charts and tables should work. Slicers and pivot tables as well as everything everything relying on dxfs styles (e.g. custom table styles and conditional formatting) is currently not implemented. Formula references are not updated to reflect interactions between workbooks.

Usage

```
wb_clone_worksheet(wb, old = current_sheet(), new = next_sheet(), from = NULL)
```

Arguments

wb	A wbWorkbook object
old	Name of existing worksheet to copy
new	Name of the new worksheet to create
from	(optional) Workbook to clone old from

Value

The wbWorkbook object, invisibly.

See Also

Other workbook wrappers: base_font-wb, col_widths-wb, creators-wb, grouping-wb, row_heights-wb, wb_add_chartsheet(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_worksheet(), wb_base_colors, wb_copy_cells(), wb_freeze_pane(), wb_merge_cells(), wb_save(), wb_set_last_modified_by(), wb_workbook()

Examples

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

# Add worksheets
wb$add_worksheet("Sheet 1")
wb$clone_worksheet("Sheet 1", new = "Sheet 2")
# Take advantage of waiver functions
wb$clone_worksheet(old = "Sheet 1")

## cloning from another workbook
```

```
# create a workbook
wb <- wb_workbook()$
add_worksheet("NOT_SUM")$
  add_data(x = head(iris))$
  add_fill(dims = "A1:B2", color = wb_color("yellow"))$
  add_border(dims = "B2:C3")

# we will clone this styled chart into another workbook
fl <- system.file("extdata", "oxlsx2_sheet.xlsx", package = "openxlsx2")
wb_from <- wb_load(fl)

# clone styles and shared strings
wb$clone_worksheet(old = "SUM", new = "SUM", from = wb_from)
```

`wb_color` *Helper to create a color*

Description

Creates a `wbColour` object.

Usage

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

Arguments

`name`	A name of a color known to R either as name or RGB/ARGB value.
`auto`	A boolean.
`indexed`	An indexed color value. This color has to be provided by the workbook.
`hex`	A rgb color either a ARGB hex value or RGB hex value With or without leading "#".
`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

See Also

wb_get_base_colors() grDevices::colors()

---

wb_comment *Helper to create a comment object*

Description

Creates a wbComment object. Use with wb_add_comment() to add to a worksheet location.

Usage

```
wb_comment(
  text = NULL,
  style = NULL,
  visible = FALSE,
  author = getOption("openxlsx2.creator"),
  width = 2,
  height = 4
)
```

Arguments

text	Comment text. Character vector. or a fmt_txt() string.
style	A Style object or list of style objects the same length as comment vector.
visible	Is comment visible? Default: FALSE.
author	Author of comment. A string. By default, will look at options("openxlsx2.creator"). Otherwise, will check the system username.
width	Textbox integer width in number of cells
height	Textbox integer height in number of cells

Value

A wbComment object

Examples

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

# write comment without author
c1 <- wb_comment(text = "this is a comment", author = "", visible = TRUE)
wb$add_comment(dims = "B10", comment = c1)

# Write another comment with author information
c2 <- wb_comment(text = "this is another comment", author = "Marco Polo")
```

```
wb$add_comment(sheet = 1, dims = "C10", comment = c2)

# write a styled comment with system author
s1 <- create_font(b = "true", color = wb_color(hex = "FFFF0000"), sz = "12")
s2 <- create_font(color = wb_color(hex = "FF000000"), sz = "9")
c3 <- wb_comment(text = c("This Part Bold red\n\n", "This part black"), style = c(s1, s2))

wb$add_comment(sheet = 1, dims = wb_dims(3, 6), comment = c3)
```

`wb_copy_cells` *Copy cells around within a worksheet*

Description

Copy cells around within a worksheet

Usage

```
wb_copy_cells(
  wb,
  sheet = current_sheet(),
  dims = "A1",
  data,
  as_value = FALSE,
  as_ref = FALSE,
  transpose = FALSE,
  ...
)
```

Arguments

`wb`	A workbook
`sheet`	a worksheet
`dims`	A cell where to place the copy
`data`	A `wb_data` object containing cells to copy
`as_value`	Should a copy of the value be written?
`as_ref`	Should references to the cell be written?
`transpose`	Should the data be written transposed?
`...`	additional arguments passed to add_data() if used with `as_value`

Value

the `wbWorkbook` invisibly

See Also

wb_data()

Other workbook wrappers: base_font-wb, col_widths-wb, creators-wb, grouping-wb, row_heights-wb, wb_add_chartsheet(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_worksheet(), wb_base_colors, wb_clone_worksheet(), wb_freeze_pane(), wb_merge_cells(), wb_save(), wb_set_last_modified_by(), wb_workbook()

Examples

```
wb <- wb_workbook()$
add_worksheet()$
  add_data(x = mtcars)$
  add_fill(dims = "A1:F1", color = wb_color("yellow"))

dat <- wb_data(wb, dims = "A1:D4", col_names = FALSE)
# 1:1 copy to M2
wb$
  clone_worksheet(old = 1, new = "Clone1")$
  copy_cells(data = dat, dims = "M2")
```

---

wb_data *Add the* wb_data *attribute to a data frame in a worksheet*

Description

provide wb_data object as mschart input

Usage

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

## S3 method for class 'wb_data'

  x[
  i,
  j,
 drop = ifelse((missing(j) && length(i) > 1) || (!missing(i) && length(j) > 1), FALSE,
    TRUE)
]
```

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.

x	x
i	i
j	j
drop	drop

Value

A data frame of class wb_data.

See Also

wb_to_df() wb_add_mschart(), wb_add_pivot_table()

Examples

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

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

---

wb_dims *Helper to specify the* dims *argument*

Description

wb_dims() can be used to help provide the dims argument, in the wb_add_* functions. It returns a A1 spreadsheet range ("A1:B1" or "A2"). It can be very useful as you can specify many parameters that interact together In general, you must provide named arguments. wb_dims() will only accept unnamed arguments if they are rows, cols, for example wb_dims(1:4, 1:2), that will return "A1:B4".

wb_dims() can also be used with an object (a data.frame or a matrix for example.) All parameters are numeric unless stated otherwise.

Usage

