Package: qxl (via r-universe)

October 24, 2024

Title Quick Customized Excel Files

Version 0.0.0.9000

Description A wrapper to the openxlsx package optimized for writing flat data structures.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Depends R (>= 2.10)

Imports openxlsx, readxl, dplyr, rlang, tidyr, lubridate

Suggests testthat (>= 3.0.0), tibble, covr

Repository https://epicentre-msf.r-universe.dev

RemoteUrl https://github.com/epicentre-msf/qxl

RemoteRef HEAD

RemoteSha 3791270366a2aaf38bb60f26bab30f87d6e8f8b9

Contents

expr_to_excel *Translate an R expression into an Excel conditional formatting formula*

Description

Translate an R expression into an Excel conditional formatting formula

Usage

```
expr_to_excel(x, data, row_start = 2L)
```

Arguments

x	An R-style expression
data	A data frame that expression x relates to
row_start	Integer reflecting the first row of data in the Excel sheet to be output. Defaults to 2.

Value

A character string reflecting an Excel conditional formatting formula

Examples

```
library(datasets)

expr_to_excel(cyl > 4, mtcars)
expr_to_excel(cyl > 4 & hp < 200, mtcars)
```

qprotect *Protect one ore more columns of a worksheet*

Description

Wrapper to `openxlsx::protectWorksheet` with additional argument `cols` to enable protection to be limited to specific columns.

In practice, protection is first applied to the entire worksheet, and then subsequently columns not selected for protection (if any) are unlocked one by one. This unlock step (when relevant) also requires a row specification, which by default we limit to the range of the current data. Thus, in 'unprotected' columns within protected worksheets, rows beyond the range of the data will remain protected. As a hack to work around this, the user can specify a 'buffer' of additional empty rows to unprotect within each non-protected column (e.g. to allow further data entry).

Usage

```
qprotect(
  password = NULL,
  cols = everything(),
  row_buffer = 0L,
  protect = TRUE,
  lockSelectingLockedCells = FALSE,
  lockSelectingUnlockedCells = FALSE,
  lockFormattingCells = FALSE,
  lockFormattingColumns = FALSE,
  lockFormattingRows = FALSE,
  lockInsertingColumns = TRUE,
  lockInsertingRows = TRUE,
  lockInsertingHyperlinks = FALSE,
  lockDeletingColumns = TRUE,
  lockDeletingRows = TRUE,
  lockSorting = FALSE,
  lockAutoFilter = FALSE,
  lockPivotTables = NULL,
  lockObjects = NULL,
  lockScenarios = NULL
)
```

Arguments

`password`	(optional) password required to unprotect the worksheet
`cols`	Tidy-selection specifying the columns that protection should apply to. Defaults to `dplyr::everything` to select all columns.
`row_buffer`	The number of additional rows (beyond the range of the current data) to unprotect within columns not specified in argument `cols`. See explanation in Description. Defaults to `0`.
`protect`	Whether to protect or unprotect the sheet (default=TRUE)
`lockSelectingLockedCells`	Whether selecting locked cells is locked
`lockSelectingUnlockedCells`	Whether selecting unlocked cells is locked
`lockFormattingCells`	Whether formatting cells is locked
`lockFormattingColumns`	Whether formatting columns is locked
`lockFormattingRows`	Whether formatting rows is locked
`lockInsertingColumns`	Whether inserting columns is locked
`lockInsertingRows`	Whether inserting rows is locked

`lockInsertingHyperlinks`
: Whether inserting hyperlinks is locked

`lockDeletingColumns`
: Whether deleting columns is locked

`lockDeletingRows`
: Whether deleting rows is locked

`lockSorting`
: Whether sorting is locked

`lockAutoFilter`
: Whether auto-filter is locked

`lockPivotTables`
: Whether pivot tables are locked

`lockObjects`
: Whether objects are locked

`lockScenarios`
: Whether scenarios are locked

`qread` *Read xls and xlsx worksheet*

Description

Wrapper to `readxl::read_excel` with minor changes to default settings:

- columns of dates with no time component have class "Date" rather than "POSIX"
- empty columns are read in as class "character" rather than "logical"
- the max number of rows used to guess column types is 10k rather than 1k

Usage

```
qread(
  path,
  sheet = NULL,
  range = NULL,
  col_names = TRUE,
  col_types = NULL,
  simplify_dates = TRUE,
  empty_cols_to_chr = TRUE,
  na = "",
  trim_ws = TRUE,
  skip = 0,
  n_max = Inf,
  guess_max = min(10000, n_max),
  progress = FALSE,
  .name_repair = "unique"
)
```

