Package 'tidyxl'

November 16, 2020

Title Read Untidy Excel Files

Version 1.0.7

Description Imports non-tabular from Excel files into R. Exposes cell content, position and formatting in a tidy structure for further manipulation. Tokenizes Excel formulas. Supports '.xlsx' and '.xlsm' via the embedded 'RapidXML' C++ library <http://rapidxml.sourceforge.net>. Does not support '.xlsb' or '.xls'.

Depends R (>= 3.2.0)

License GPL-3

Encoding UTF-8

LazyData true

LinkingTo Rcpp, piton (>= 1.0.0)

Imports Rcpp

URL https://github.com/nacnudus/tidyxl

BugReports https://github.com/nacnudus/tidyxl/issues

RoxygenNote 7.1.1

Suggests testthat, here, knitr, rmarkdown, readxl, dplyr, tidyr, purrr, tibble, ggplot2, cellranger, openxlsx, rlang

VignetteBuilder knitr

NeedsCompilation yes

Author Duncan Garmonsway [aut, cre], Hadley Wickham [ctb] (Author of included readxl fragments), Jenny Bryan [ctb] (Author of included readxl fragments), RStudio [cph] (Copyright holder of included readxl fragments), Marcin Kalicinski [ctb, cph] (Author of included RapidXML code)

Maintainer Duncan Garmonsway <nacnudus@gmail.com>

Repository CRAN

Date/Publication 2020-11-16 09:20:03 UTC

R topics documented:

`excel_functions` *Names of all Excel functions*

Description

A dataset containing the names of all functions available in Excel. This is useful for identifying user-defined functions in formulas tokenized by `xlex()`.

Usage

```
excel_functions
```

Format

A character vector of length 600.

Details

Note that this includes future function names that are already reserved.

Source

Pages 26–27 of Microsoft's document "Excel (.xlsx) extensions to the office openxml spreadsheetml file format p.24" https://docs.microsoft.com/en-us/openspecs/office_standards/ms-xlsx/2c5dee00-eff2-4b22-92b6-0738acd4475e, revision 8.0 2017-06-20.

is_date_format *Test that Excel number formats are date formats*

Description

`is_date_format()` tests whether an Excel number format string would a resolve to a date in Excel. For example, the number format string `"yyyy-mm-dd"` would resolve to a date, whereas the string `"0.0\\%"` would not.

This is useful if a cell formula contains a number formatting string (e.g. `TEXT(45678,"yyyy")`), and you need to know that the constant 45678 is a date in order to recover it at full resolution (rather than parsing the character output "2025" as a year).

It is used internally to convert the value of a cell to the correct data type.

Usage

```
is_date_format(x)
```

Arguments

`x`	character vector of number format strings

Examples

```
is_date_format(c("yyyy-mm-dd", "0.0%", "h:m:s", "£#,##0;[Red]-£#,##0"))
```

is_range *Test that Excel formulas are ranges*

Description

`is_range()` tests whether or not an Excel formula is a range. A formula like `A1` is a range, whereas a formula like `MAX(A1,2)` is not. Formulas are not evaluated, so it returns `FALSE` for formulas that would eventually resolve to arrange (e.g. `INDEX(A1:A10,2)`) but that are not immediately a range.

Usage

```
is_range(x)
```

Arguments

`x`	character vector of formulas

Examples

```
x <- c("A1", "Sheet1!A1", "[0]Sheet1!A1", "A1,A2", "A:A 3:3", "MAX(A1,2)")
is_range(x)
```

`maybe_xlsx` *Determine file format*

Description

Whether a file may be be xlsx, xlsm, xltx or xltm (rather than xls or xlt), based on the file signature or "magic number", rather than the filename extension.

Usage

```
maybe_xlsx(path)
```

Arguments

`path`	File path

Details

Only 'maybe', not 'is', because the xlsx magic number is common to all zip files, not specific to xlsx files. The inverse, 'is_xls' isn't possible either, because the xls magic number is common to other Microsoft Office files such as .doc and .ppt.

This uses some logic from Jenny Bryan's commit to the readxl package.