```
wb_dims(..., select = NULL)
```

Arguments

...	construct dims arguments, from rows/cols vectors or objects that can be coerced to data frame. x, rows, cols, from_row, from_col, from_dims row_names, and col_names are accepted.
select	A string, one of the followings. it improves the selection of various parts of x One of "x", "data", "col_names", or "row_names". "data" will only select the data part, excluding row names and column names (default if cols or rows are specified) "x" Includes the data, column and row names if they are present. (default if none of rows and cols are provided) "col_names" will only return column names "row_names" Will only return row names.

Details

When using wb_dims() with an object, the default behavior is to select only the data / row or columns in x If you need another behavior, use wb_dims() without supplying x.

- x An object (typically a matrix or a data.frame, but a vector is also accepted.)
- from_row / from_col / from_dims the starting position of x (The dims returned will assume that the top left corner of x is at from_row / from_col
- rows Optional Which row span in x should this apply to. If rows = 0, only column names will be affected.
- cols a range of columns id in x, or one of the column names of x (length 1 only accepted for column names of x.)
- row_names A logical, this is to let wb_dims() know that x has row names or not. If row_names = TRUE, wb_dims() will increment from_col by 1.
- col_names wb_dims() assumes that if x has column names, then trying to find the dims.

wb_dims() tries to support most possible cases with row_names = TRUE and col_names = FALSE, but it works best if x has named dimensions (data.frame, matrix), and those parameters are not specified. data with column names, and without row names. as the code is more clean.

In the add_data() / add_font() example, if writing the data with row names

Value

A dims string

Using wb_dims() without an x object

- rows / cols (if you want to specify a single one, use from_row / from_col)
- from_row / from_col the starting position of the dims (similar to start_row / start_col, but with a clearer name.)

Using wb_dims() with an x object

wb_dims() with an object has 8 use-cases (they work with any position values of from_row / from_col), from_col/from_row correspond to the coordinates at the top left of x including column and row names if present.

These use cases are provided without from_row / from_col, but they work also with from_row / from_col.

1. provide the full grid with wb_dims(x = mtcars)
2. provide the data grid wb_dims(x = mtcars, select = "data")
3. provide the dims of column names wb_dims(x = mtcars, select = "col_names)
4. provide the dims of row names wb_dims(x = mtcars, row_names = TRUE, select = "row_names")
5. provide the dims of a row span wb_dims(x = mtcars, rows = 1:10) selects the first 10 data rows of mtcars (ignoring column names)
6. provide the dims of the data in a column span wb_dims(x = mtcars, cols = 1:5) select the data first 5 columns of mtcars

7. provide a column span (including column names) `wb_dims(x = mtcars, cols = 4:7, select = "x")` select the data columns 4, 5, 6, 7 of `mtcars` + column names
8. provide the position of a single column by name `wb_dims(x = mtcars, cols = "mpg")`.
9. provide a row span with a column. `wb_dims(x = mtcars, cols = "mpg", rows = 5:22)`

To reuse, a good trick is to create a wrapper function, so that styling can be performed seamlessly.

```
wb_dims_cars <- function(...) {
  wb_dims(x = mtcars, from_row = 2, from_col = "B", ...)
}
# using this function
wb_dims_cars()                      # full grid (data + column names)
wb_dims_cars(select = "data")       # data only
wb_dims_cars(select = "col_names") # select column names
wb_dims_cars(cols = "vs")           # select the `vs` column
```

It can be very useful to apply many rounds of styling sequentially.

Examples

```
# Provide coordinates
wb_dims(1, 4)
wb_dims(rows = 1, cols = 4)
wb_dims(from_row = 4)
wb_dims(from_col = 2)
wb_dims(from_col = "B")
wb_dims(1:4, 6:9, from_row = 5)
# Provide vectors
wb_dims(1:10, c("A", "B", "C"))
wb_dims(rows = 1:10, cols = 1:10)
# provide `from_col` / `from_row`
wb_dims(rows = 1:10, cols = c("A", "B", "C"), from_row = 2)
wb_dims(rows = 1:10, cols = 1:10, from_col = 2)
wb_dims(rows = 1:10, cols = 1:10, from_dims = "B1")
# or objects
wb_dims(x = mtcars, col_names = TRUE)

# select all data
wb_dims(x = mtcars, select = "data")

# column names of an object (with the special select = "col_names")
wb_dims(x = mtcars, select = "col_names")


# dims of the column names of an object
wb_dims(x = mtcars, select = "col_names", col_names = TRUE)

## add formatting to `mtcars` using `wb_dims()`----
wb <- wb_workbook()
wb$add_worksheet("test wb_dims() with an object")
dims_mtcars_and_col_names <- wb_dims(x = mtcars)
```

```
wb$add_data(x = mtcars, dims = dims_mtcars_and_col_names)

# Put the font as Arial for the data
dims_mtcars_data <- wb_dims(x = mtcars, select = "data")
wb$add_font(dims = dims_mtcars_data, name = "Arial")

# Style col names as bold using the special `select = "col_names"` with `x` provided.
dims_column_names <- wb_dims(x = mtcars, select = "col_names")
wb$add_font(dims = dims_column_names, bold = TRUE, size = 13)

# Finally, to add styling to column "cyl" (the 4th column) (only the data)
# there are many options, but here is the preferred one
# if you know the column index, wb_dims(x = mtcars, cols = 4) also works.
dims_cyl <- wb_dims(x = mtcars, cols = "cyl")
wb$add_fill(dims = dims_cyl, color = wb_color("pink"))