Arguments

path	Path to the xls/xlsx file.
sheet	Sheet to read. Either a string (the name of a sheet), or an integer (the position of the sheet). Ignored if the sheet is specified via range. If neither argument specifies the sheet, defaults to the first sheet.
range	A cell range to read from, as described in cell-specification. Includes typical Excel ranges like "B3:D87", possibly including the sheet name like "Budget!B2:G14", and more. Interpreted strictly, even if the range forces the inclusion of leading or trailing empty rows or columns. Takes precedence over skip, n_max and sheet.
col_names	TRUE to use the first row as column names, FALSE to get default names, or a character vector giving a name for each column. If user provides col_types as a vector, col_names can have one entry per column, i.e. have the same length as col_types, or one entry per unskipped column.
col_types	Either NULL to guess all from the spreadsheet or a character vector containing one entry per column from these options: "skip", "guess", "logical", "numeric", "date", "text" or "list". If exactly one col_type is specified, it will be recycled. The content of a cell in a skipped column is never read and that column will not appear in the data frame output. A list cell loads a column as a list of length 1 vectors, which are typed using the type guessing logic from col_types = NULL, but on a cell-by-cell basis.
simplify_dates	Logical indicating whether to convert date columns lacking a time component to class "Date". By default readxl::read_excel reads columns containing dates or datetimes as class POSIX, even if there is no time component (i.e. in which case the times will all be "00:00:00"). If simplify_posix is TRUE (the default), columns containing dates with no nonzero time values are converted to class "Date" using lubridate::as_date.
empty_cols_to_chr	Logical indicating whether columns of class "logical" containing all missing values should be converted to class "character". If argument col_types is NULL (the default), columns containing all missing values are read in by readxl::read_excel as class "logical". If empty_cols_to_chr is TRUE (the default), such columns are converted to class "character".
na	Character vector of strings to interpret as missing values. By default, readxl treats blank cells as missing data.
trim_ws	Should leading and trailing whitespace be trimmed?
skip	Minimum number of rows to skip before reading anything, be it column names or data. Leading empty rows are automatically skipped, so this is a lower bound. Ignored if range is given.
n_max	Maximum number of data rows to read. Trailing empty rows are automatically skipped, so this is an upper bound on the number of rows in the returned tibble. Ignored if range is given.
guess_max	Maximum number of data rows to use for guessing column types.
progress	Display a progress spinner? By default, the spinner appears only in an interactive session, outside the context of knitting a document, and when the call is

likely to run for several seconds or more. See `readxl_progress()` for more details.

`.name_repair` Handling of column names. Passed along to `tibble::as_tibble()`. readxl's default is '.name_repair = "unique", which ensures column names are not empty and are unique.

`qstyle` *Conditional cell styles*

Description

Wrapper to `openxlsx::createStyle` to create cell styles, with additional arguments `rows` and `cols` to specify the rows and/or columns that the style should apply to.

Usage

```
qstyle(
  rows = "data",
  cols = everything(),
  fontName = NULL,
  fontSize = NULL,
  fontColour = NULL,
  border = NULL,
  borderColour = getOption("openxlsx.borderColour", "black"),
  borderStyle = getOption("openxlsx.borderStyle", "thin"),
  bgFill = NULL,
  fgFill = NULL,
  halign = NULL,
  valign = NULL,
  textDecoration = NULL,
  wrapText = FALSE,
  textRotation = NULL,
  indent = NULL,
  locked = NULL,
  hidden = NULL
)
```

Arguments

`rows` Which rows the style should apply to. Can be set using either:

Keyword: (e.g. `rows = "data"` or `rows = "all"`)
Keyword "data" (the default) applies a style to all data rows (excludes the header), whereas keyword "all" applies a style to all rows (header and data)

Integer rows indexes: (e.g. `rows = c(2, 5, 6)`)
Note that in this case indexes represent Excel rows rather than R rows (i.e. the header is row 1).

An expression: (e.g. `rows = cyl > 4`)
Given an expression the style is applied using conditional formatting, with the expression translated into its Excel formula equivalent.

Expressions can optionally include a `.x` selector (e.g. `.x == 1`) to refer to multiple columns. See section **Using a .x selector** below.

Note that conditional formatting can update in real time if relevant data is changed within the workbook.

`cols` Tidy-selection specifying the columns that the style should apply to. Defaults to `dplyr::everything` to select all columns.