Value

Logicial

Examples

```
examples_xlsx <- system.file("extdata/examples.xlsx", package = "tidyxl")
examples_xlsm <- system.file("extdata/examples.xlsm", package = "tidyxl")
examples_xltx <- system.file("extdata/examples.xltx", package = "tidyxl")
examples_xltm <- system.file("extdata/examples.xltm", package = "tidyxl")
examples_xlsb <- system.file("extdata/examples.xlsb", package = "tidyxl")
examples_xls <- system.file("extdata/examples.xls", package = "tidyxl")

maybe_xlsx(examples_xlsx)
maybe_xlsx(examples_xlsm)
maybe_xlsx(examples_xltx)
maybe_xlsx(examples_xltm)
maybe_xlsx(examples_xlsb)
maybe_xlsx(examples_xls)
```

tidyxl — *tidyxl: Import xlsx (Excel) spreadsheet data and formatting into tidy structures.*

Description

Tidyxl imports data from spreadsheets without coercing it into a rectangle, and retains information encoded in cell formatting (e.g. font/fill/border). This data structure is compatible with the 'unpivotr' package for recognising information expressed by relative cell positions and cell formatting, and re-expressing it in a tidy way.

Details

- `xlsx_cells()` Import cells from an xlsx file.
- `xlsx_formats()` Import formatting from an xlsx file.
- `xlsx_sheet_names()` List the names of sheets in an xlsx file.
- `xlsx_names()` Import names and definitions of named ranges (aka 'named formulas', 'defined names') from an xlsx file.
- `is_range()` Test whether a 'name' from `xlsx_names()` refers to a range or not.
- `xlsx_validation()` Import cell input validation rules (e.g. 'must be from this drop-down list') from an xlsx file.
- `xlsx_colour_standard()` A data frame of standard colour names and their RGB values.
- `xlsx_colour_theme()` Imports a data frame of theme colour names and their RGB values from an xlsx file.
- `xlex()` Tokenise (lex) an Excel formula.

tidy_xlsx — *Import xlsx (Excel) cell contents into a tidy structure.*

Description

`tidy_xlsx()` is deprecated. Please use `xlsx_cells()` or `xlsx_formats()` instead.

`tidy_xlsx()` imports data from spreadsheets without coercing it into a rectangle. Each cell is represented by a row in a data frame, giving the cell's address, contents, formula, height, width, and keys to look up the cell's formatting in an adjacent data structure within the list returned by this function.

Usage

```
tidy_xlsx(path, sheets = NA)
```

Arguments

path	Path to the xlsx file.
sheets	Sheets to read. Either a character vector (the names of the sheets), an integer vector (the positions of the sheets), or NA (default, all sheets).

Details

A cell has two 'values': its content, and sometimes also a formula. It also has formatting applied at the 'style' level, which can be locally overridden.

Content: Depending on the cell, the content may be a numeric value such as 365 or 365.25, it may represent a date/datetime in one of Excel's date/datetime systems, or it may be an index into an internal table of strings. `tidy_xlsx()` attempts to infer the correct data type of each cell, returning its value in the appropriate column (error, logical, numeric, date, character). In case this cleverness is unhelpful, the unparsed value and type information is available in the 'content' and 'type' columns.

Formula: When a cell has a formula, the value in the 'content' column is the result of the formula the last time it was evaluated.

Certain groups of cells may share a formula that differs only by addresses referred to in the formula; such groups are identified by an index, the 'formula_group'. The xlsx (Excel) file format only records the formula against one cell in any group, but `tidy_xlsx()` propagates the formula to all the cells in the group, making the necessary changes to relative addresses in the formula.

Array formulas may also apply to a group of cells, identified by an address 'formula_ref', but xlsx (Excel) file format only records the formula against one cell in the group. Unlike ordinary formulas, `tidy_xlsx()` does not propagate these to the other cells in the group.

Formulas that refer to other workbooks currently do not name the workbooks directly, instead via indices such as [1]. It is planned to dereference these.

Formatting: Cell formatting is returned in `x$formats`. There are two types of formatting: 'style' formatting, such as Excel's built-in styles 'normal', 'bad', etc., and 'local' formatting, which overrides the style. These are returned in `x$formats$style` and `x$formats$local`, with identical structures. To look up the local formatting of a given cell, take the cell's 'local_format_id' value (`x$data$Sheet1[1,"local_format_id"]`), and use it as an index into the format structure. E.g. to look up the font size, `x$formats$local$font$size[local_format_id]`. To see all available formats, type `str(x$formats$local)`.

Value

A list of the data within each sheet ($data), and the formatting applied to each cell ($formats).