# Mark a full column as important(with the column name too)
wb_dims_vs <- wb_dims(x = mtcars, cols = "vs", select = "x")
wb$add_fill(dims = wb_dims_vs, fill = wb_color("yellow"))
wb$add_conditional_formatting(dims = wb_dims(x = mtcars, cols = "mpg"), type = "dataBar")
# wb_open(wb)
```

---

wb_freeze_pane *Freeze pane of a worksheet*

Description

Add a Freeze pane in a worksheet.

Usage

```
wb_freeze_pane(
  wb,
  sheet = current_sheet(),
  first_active_row = NULL,
  first_active_col = NULL,
  first_row = FALSE,
  first_col = FALSE,
  ...
)
```

Arguments

wb	A workbook object
sheet	A name or index of a worksheet
first_active_row	Top row of active region
first_active_col	Furthest left column of active region

first_row	If TRUE, freezes the first row (equivalent to first_active_row = 2)
first_col	If TRUE, freezes the first column (equivalent to first_active_col = 2)
...	additional arguments

See Also

Other workbook wrappers: base_font-wb, col_widths-wb, creators-wb, grouping-wb, row_heights-wb, wb_add_chartsheet(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_worksheet(), wb_base_colors, wb_clone_worksheet(), wb_copy_cells(), wb_merge_cells(), wb_save(), wb_set_last_modified_by(), wb_workbook()

Other worksheet content functions: col_widths-wb, filter-wb, grouping-wb, named_region-wb, row_heights-wb, wb_add_conditional_formatting(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_thread(), wb_merge_cells()

Examples

```
## 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$freeze_pane("Sheet 1", first_active_row = 5, first_active_col = 3)
wb$freeze_pane("Sheet 2", first_col = TRUE) ## shortcut to first_active_col = 2
wb$freeze_pane(3, first_row = TRUE) ## shortcut to first_active_row = 2
wb$freeze_pane(4, first_active_row = 1, first_active_col = "D")
```

---

wb_get_tables *List Excel tables in a worksheet*

Description

List Excel tables in a worksheet

Usage

```
wb_get_tables(wb, sheet = current_sheet())
```

Arguments

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

Value

A character vector of table names on the specified sheet

Examples

```
wb <- wb_workbook()
wb$add_worksheet(sheet = "Sheet 1")
wb$add_data_table(x = iris)
wb$add_data_table(x = mtcars, table_name = "mtcars", start_col = 10)

wb$get_tables(sheet = "Sheet 1")
```

---

wb_load *Load an existing .xlsx, .xlsm or .xlsb file*

Description

`wb_load()` returns a wbWorkbook object conserving the content of the original input file, including data, styles, media. This workbook can be modified, read from, and be written back into a xlsx file.

Usage

```
wb_load(file, sheet, data_only = FALSE, ...)
```

Arguments

`file`	A path to an existing .xlsx, .xlsm or .xlsb file
`sheet`	optional sheet parameter. if this is applied, only the selected sheet will be loaded. This can be a numeric, a string or `NULL`.
`data_only`	mode to import if only a data frame should be returned. This strips the `wbWorkbook` to a bare minimum.
`...`	additional arguments

Details

If a specific `sheet` is selected, the workbook will still contain sheets for all worksheets. The argument `sheet` and `data_only` are used internally by `wb_to_df()` to read from a file with minimal changes. They are not specifically designed to create rudimentary but otherwise fully functional workbooks. It is possible to import with `wb_load(data_only = TRUE, sheet = NULL)`. In this way, only a workbook framework is loaded without worksheets or data. This can be useful if only some workbook properties are of interest.

There are some internal arguments that can be passed to wb_load, which are used for development. The debug argument allows debugging of `xlsb` files in particular. With `calc_chain` it is possible to maintain the calculation chain. The calculation chain is used by spreadsheet software to determine the order in which formulas are evaluated. Removing the calculation chain has no known effect. The calculation chain is created the next time the worksheet is loaded into the spreadsheet. Keeping

the calculation chain could only shorten the loading time in said software. Unfortunately, if a cell is added to the worksheet, the calculation chain may block the worksheet as the formulas will not be evaluated again until each individual cell with a formula is selected in the spreadsheet software and the Enter key is pressed manually. It is therefore strongly recommended not to activate this function.

In rare cases, a warning is issued when loading an xlsx file that an xml namespace has been removed from xml files. This refers to the internal structure of the loaded 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`. It is therefore removed when the file is loaded into a workbook. Removal is generally considered safe, but the feature is still not commonly observed, hence the warning.

Initial support for binary openxml files (`xlsb`) has been added to the package. We parse the binary file format into pseudo-openxml files that we can import. Therefore, once imported, it is possible to interact with the file as if it had been provided in xlsx file format in the first place. This parsing into pseudo xml files is of course slower than reading directly from the binary file. Our implementation is also still missing some functions: some array formulas are not yet correct, conditional formatting and data validation are not implemented, nor are pivot tables and slicers.

Value

A Workbook object.

Examples

```
## load existing workbook
fl <- system.file("extdata", "openxlsx2_example.xlsx", package = "openxlsx2")
wb <- wb_load(file = fl)
```

---

wb_merge_cells *Merge cells within a worksheet*

Description

Worksheet cell merging

Usage

```
wb_merge_cells(wb, sheet = current_sheet(), dims = NULL, solve = FALSE, ...)