`fontName` A name of a font. Note the font name is not validated. If fontName is NULL, the workbook base font is used. (Defaults to Calibri)

`fontSize` Font size. A numeric greater than 0. If fontSize is NULL, the workbook base font size is used. (Defaults to 11)

`fontColour` Colour of text in cell. A valid hex colour beginning with "#" or one of colours(). If fontColour is NULL, the workbook base font colours is used. (Defaults to black)

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

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

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

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

- **none** No Border
- **thin** thin border
- **medium** medium border
- **dashed** dashed border
- **dotted** dotted border
- **thick** thick border
- **double** double line border

- **hair** Hairline border
- **mediumDashed** medium weight dashed border
- **dashDot** dash-dot border
- **mediumDashDot** medium weight dash-dot border
- **dashDotDot** dash-dot-dot border
- **mediumDashDotDot** medium weight dash-dot-dot border
- **slantDashDot** slanted dash-dot border

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

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

`halign` Horizontal alignment of cell contents

- **left** Left horizontal align cell contents
- **right** Right horizontal align cell contents
- **center** Center horizontal align cell contents
- **justify** Justify horizontal align cell contents

`valign` A name Vertical alignment of cell contents

- **top** Top vertical align cell contents
- **center** Center vertical align cell contents
- **bottom** Bottom vertical align cell contents

`textDecoration` Text styling.

- **bold** Bold cell contents
- **strikeout** Strikeout cell contents
- **italic** Italicise cell contents
- **underline** Underline cell contents
- **underline2** Double underline cell contents
- **accounting** Single accounting underline cell contents
- **accounting2** Double accounting underline cell contents

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

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

`indent` Horizontal indentation of cell contents.

`locked` Whether cell contents are locked (if worksheet protection is turned on)

`hidden` Whether the formula of the cell contents will be hidden (if worksheet protection is turned on)

Using a .x selector

An expression passed to the `rows` argument can optionally incorporate a `.x` selector to refer to multiple columns within the worksheet.

When a `.x` selector is used, each column specified in arguments `cols` is independently swapped into the `.x` position of the expression, which is then translated to the Excel formula equivalent and applied as conditional formatting to the worksheet.

For example, given the following `qstyle` specification with respect to the `mtcars` dataset

```
qstyle(
  rows = .x == 1,
  cols = c(vs, am, carb),
  bgFill = "#FFC7CE"
)
```

the style `bgFill = "#FFC7CE"` would be independently applied to any cell in columns `vs`, `am`, or `carb` with a value of `1`.

Examples

```
# apply style halign = "center" to all data rows (by default rows = "data")
qstyle(halign = "center")

# apply style halign = "center" to all rows including header
qstyle(rows = "all", halign = "center")

# apply style halign = "center" to Excel rows 2:10
qstyle(rows = 2:10, halign = "center")

# apply conditional formatting to rows where cyl == 8 & mpg > 16
qstyle(cyl == 8 & mpg > 16, fgFill = "#fddbc7", textDecoration = "bold")
```

`qwrite` *Write workbook to an xlsx file*

Description

Wrapper to `openxlsx::saveWorkbook` to write an Excel workbook to file

Usage

```
qwrite(wb, file, overwrite = FALSE)
```

Arguments

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

qxl *Quickly write a tidy data frame to xlsx, with options for customization*

Description

A wrapper to the openxlsx package optimized for writing tidy data frames. Includes arguments to quickly add customization like:

- conditional formatting written as R expressions
- data validation rules based on a tidy dictionary structure
- column-specific worksheet protection
- custom column names with original variable-names hidden in the row below

Usage

```
qxl(
  x,
  file = NULL,
  wb = openxlsx::createWorkbook(),
  sheet = NULL,
  header = NULL,
  style_head = qstyle(rows = 1, textDecoration = "bold"),
  hide_subhead = TRUE,
  style1 = NULL,
  style2 = NULL,
  style3 = NULL,
  style4 = NULL,
  style5 = NULL,
  group,
  group_style = qstyle(bgFill = "#ffcccb"),
  row_heights = NULL,
  col_widths = "auto",
  freeze_row = 1L,
  freeze_col = NULL,
  protect,
  validate = NULL,
  validate_cond = NULL,
  validate_cond_all = NULL,
  filter = FALSE,
  filter_cols = everything(),
  zoom = 120L,
  date_format = "yyyy-mm-dd",
  overwrite = TRUE
)
```

Arguments

x	A data frame, or list of data frames
file	Filename to write to. If NULL the resulting workbook is returned as an openxlsx object of class "Workbook" rather than written to a file.
wb	An openxlsx workbook object to write to. Defaults to a fresh workbook created with openxlsx::createWorkbook. Only need to update when repeatedly calling qxl() to add worksheets to an existing workbook.
sheet	Optional character vector of worksheet names. If NULL (the default) and x is a named list of data frames, worksheet names are taken from names(x). Otherwise, names default to "Sheet1", "Sheet2", ...

header	Optional column header. Defaults to NULL in which case column names are taken directly from the data frame(s) in x, to create normal single-row headers. Can alternatively pass a named character vector to set custom names as the first row and a subheader with variable names as a hidden second row.
	