Each sheet's data is returned as a data frames, one per sheet, by the sheet name. For example, the data in a sheet named 'My Worksheet' is in x$data$My Worksheet. Each data frame has the following columns:

- `address` The cell address in A1 notation.
- `row` The row number of a cell address (integer).
- `col` The column number of a cell address (integer).

- `is_blank` Whether or not the cell has a value
- `data_type` The type of a cell, referring to the following columns: error, logical, numeric, date, character, blank.
- `error` The error value of a cell.
- `logical` The boolean value of a cell.
- `numeric` The numeric value of a cell.
- `date` The date value of a cell.
- `character` The string value of a cell.
- `character_formatted` A data frame of substrings and their individual formatting.
- `formula` The formula in a cell (see 'Details').
- `is_array` Whether or not the formula is an array formula.
- `formula_ref` The address of a range of cells group to which an array formula or shared formula applies (see 'Details').
- `formula_group` The formula group to which the cell belongs (see 'Details').
- `comment` The text of a comment attached to a cell.
- `height` The height of a cell's row, in Excel's units.
- `width` The width of a cell's column, in Excel's units.
- `style_format` An index into a table of style formats `x$formats$style` (see 'Details').
- `local_format_id` An index into a table of local cell formats `x$formats$local` (see 'Details').

Formula: When a cell has a formula, the value in the 'content' column is the result of the formula the last time it was evaluated.

Certain groups of cells may share a formula that differs only by addresses referred to in the formula; such groups are identified by an index, the 'formula_group'. The xlsx (Excel) file format only records the formula against one cell in any group. `xlsx_cells()` propagates such formulas to the other cells in a group, making the necessary changes to relative addresses in the formula.

Array formulas may also apply to a group of cells, identified by an address 'formula_ref', but xlsx (Excel) file format only records the formula against one cell in the group. `xlsx_cells()` propagates such formulas to the other cells in a group. Unlike shared formulas, no changes to addresses in array formulas are necessary.

Formulas that refer to other workbooks currently do not name the workbooks directly, instead via indices such as [1]. It is planned to dereference these.

Formatting: Cell formatting is returned in `x$formats`. There are two types or scopes of formatting: 'style' formatting, such as Excel's built-in styles 'normal', 'bad', etc., and 'local' formatting, which overrides particular elements of the style, e.g. by making it bold. Both types of are returned in `x$formats$style` and `x$formats$local`, with identical structures. To look up the local formatting of a given cell, take the cell's 'local_format_id' value (`x$data$Sheet1[1,"local_format_id"]`), and use it as an index into the format structure. E.g. to look up the font size, `x$formats$local$font$size[local_format`
To see all available formats, type `str(x$formats$local)`.

Colours may be recorded in any of three ways: a hexadecimal RGB string with or without alpha, an 'indexed' colour, and an index into a 'theme'. `tidy_xlsx` dereferences 'indexed' and 'theme'

colours to their hexadecimal RGB string representation, and standardises all RGB strings to have an alpha channel in the first two characters. The 'index' and the 'theme' name are still provided. To filter by an RGB string, you could look up the RGB values in a spreadsheet program (e.g. Excel, LibreOffice, Gnumeric), and use the `grDevices::rgb()` function to convert these to a hexadecimal string. Put the alpha value in first, e.g.

```
A <- 1; R <- 0.5; G <- 0; B <- 0
rgb(A, R, G, B)
# [1] "#FF800000"
```

Strings can be formatted within a cell, so that a single cell can contain substrings with different formatting. This in-cell formatting is available in the column `character_formatted`, which is a list-column of data frames. Each row of each data frame describes a substring and its formatting. For cells without a character value, `character_formatted` is `NULL`, so for further processing you might need to filter out the `NULL`s first.

Examples

```
## Not run:
examples <- system.file("extdata/examples.xlsx", package = "tidyxl")

# All sheets
str(tidy_xlsx(examples)$data)

# Specific sheet either by position or by name
str(tidy_xlsx(examples, 2)$data)
str(tidy_xlsx(examples, "Sheet1")$data)

# Data (cell values)
x <- tidy_xlsx(examples)
str(x$data$Sheet1)

# Formatting
str(x$formats$local)

# The formats of particular cells can be retrieved like this:

Sheet1 <- x$data$Sheet1
x$formats$style$font$bold[Sheet1$style_format]
x$formats$local$font$bold[Sheet1$local_format_id]

# To filter for cells of a particular format, first filter the formats to get
# the relevant indices, and then filter the cells by those indices.
bold_indices <- which(x$formats$local$font$bold)
Sheet1[Sheet1$local_format_id %in% bold_indices, ]

# In-cell formatting is available in the `character_formatted` column as a
# data frame, one row per substring.
tidy_xlsx(examples)$data$Sheet1$character_formatted[77]

## End(Not run)
```