wb_unmerge_cells(wb, sheet = current_sheet(), dims = NULL, ...)
```

Arguments

`wb`	A Workbook object
`sheet`	A name or index of a worksheet
`dims`	worksheet cells
`solve`	logical if intersecting merges should be solved
`...`	additional arguments

Details

If using the deprecated arguments `rows` and `cols` with a merged region must be rectangular, only min and max of `cols` and `rows` are used.

See Also

Other workbook wrappers: `base_font-wb`, `col_widths-wb`, `creators-wb`, `grouping-wb`, `row_heights-wb`, `wb_add_chartsheet()`, `wb_add_data()`, `wb_add_data_table()`, `wb_add_formula()`, `wb_add_pivot_table()`, `wb_add_slicer()`, `wb_add_worksheet()`, `wb_base_colors`, `wb_clone_worksheet()`, `wb_copy_cells()`, `wb_freeze_pane()`, `wb_save()`, `wb_set_last_modified_by()`, `wb_workbook()`

Other worksheet content functions: `col_widths-wb`, `filter-wb`, `grouping-wb`, `named_region-wb`, `row_heights-wb`, `wb_add_conditional_formatting()`, `wb_add_data()`, `wb_add_data_table()`, `wb_add_formula()`, `wb_add_pivot_table()`, `wb_add_slicer()`, `wb_add_thread()`, `wb_freeze_pane()`

Examples

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

# Merge cells: Row 2 column C to F (3:6)
wb <- wb_merge_cells(wb, dims = "C3:F6")

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

wb$add_worksheet()

## Intersecting merges
wb <- wb_merge_cells(wb, dims = wb_dims(cols = 1:10, rows = 1))
wb <- wb_merge_cells(wb, dims = wb_dims(cols = 5:10, rows = 2))
wb <- wb_merge_cells(wb, dims = wb_dims(cols = 1:10, rows = 12))
try(wb_merge_cells(wb, dims = "A1:A10"))

## remove merged cells
# removes any intersecting merges
wb <- wb_unmerge_cells(wb, dims = wb_dims(cols = 1, rows = 1))
wb <- wb_merge_cells(wb, dims = "A1:A10")

# or let us decide how to solve this
wb <- wb_merge_cells(wb, dims = "A1:A10", solve = TRUE)
```

---

wb_open *Preview a workbook in a spreadsheet software*

Description

You can also use the shorter `wb$open()` as a replacement. To open xlsx files, see `xl_open()`.

Usage

```
wb_open(wb)
```

Arguments

wb	a wbWorkbook object

---

wb_order *Order worksheets in a workbook*

Description

Get/set order of worksheets in a Workbook object

Usage

```
wb_get_order(wb)

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

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

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

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

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

wb_page_setup *Set page margins, orientation and print scaling of a worksheet*

Description

Set page margins, orientation and print scaling.

Usage

```
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,
  fit_to_width = FALSE,
  fit_to_height = FALSE,
  paper_size = NULL,
  print_title_rows = NULL,
  print_title_cols = NULL,
  summary_row = NULL,
  summary_col = 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, right, top, bottom	Page margin in inches
header, footer	Margin in inches
fit_to_width, fit_to_height	An integer that tells the spreadsheet software on how many pages the scaling should fit. This does not actually scale the sheet.
paper_size	See details. Default value is 9 (A4 paper).
print_title_rows, print_title_cols	Rows / columns to repeat at top of page when printing. Integer vector.

summary_row	Location of summary rows in groupings. One of "Above" or "Below".
summary_col	Location of summary columns in groupings. One of "Right" or "Left".
...	additional arguments

Details

When adding fitting to width and height manual adjustment of the scaling factor is required. Setting `fit_to_width` and `fit_to_height` only tells spreadsheet software that the scaling was applied, but not which scaling was applied.

`paper_size` is an integer corresponding to:

size	"paper type"
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)
69 Japanese Double Postcard (200 mm x 148 mm) 70=A6(105mm x 148mm)
71 Japanese Envelope Kaku #2
72 Japanese Envelope Kaku #3
73 Japanese Envelope Chou #3
74 Japanese Envelope Chou #4
75 Letter Rotated (11in x 8 1/2 11 in)
76 A3 Rotated (420 mm x 297 mm)
77 A4 Rotated (297 mm x 210 mm)
78 A5 Rotated (210 mm x 148 mm)
79 B4 (JIS) Rotated (364 mm x 257 mm)
80 B5 (JIS) Rotated (257 mm x 182 mm)
81 Japanese Postcard Rotated (148 mm x 100 mm)
82 Double Japanese Postcard Rotated (148 mm x 200 mm) 83 = A6 Rotated (148 mm x 105 mm)
84 Japanese Envelope Kaku #2 Rotated
85 Japanese Envelope Kaku #3 Rotated
86 Japanese Envelope Chou #3 Rotated
87 Japanese Envelope Chou #4 Rotated 88=B6(JIS)(128mm x 182mm)
89 B6 (JIS) Rotated (182 mm x 128 mm)

90 (12 in x 11 in)
91 Japanese Envelope You #4
92 Japanese Envelope You #4 Rotated 93=PRC16K(146mm x 215mm) 94=PRC32K(97mm x 151mm)
95 PRC 32K(Big) (97 mm x 151 mm)
96 PRC Envelope #1 (102 mm x 165 mm)
97 PRC Envelope #2 (102 mm x 176 mm)
98 PRC Envelope #3 (125 mm x 176 mm)
99 PRC Envelope #4 (110 mm x 208 mm)
100 PRC Envelope #5 (110 mm x 220 mm)
101 PRC Envelope #6 (120 mm x 230 mm)
102 PRC Envelope #7 (160 mm x 230 mm)
103 PRC Envelope #8 (120 mm x 309 mm)
104 PRC Envelope #9 (229 mm x 324 mm)
105 PRC Envelope #10 (324 mm x 458 mm)
106 PRC 16K Rotated
107 PRC 32K Rotated
108 PRC 32K(Big) Rotated
109 PRC Envelope #1 Rotated (165 mm x 102 mm)
110 PRC Envelope #2 Rotated (176 mm x 102 mm)
111 PRC Envelope #3 Rotated (176 mm x 125 mm)
112 PRC Envelope #4 Rotated (208 mm x 110 mm)
113 PRC Envelope #5 Rotated (220 mm x 110 mm)
114 PRC Envelope #6 Rotated (230 mm x 120 mm)
115 PRC Envelope #7 Rotated (230 mm x 160 mm)
116 PRC Envelope #8 Rotated (309 mm x 120 mm)
117 PRC Envelope #9 Rotated (324 mm x 229 mm)
118 PRC Envelope #10 Rotated (458 mm x 324 mm)