	`header = c(`
	`  mpg = "Miles per US gallon",`
	`  cyl = "Number of cylinders",`
	`  disp = "Engine displacement (cubic in.)`
	`)`

style_head	Style for the header row. Set with qstyle(), or set to NULL for no header styling. Defaults to bold text.
hide_subhead	Logical indicating whether to hide the subheader (if present). Defaults to TRUE.
style1, style2, style3, style4, style5	Optional style to set using qstyle()
group	Optional vector of one or more column names used to create alternating groupings of rows, with every other row grouping styled as per argument group_style. See section **Grouping rows**.
group_style	Optional style to apply to alternating groupings of rows, as specified using argument groups. Set using qstyle()
row_heights	Numeric vector of row heights (in Excel units). The vector is recycled if shorter than the number of rows in x. Defaults to NULL to use default row heights.

col_widths	Vector of column widths (in Excel units). Can be numeric or character, and may include keyword "auto" for automatic column sizing. The vector is recycled if shorter than the number of columns in x. Defaults to "auto".
	Use named vector to give column widths for specific columns, where names represent column names of x or the keyword ".default" to set a default column width for all columns not otherwise specified. E.g.
	
	`# specify widths for cols mpg and cyl, all others default to "auto"`
	`col_widths <- c(mpg = 5, cyl = 10)`
	
	`# specify widths for cols mpg and cyl, and explicit default for all others`
	`col_widths <- c(mpg = 5, cyl = 10, .default = 7)`

`freeze_row` Integer specifying a row to freeze at. Defaults to `1` to add a freeze below the header row. Set to `0` or `NULL` to omit freezing.

`freeze_col` Integer specifying a column to freeze at. Defaults to `NULL`. Set to `0` or `NULL` to omit freezing.

`protect` Optional function specifying how to protect worksheet components from user modification. See function `qprotect`.

`validate` Optional specification of list-style data validation for one or more columns. Can specify either as a list of vectors giving options for one or more column in `x`, e.g.:

```
list(
  var_x = c("Yes", "No"),
  var_y = c("Small", "Medium", "Large")
)
```

or as a data.frame where the first column gives column names and the second column gives corresponding options, e.g.:

```
data.frame(
  col = c("var_x", "var_x", "var_y", "var_y", "var_y"),
  val = c("Yes", "No", "Small", "Medium", "Large")
)
```

Validation options are written/appended to a hidden worksheet named "valid_options".

`validate_cond` Optional specification of conditional list-style validation, where the set of values to be allowed in a given column depends on the corresponding value within one or more other columns (e.g. the allowed values in column 'city' depend on the corresponding value in columns 'country' and 'province'). Must be a data.frame with at least two columns, where the first column(s) give the conditional entries (e.g. 'country', 'province') and the last column gives the corresponding allowed entries (e.g. 'city') to be implemented as data validation. The column names in `validate_cond` should match the relevant columns within `x`.

Note that in the current implementation validation is based on values in the conditional column(s) of `x` at the time the workbook is written, and will not update in real time if those values are later changed.

`validate_cond_all` Optional vector of value(s) to always allow, independent of the value in the conditional column (e.g. "Unknown").

`filter` Logical indicating whether to add column filters.

`filter_cols` Tidy-selection specifying which columns to filter. Only used if `filter` is `TRUE`. Defaults to `everything()` to select all columns.

`zoom` Integer specifying initial zoom percentage. Defaults to 130.

`date_format` Excel format for date columns. Defaults to "yyyy-mm-dd".

`overwrite` Logical indicating whether to overwrite existing file. Defaults to `TRUE`

Value

If argument `file` is not specified, returns an openxlsx workbook object. Otherwise writes workbook to file with no return.

Grouping rows

Given a dataset with multiple rows per group (e.g. repeated observations on a given individual), it can sometimes be useful to uniquely stylize alternating groups to allow for quick visual distinction of the rows belonging to any given group.

Given one or more grouping columns specified using argument `groups`, the `qxl` function arranges the rows of the resulting worksheet by group and then applies the style `group_style` to the rows in every *other* group, to create an alternating pattern. The alternating pattern is achieved by first creating a group index variable called `g` which is assigned a value of either `1` or `0`: `1` for the 1st group, `0` for the 2nd, `1` for the 3rd, `0` for the 4th, etc. The style specified by `group_style` is then applied conditionally to rows where `g == 0`. The grouping variable is written in column A, which is hidden.

Examples

```
library(datasets)
qxl(mtcars, file = tempfile(fileext = ".xlsx"))
```

Index