xlex *Parse xlsx (Excel) formulas into tokens*

Description

xlex takes an Excel formula and separates it into tokens. The name is a bad pun on 'Excel' and 'lexer'. It returns a dataframe, one row per token, giving the token itself, its type (e.g. number, or error), and its level.

The level is a number to show the depth of a token within nested function calls. The token A2 in the formula IF(A1=1,A2,MAX(A3,A4)) is at level 1. Tokens A3 and A4 are at level 2. The token IF is at level 0, which is the outermost level.

The output isn't enough to enable computation or validation of formulas, but it is enough to investigate the structure of formulas and spreadsheets. It has been tested on millions of formulas in the Enron corpus.

Usage

```
xlex(x)
```

Arguments

x	Character vector of length 1, giving the formula.

Details

The different types of tokens are:

- ref A cell reference/address e.g. A1 or $B2:C$14.
- sheetA sheet name, e.g. Sheet1! or 'My Sheet'!. If the sheet is from a different file, then the file is included in this token – usually it has been normalized to the form [0].
- name A named range, or more properly a named formula.
- function An Excel or user-defined function, e.g. MAX or _xll.MY_CUSTOM_FUNCTION. A complete list of official Excel functions is available in the vector excel_functions.
- error An error, e.g. #N/A or #REF!.
- bool TRUE or FALSE – note that there are also functions TRUE() and FALSE().
- number All forms of numbers, e.g. 1, 1.1, -1, 1.2E3.
- text Strings inside double quotes, e.g. "Hello,World!".
- operator The usual infix operators, +, -, *, /, ^, <, <=, <>, etc. and also the range operator : when it is used with ranges that aren't cell addresses, e.g. INDEX(something):A1. The union operator , is the same symbol that is used to separate function arguments and array columns, so it is only tagged operator when it is inside parentheses that are not function parentheses or array curly braces (see the examples).
- paren_open An open parenthesis (indicating an increase in the level of nesting, but not directly enclosing function arguments.

- `paren_close` As `open`, but reducing the level of nesting.
- `open_array` An open curly brace '{' indicating the start of an array of constants, and an increase in the level of nesting.
- `close_array` As `open_array`, but ending the array of constants
- `fun_open` An open parenthesis `(` immediately after a function name, directly enclosing the function arguments.
- `fun_close` As `fun_open` but immediately after the function arguments.
- `separator` A comma , separating function arguments or array columns, or a semicolon ; separating array rows.
- `DDE` A call to a Dynamic Data Exchange server, usually normalized to the form [1]!'DDE_parameter=1', but the full form is 'ABCD'|'EFGH'!'IJKL'.
- `space` Some old files haven't stripped formulas of meaningless spaces. They are returned as `space` tokens so that the original formula can always be reconstructed by concatenating all tokens.
- `other` If you see this, then something has gone wrong – please report it at https://github.com/nacnudus/tidyxl/issues with a reproducible example (e.g. using the reprex package).

Every part of the original formula is returned as a token, so the original formula can be reconstructed by concatenating the tokens. If that doesn't work, please report it at https://github.com/nacnudus/tidyxl/issues with a reproducible example (e.g. using the reprex package).

The XLParser project was a great help in creating the grammar. https://github.com/spreadsheetlab/XLParser.

Value

A data frame (a tibble, if you use the tidyverse) one row per token, giving the token itself, its type (e.g. `number`, or `error`), and its level.

A class attribute `xlex` is added, so that the `base::print()` generic can be specialised to print the tree prettily.