Examples

```
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, ], dims = c("C5"))

## 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), row_names = TRUE)
```

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

```
wb_protect(
  wb,
  protect = TRUE,
  password = NULL,
  lock_structure = FALSE,
  lock_windows = FALSE,
  type = 1,
  file_sharing = FALSE,
  username = unname(Sys.info()["user"]),
  read_only_recommended = FALSE,
  ...
)
```

Arguments

wb	A Workbook object
protect	Whether to protect or unprotect the sheet (default TRUE)
password	(optional) password required to unprotect the workbook
lock_structure	Whether the workbook structure should be locked
lock_windows	Whether the window position of the spreadsheet should be locked
type	Lock type (see **Details**)
file_sharing	Whether to enable a popup requesting the unlock password is prompted
username	The username for the file_sharing popup
read_only_recommended	Whether or not a post unlock message appears stating that the workbook is recommended to be opened in read-only mode.
...	additional arguments

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()
wb$add_worksheet("S1")
wb_protect(wb, protect = TRUE, password = "Password", lock_structure = TRUE)

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

wb <- wb_protect(
  wb,
  protect = TRUE,
  password = "Password",
  lock_structure = TRUE,
  type = 2L,
  file_sharing = TRUE,
  username = "Test",
  read_only_recommended = 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. Certain features require applying unlocking of initialized cells in the worksheet and across columns and/or rows.

Usage

```
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()
wb$add_worksheet("S1")
wb$add_data_table(1, x = iris[1:30, ])

wb$protect_worksheet(
  "S1",
  protect = TRUE,
  properties = c("formatCells", "formatColumns", "insertColumns", "deleteColumns")
)

# Formatting cells / columns is allowed , but inserting / deleting columns is protected:
wb$protect_worksheet(
  "S1",
  protect = TRUE,
   c(formatCells = FALSE, formatColumns = FALSE,
                insertColumns = TRUE, deleteColumns = TRUE)
)

# Remove the protection
wb$protect_worksheet("S1", protect = FALSE)
```

---

wb_remove_tables *Remove a data table from a worksheet*

Description

Remove Excel tables in a workbook using its name.

Usage

```
wb_remove_tables(wb, sheet = current_sheet(), table, remove_data = TRUE)
```

Arguments

wb	A Workbook object
sheet	A name or index of a worksheet
table	Name of table to remove. Use wb_get_tables() to view the tables present in the worksheet.
remove_data	Default TRUE. If FALSE, will only remove the data table attributes but will keep the data in the worksheet.

Value

The wbWorkbook, invisibly

Examples

```
wb <- wb_workbook()
wb$add_worksheet(sheet = "Sheet 1")
wb$add_worksheet(sheet = "Sheet 2")
wb$add_data_table(sheet = "Sheet 1", x = iris, table_name = "iris")
wb$add_data_table(sheet = 1, x = mtcars, table_name = "mtcars", start_col = 10)

## delete worksheet removes table objects
wb <- wb_remove_worksheet(wb, sheet = 1)

wb$add_data_table(sheet = 1, x = iris, table_name = "iris")
wb$add_data_table(sheet = 1, x = mtcars, table_name = "mtcars", start_col = 10)

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

wb_get_tables(wb, sheet = 1)
wb$remove_tables(sheet = 1, table = "iris")
```

---

wb_remove_worksheet *Remove a worksheet from a workbook*

Description

Remove a worksheet from a workbook

Usage

```
wb_remove_worksheet(wb, sheet = current_sheet())
```

Arguments

wb	A wbWorkbook object
sheet	The sheet name or index to remove

Value

The wbWorkbook object, invisibly.

Examples

```
## load a workbook
wb <- wb_load(file = system.file("extdata", "openxlsx2_example.xlsx", package = "openxlsx2"))

## Remove sheet 2
wb <- wb_remove_worksheet(wb, 2)
```

---

wb_save *Save a workbook to file*

Description

Save a workbook to file

Usage

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

Arguments

wb	A wbWorkbook object to write to file
file	A path to save the workbook to
overwrite	If FALSE, will not overwrite when file already exists.
path	Deprecated argument. Please use file in new code.

Value

the wbWorkbook object, invisibly

See Also

Other workbook wrappers: base_font-wb, col_widths-wb, creators-wb, grouping-wb, row_heights-wb, wb_add_chartsheet(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_worksheet(), wb_base_colors, wb_clone_worksheet(), wb_copy_cells(), wb_freeze_pane(), wb_merge_cells(), wb_set_last_modified_by(), wb_workbook()

Examples

```
## 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, file = temp_xlsx(), overwrite = TRUE)
```

`wb_set_bookview` *Set the workbook position, size and filter*

Description

Set the workbook position, size and filter

Usage

```
wb_set_bookview(
  wb,
  active_tab = NULL,
  auto_filter_date_grouping = NULL,
  first_sheet = NULL,
  minimized = NULL,
  show_horizontal_scroll = NULL,
  show_sheet_tabs = NULL,
  show_vertical_scroll = NULL,
  tab_ratio = NULL,
  visibility = NULL,
  window_height = NULL,
  window_width = NULL,
  x_window = NULL,
  y_window = NULL,
  ...
)
```

Arguments

`wb`	A wbWorkbook object
`active_tab`	activeTab
`auto_filter_date_grouping`	autoFilterDateGrouping
`first_sheet`	The first sheet to be displayed
`minimized`	minimized

show_horizontal_scroll	showHorizontalScroll
show_sheet_tabs	showSheetTabs
show_vertical_scroll	showVerticalScroll
tab_ratio	tabRatio
visibility	visibility
window_height	windowHeight
window_width	windowWidth
x_window	xWindow
y_window	yWindow
...	additional arguments

Value

The Workbook object

`wb_set_grid_lines` *Modify grid lines visibility in a worksheet*

Description

Set worksheet grid lines to show or hide. You can also add / remove grid lines when creating a worksheet with `wb_add_worksheet(grid_lines = FALSE)`

Usage

```
wb_set_grid_lines(wb, sheet = current_sheet(), show = FALSE, print = show)

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

```
wb <- wb_workbook()$add_worksheet()$add_worksheet()
wb$get_sheet_names() ## list worksheets in workbook
wb$set_grid_lines(1, show = FALSE)
wb$set_grid_lines("Sheet 2", show = FALSE)
```

`wb_set_header_footer` *Set headers and footers of a worksheet*

Description

Set document headers and footers. You can also do this when adding a worksheet with `wb_add_worksheet()` with the `header`, `footer` arguments and friends. These will show up when printing an xlsx file.

Usage

```
wb_set_header_footer(
  wb,
  sheet = current_sheet(),
  header = NULL,
  footer = NULL,
  even_header = NULL,
  even_footer = NULL,
  first_header = NULL,
  first_footer = NULL,
  ...
)
```

Arguments

`wb`	A Workbook object
`sheet`	A name or index of a worksheet
`header`, `even_header`, `first_header`, `footer`, `even_footer`, `first_footer`	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.
`...`	additional arguments

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

Examples

```
wb <- wb_workbook()

# Add example data
wb$add_worksheet("S1")$add_data(x = 1:400)
wb$add_worksheet("S2")$add_data(x = 1:400)
wb$add_worksheet("S3")$add_data(x = 3:400)
wb$add_worksheet("S4")$add_data(x = 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"),
  even_header = c("EVEN HEAD LEFT", "EVEN HEAD CENTER", "EVEN HEAD RIGHT"),
  even_footer = c("EVEN FOOT RIGHT", "EVEN FOOT CENTER", "EVEN FOOT RIGHT"),
  first_header = c("TOP", "OF FIRST", "PAGE"),
  first_footer = c("BOTTOM", "OF FIRST", "PAGE")
)

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

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

wb$set_header_footer(
  sheet = 4,
  first_header = c("FIRST ONLY L", NA, "FIRST ONLY R"),
  first_footer = c("FIRST ONLY L", NA, "FIRST ONLY R")
)
```

---

wb_set_last_modified_by

Modify author in the metadata of a workbook

Description

Just a wrapper of `wb$set_last_modified_by()`

Usage

```
wb_set_last_modified_by(wb, name, ...)
```

Arguments

wb	A workbook object
name	A string object with the name of the LastModifiedBy-User
...	additional arguments

See Also

Other workbook wrappers: base_font-wb, col_widths-wb, creators-wb, grouping-wb, row_heights-wb, wb_add_chartsheet(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_worksheet(), wb_base_colors, wb_clone_worksheet(), wb_copy_cells(), wb_freeze_pane(), wb_merge_cells(), wb_save(), wb_workbook()

Examples

```
wb <- wb_workbook()
wb_set_last_modified_by(wb, "test")
```

---

wb_set_sheetview *Modify the default view of a worksheet*

Description

This helps set a worksheet's appearance, such as the zoom, whether to show grid lines

Usage

```
wb_set_sheetview(
  wb,
  sheet = current_sheet(),
  color_id = NULL,
  default_grid_color = NULL,
  right_to_left = NULL,
  show_formulas = NULL,
  show_grid_lines = NULL,
  show_outline_symbols = NULL,
  show_row_col_headers = NULL,
  show_ruler = NULL,
  show_white_space = NULL,
  show_zeros = NULL,
  tab_selected = NULL,
  top_left_cell = NULL,
  view = NULL,
  window_protection = NULL,
  workbook_view_id = NULL,
  zoom_scale = NULL,
  zoom_scale_normal = NULL,
```

```
  zoom_scale_page_layout_view = NULL,
  zoom_scale_sheet_layout_view = NULL,
  ...
)
```

Arguments

wb	A Workbook object
sheet	sheet
color_id, default_grid_color	Integer: A color, default is 64
right_to_left	Logical: if TRUE column ordering is right to left
show_formulas	Logical: if TRUE cell formulas are shown
show_grid_lines	Logical: if TRUE the worksheet grid is shown
show_outline_symbols	Logical: if TRUE outline symbols are shown
show_row_col_headers	Logical: if TRUE row and column headers are shown
show_ruler	Logical: if TRUE a ruler is shown in page layout view
show_white_space	Logical: if TRUE margins are shown in page layout view
show_zeros	Logical: if FALSE cells containing zero are shown blank if show_formulas = FALSE
tab_selected	Integer: zero vector indicating the selected tab
top_left_cell	Cell: the cell shown in the top left corner / or top right with rightToLeft
view	View: "normal", "pageBreakPreview" or "pageLayout"
window_protection	Logical: if TRUE the panes are protected
workbook_view_id	integer: Pointing to some other view inside the workbook
zoom_scale, zoom_scale_normal, zoom_scale_page_layout_view, zoom_scale_sheet_layout_view	Integer: the zoom scale should be between 10 and 400. These are values for current, normal etc.
...	additional arguments

Value

The wbWorkbook object, invisibly

Examples

```
wb <- wb_workbook()$add_worksheet()

wb$set_sheetview(
  zoom_scale = 75,
  right_to_left = FALSE,
  show_formulas = TRUE,
  show_grid_lines = TRUE,
  show_outline_symbols = FALSE,
  show_row_col_headers = TRUE,
  show_ruler = TRUE,
  show_white_space = FALSE,
  tab_selected = 1,
  top_left_cell = "B1",
  view = "normal",
  window_protection = TRUE
)
```

---

wb_to_df *Create a data frame from a Workbook*

Description

Simple function to create a `data.frame` from a sheet in workbook. Simple as in it was simply written down. `read_xlsx()` and `wb_read()` are just internal wrappers of `wb_to_df()` intended for people coming from other packages.

Usage