Examples

```
# All explicit cell references/addresses are returned as a single 'ref' token.
xlex("A1")
xlex("A$1")
xlex("$A1")
xlex("$A$1")
xlex("A1:B2")
xlex("1:1") # Whole row
xlex("A:B") # Whole column

# If one part of an address is a name or a function, then the colon ':' is
# regarded as a 'range operator', so is tagged 'operator'.
xlex("A1:SOME.NAME")
xlex("SOME_FUNCTION():B2")
xlex("SOME_FUNCTION():SOME.NAME")

# Sheet names are recognised by the terminal exclamation mark '!'.
xlex("Sheet1!A1")
```

```
xlex("'Sheet 1'!A1")       # Quoted names may contain some punctuation
xlex("'It''s a sheet'!A1") # Quotes are escaped by doubling

# Sheets can be ranged together in so-called 'three-dimensional formulas'.
# Both sheets are returned in a single 'sheet' token.
xlex("Sheet1:Sheet2!A1")
xlex("'Sheet 1:Sheet 2'!A1") # Quotes surround both (rather than each) sheet

# Sheets from other files are prefixed by the filename, which Excel
# normalizes the filenames into indexes.  Either way, xlex() includes the
# file/index in the 'sheet' token.
xlex("[1]Sheet1!A1")
xlex("'[1]Sheet 1'!A1") # Quotes surround both the file index and the sheet
xlex("'C:\\My Documents\\[file.xlsx]Sheet1'!A1")

# Function names are recognised by the terminal open-parenthesis '('.  There
# is no distinction between custom functions and built-in Excel functions.
# The open-parenthesis is tagged 'fun_open', and the corresponding
# close-parenthesis at the end of the arguments is tagged 'fun_close'.
xlex("MAX(1,2)")
xlex("_xll.MY_CUSTOM_FUNCTION()")

# Named ranges (properly called 'named formulas') are a last resort after
# attempting to match a function (ending in an open parenthesis '(') or a
# sheet (ending in an exclamation mark '!')
xlex("MY_NAMED_RANGE")

# Some cell addresses/references, functions and names can look alike, but
# xlex() should always make the right choice.
xlex("XFD1")     # A cell in the maximum column in Excel
xlex("XFE1")     # Beyond the maximum column, must be a named range/formula
xlex("A1048576") # A cell in the maximum row in Excel
xlex("A1048577") # Beyond the maximum row, must be a named range/formula
xlex("LOG10")    # A cell address
xlex("LOG10()")  # A log function
xlex("LOG:LOG")  # The whole column 'LOG'
xlex("LOG")      # Not a cell address, must be a named range/formula
xlex("LOG()")    # Another log function
xlex("A1.2!A1")  # A sheet called 'A1.2'

# Text is surrounded by double-quotes.
xlex("\"Some text\"")
xlex("\"Some \"\"text\"\"\"") # Double-quotes within text are escaped by

# Numbers are signed where it makes sense, and can be scientific
xlex("1")
xlex("1.2")
xlex("-1")
xlex("-1-1")
xlex("-1+-1")
xlex("MAX(-1-1)")
xlex("-1.2E-3")
```

```
# Booleans can be constants or functions, and names can look like booleans,
# but xlex() should always make the right choice.
xlex("TRUE")
xlex("TRUEISH")
xlex("TRUE!A1")
xlex("TRUE()")

# Errors are tagged 'error'
xlex("#DIV/0!")
xlex("#N/A")
xlex("#NAME?")
xlex("#NULL!")
xlex("#NUM!")
xlex("#REF!")
xlex("#VALUE!")

# Operators with more than one character are treated as single tokens
xlex("1<>2")
xlex("1<=2")
xlex("1<2")
xlex("1=2")
xlex("1&2")
xlex("1 2")
xlex("(1,2)")
xlex("1%")   # postfix operator

# The union operator is a comma ',', which is the same symbol that is used
# to separate function arguments or array columns.  It is tagged 'operator'
# only when it is inside parentheses that are not function parentheses or
# array curly braces.  The curly braces are tagged 'array_open' and
# 'array_close'.
tidyxl::xlex("A1,B2") # invalid formula, defaults to 'union' to avoid a crash
tidyxl::xlex("(A1,B2)")
tidyxl::xlex("MAX(A1,B2)")
tidyxl::xlex("SMALL((A1,B2),1)")

# Function arguments are separated by commas ',', which are tagged
# 'separator'.
xlex("MAX(1,2)")

# Nested functions are marked by an increase in the 'level'.  The level
# increases inside parentheses, rather than at the parentheses.  Curly
# braces, for arrays, have the same behaviour, as do subexpressions inside
# ordinary parenthesis, tagged 'paren_open' and 'paren_close'.  To see the
# levels explicitly (rather than by the pretty printing), print as a normal
# data frame or tibble by specifying `pretty = FALSE`.
# class with as.data.frame.
xlex("MAX(MIN(1,2),3)")
xlex("{1,2;3,4}")
xlex("1*(2+3)")
print(xlex("1*(2+3)"), pretty = FALSE)

# Arrays are marked by opening and closing curly braces, with comma ','
```

```
# between columns, and semicolons ';' between rows  Commas and semicolons are
# both tagged 'separator'.  Arrays contain only constants, which are
# booleans, numbers, text, and errors.
xlex("MAX({1,2;3,4})")
xlex("=MAX({-1E-2,TRUE;#N/A,\"Hello, World!\"})")

# Structured references are surrounded by square brackets.  Subexpressions
# may also be surrounded by square brackets, but xlex() returns the whole
# expression in a single 'structured_ref' token.
xlex("[@col2]")
xlex("SUM([col22])")
xlex("Table1[col1]")
xlex("Table1[[col1]:[col2]]")
xlex("Table1[#Headers]")
xlex("Table1[[#Headers],[col1]]")
xlex("Table1[[#Headers],[col1]:[col2]]")

# DDE calls (Dynamic Data Exchange) are normalized by Excel into indexes.
# Either way, xlex() includes everything in one token.
xlex("[1]!'DDE_parameter=1'")
xlex("'Quote'|'NYSE'!ZAXX")
# Meaningless spaces that appear in some old files are returned as 'space'
# tokens, so that the original formula can still be recovered by
# concatenating all the tokens.  Spaces between function names and their open
# parenthesis have not been observed, so are not permitted.
xlex(" MAX( A1 ) ")

# print.xlex() invisibly returns the original argument, so that it can be
# used in magrittr pipelines.
str(print(xlex("ROUND(A1*2")))
```