```
wb_to_df(
  file,
  sheet,
  start_row = 1,
  start_col = NULL,
  row_names = FALSE,
  col_names = TRUE,
  skip_empty_rows = FALSE,
  skip_empty_cols = FALSE,
  skip_hidden_rows = FALSE,
  skip_hidden_cols = FALSE,
  rows = NULL,
  cols = NULL,
  detect_dates = TRUE,
  na.strings = "#N/A",
  na.numbers = NA,
  fill_merged_cells = FALSE,
  dims,
  show_formula = FALSE,
```

```
  convert = TRUE,
  types,
  named_region,
  keep_attributes = FALSE,
  check_names = FALSE,
  ...
)

read_xlsx(
  file,
  sheet,
  start_row = 1,
  start_col = NULL,
  row_names = FALSE,
  col_names = TRUE,
  skip_empty_rows = FALSE,
  skip_empty_cols = FALSE,
  rows = NULL,
  cols = NULL,
  detect_dates = TRUE,
  named_region,
  na.strings = "#N/A",
  na.numbers = NA,
  fill_merged_cells = FALSE,
  check_names = FALSE,
  ...
)

wb_read(
  file,
  sheet = 1,
  start_row = 1,
  start_col = NULL,
  row_names = FALSE,
  col_names = TRUE,
  skip_empty_rows = FALSE,
  skip_empty_cols = FALSE,
  rows = NULL,
  cols = NULL,
  detect_dates = TRUE,
  named_region,
  na.strings = "NA",
  na.numbers = NA,
  check_names = FALSE,
  ...
)
```

Arguments

file	An xlsx file, wbWorkbook object or URL to xlsx file.
sheet	Either sheet name or index. When missing the first sheet in the workbook is selected.
start_row	first row to begin looking for data.
start_col	first column to begin looking for data.
row_names	If TRUE, the first col of data will be used as row names.
col_names	If TRUE, the first row of data will be used as column names.
skip_empty_rows	If TRUE, empty rows are skipped.
skip_empty_cols	If TRUE, empty columns are skipped.
skip_hidden_rows	If TRUE, hidden rows are skipped.
skip_hidden_cols	If TRUE, hidden columns are skipped.
rows	A numeric vector specifying which rows in the xlsx file to read. If NULL, all rows are read.
cols	A numeric vector specifying which columns in the xlsx file to read. If NULL, all columns are read.
detect_dates	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.
fill_merged_cells	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.
show_formula	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. Names must match the returned data. See **Details** for more.
named_region	Character string with a named_region (defined name or table). If no sheet is selected, the first appearance will be selected. See wb_get_named_regions()
keep_attributes	If TRUE additional attributes are returned. (These are used internally to define a cell type.)
check_names	If TRUE then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names.
...	additional arguments

Details

The returned data frame will have named rows matching the rows of the worksheet. With `col_names = FALSE` the returned data frame will have column names matching the columns of the worksheet. Otherwise the first row is selected as column name.

Depending if the R package `hms` is loaded, `wb_to_df()` returns `hms` variables or string variables in the `hh:mm:ss` format.

The `types` argument must be a named numeric.

- 0: character
- 1: numeric
- 2: date
- 3: posixt (datetime)
- 4: logical

`wb_to_df()` will not pick up formulas added to a workbook object via `wb_add_formula()`. This is because only the formula is written and left to be evaluated when the file is opened in a spreadsheet software. Opening, saving and closing the file in a spreadsheet software will resolve this.

See Also

`wb_get_named_regions()`

Examples

```
################################################################################
# numerics, dates, missings, bool and string
example_file <- system.file("extdata", "openxlsx2_example.xlsx", package = "openxlsx2")
wb1 <- wb_load(example_file)

# import workbook
wb_to_df(wb1)

# do not convert first row to column names
wb_to_df(wb1, col_names = FALSE)

# do not try to identify dates in the data
wb_to_df(wb1, detect_dates = FALSE)

# return the underlying Excel formula instead of their values
wb_to_df(wb1, show_formula = TRUE)

# read dimension without colNames
wb_to_df(wb1, dims = "A2:C5", col_names = FALSE)

# read selected cols
wb_to_df(wb1, cols = c("A:B", "G"))

# read selected rows
wb_to_df(wb1, rows = c(2, 4, 6))
```

```
# convert characters to numerics and date (logical too?)
wb_to_df(wb1, convert = FALSE)

# erase empty rows from dataset
wb_to_df(wb1, skip_empty_rows = TRUE)

# erase empty columns from dataset
wb_to_df(wb1, skip_empty_cols = TRUE)

# convert first row to rownames
wb_to_df(wb1, sheet = 2, dims = "C6:G9", row_names = TRUE)

# define type of the data.frame
wb_to_df(wb1, cols = c(2, 5), types = c("Var1" = 0, "Var3" = 1))

# start in row 5
wb_to_df(wb1, start_row = 5, col_names = FALSE)

# na string
wb_to_df(wb1, na.strings = "a")

#############################################################################
# Named regions
file_named_region <- system.file("extdata", "namedRegions3.xlsx", package = "openxlsx2")
wb2 <- wb_load(file_named_region)

# read dataset with named_region (returns global first)
wb_to_df(wb2, named_region = "MyRange", col_names = FALSE)

# read named_region from sheet
wb_to_df(wb2, named_region = "MyRange", sheet = 4, col_names = FALSE)

# read_xlsx() and wb_read()
example_file <- system.file("extdata", "openxlsx2_example.xlsx", package = "openxlsx2")
read_xlsx(file = example_file)
df1 <- wb_read(file = example_file, sheet = 1)
df2 <- wb_read(file = example_file, sheet = 1, rows = c(1, 3, 5), cols = 1:3)
```

`wb_update_table` *Update a data table position in a worksheet*

Description

Update the position of a data table, possibly written using `wb_add_data_table()`

Usage

```
wb_update_table(wb, sheet = current_sheet(), dims = "A1", tabname)
```

Arguments

wb	A workbook
sheet	A worksheet
dims	Cell range used for new data table.
tabname	A table name

Details

Be aware that this function does not alter any filter. Excluding or adding rows does not make rows appear nor will it hide them.

Examples

```
wb <- wb_workbook()$add_worksheet()$add_data_table(x = mtcars)
wb$update_table(tabname = "Table1", dims = "A1:J4")
```

---

wb_workbook *Create a new Workbook object*

Description

Initialize a wbWorkbook object. You can set workbook properties as well.

Usage

```
wb_workbook(
  creator = NULL,
  title = NULL,
  subject = NULL,
  category = NULL,
  datetime_created = Sys.time(),
  theme = NULL,
  keywords = NULL,
  comments = NULL,
  manager = NULL,
  company = NULL,
  ...
)
```

Arguments

creator	Creator of the workbook (your name). Defaults to login username or options("openxlsx2.creator") if set.
title, subject, category, keywords, comments, manager, company	Workbook property, a string.

datetime_created	The time of the workbook is created
theme	Optional theme identified by string or number. See **Details** for options.
...	additional arguments

Details

theme can be one of "Atlas", "Badge", "Berlin", "Celestial", "Crop", "Depth", "Droplet", "Facet", "Feathered", "Gallery", "Headlines", "Integral", "Ion", "Ion Boardroom", "LibreOffice", "Madison", "Main Event", "Mesh", "Office 2007 - 2010 Theme", "Office 2013 - 2022 Theme", "Office Theme", "Old Office Theme", "Organic", "Parallax", "Parcel", "Retrospect", "Savon", "Slice", "Vapor Trail", "View", "Wisp", "Wood Type"

Value

A wbWorkbook object

See Also

Other workbook wrappers: base_font-wb, col_widths-wb, creators-wb, grouping-wb, row_heights-wb, wb_add_chartsheet(), wb_add_data(), wb_add_data_table(), wb_add_formula(), wb_add_pivot_table(), wb_add_slicer(), wb_add_worksheet(), wb_base_colors, wb_clone_worksheet(), wb_copy_cells(), wb_freeze_pane(), wb_merge_cells(), wb_save(), wb_set_last_modified_by()

Examples

```
## 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"
)
```

---

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, as_table = FALSE, ...)
```

Arguments

x	An object or a list of objects that can be handled by wb_add_data() to write to file.
file	An optional xlsx file name. If no file is passed, the object is not written to disk and only a workbook object is returned.
as_table	If TRUE, will write as a data table, instead of data.
...	Arguments passed on to wb_workbook, wb_add_worksheet, wb_add_data_table, wb_add_data, wb_freeze_pane, wb_set_col_widths, wb_save

creator Creator of the workbook (your name). Defaults to login username or options("openxlsx2.creator") if set.

sheet A name for the new worksheet

grid_lines A logical. If FALSE, the worksheet grid lines will be hidden.

tab_color Color of the sheet tab. A wb_color(), a valid color (belonging to grDevices::colors()) or a valid hex color beginning with "#".

zoom The sheet zoom level, a numeric between 10 and 400 as a percentage. (A zoom value smaller than 10 will default to 10.)

total_row logical. With the default FALSE no total row is added.

start_col A vector specifying the starting column to write x to.

start_row A vector specifying the starting row to write x to.

col_names If TRUE, column names of x are written.

row_names If TRUE, the row names of x are written.

na.strings Value used for replacing NA values from x. Default looks if options(openxlsx2.na.string is set. Otherwise na_strings() uses the special #N/A value within the workbook.

first_active_row Top row of active region

first_active_col Furthest left column of active region

first_row If TRUE, freezes the first row (equivalent to first_active_row = 2)

first_col If TRUE, freezes the first column (equivalent to first_active_col = 2)

widths Width to set cols to specified column width or "auto" for automatic sizing. widths is recycled to the length of cols. openxlsx2 sets the default width is 8.43, as this is the standard in some spreadsheet software. See **Details** for general information on column widths.

overwrite If FALSE, will not overwrite when file already exists.

Details

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

Value

A workbook object

Examples

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

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

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

## different sheets can be given different parameters
write_xlsx(l, temp_xlsx(),
  start_col = c(1, 2, 3), start_row = 2,
  as_table = c(TRUE, TRUE, FALSE), with_filter = c(TRUE, FALSE, FALSE)
)

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

---

xl_open *Open an xlsx file or a* wbWorkbook *object*

Description

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

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

On Mac, (c) it uses system default handlers, given the file type.

On 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

```
xl_open(x, interactive = NA)

## S3 method for class 'wbWorkbook'
xl_open(x, interactive = NA)

## Default S3 method:
xl_open(x, interactive = NA)
```

Arguments

x	A path to the Excel (xls/xlsx) file or wbWorkbook object.
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()) {
  xlsx_file <- system.file("extdata", "openxlsx2_example.xlsx", package = "openxlsx2")
  xl_open(xlsx_file)

  # (not yet saved) Workbook example
  wb <- wb_workbook()
  x <- mtcars[1:6, ]
  wb$add_worksheet("Cars")
  wb$add_data("Cars", x, start_col = 2, start_row = 3, row_names = 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

```
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

```
xml_attr_mod(
  xml_content,
  xml_attributes,
  escapes = FALSE,
  declaration = FALSE,
  remove_empty_attr = TRUE
)
```

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
`remove_empty_attr`	bool remove empty attributes or ignore them

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

```
# 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_attr <- c(foo = "baz", qux = "quux")
  # "<a foo=\"baz\" qux=\"quux\">openxlsx2</a><b foo=\"baz\" qux=\"quux\"/>"
  xml_attr_mod(xml_node, xml_attr)

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

`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