`xlsx_cells` *Import xlsx (Excel) cell contents into a tidy structure.*

Description

`xlsx_cells()` imports data from spreadsheets without coercing it into a rectangle. Each cell is represented by a row in a data frame, giving the cell's address, contents, formula, height, width, and keys to look up the cell's formatting in the return value of `xlsx_formats()`.

Usage

```
xlsx_cells(
  path,
  sheets = NA,
  check_filetype = TRUE,
  include_blank_cells = TRUE
)
```

Arguments

path	Path to the xlsx file.
sheets	Sheets to read. Either a character vector (the names of the sheets), an integer vector (the positions of the sheets), or NA (default, all sheets).
check_filetype	Logical. Whether to check that the filetype is xlsx (or xlsm) by looking at the file itself, rather than using the filename extension.
include_blank_cells	Logical. Whether to include cells that have no value or formula (but might have formatting or comments). Useful when a whole column of cells has been formatted, but most are empty. Try setting this to FALSE if a spreadsheet seems too large to load.

Details

A cell has two 'values': its content, and sometimes also a formula. It also has formatting applied at the 'style' level, which can be locally overridden.

Content: Depending on the cell, the content may be a numeric value such as 365 or 365.25, it may represent a date/datetime in one of Excel's date/datetime systems, or it may be an index into an internal table of strings. `xlsx_cells()` attempts to infer the correct data type of each cell, returning its value in the appropriate column (error, logical, numeric, date, character). In case this cleverness is unhelpful, the unparsed value and type information is available in the 'content' and 'type' columns.

Formula: When a cell has a formula, the value in the 'content' column is the result of the formula the last time it was evaluated.

Certain groups of cells may share a formula that differs only by addresses referred to in the formula; such groups are identified by an index, the 'formula_group'. The xlsx (Excel) file format only records the formula against one cell in any group. `xlsx_cells()` propagates such formulas to the other cells in a group, making the necessary changes to relative addresses in the formula.

Array formulas may also apply to a group of cells, identified by an address 'formula_ref', but xlsx (Excel) file format only records the formula against one cell in the group. `xlsx_cells()` propagates such formulas to the other cells in a group. Unlike shared formulas, no changes to addresses in array formulas are necessary.

Formulas that refer to other workbooks currently do not name the workbooks directly, instead via indices such as [1]. It is planned to dereference these.

Formatting: Cell formatting is returned by `xlsx_formats()`. There are two types of formatting: 'style' formatting, such as Excel's built-in styles 'normal', 'bad', etc., and 'local' formatting, which overrides the style. These are returned in the $style and $local sublists of `xlsx_formats()`, with identical structures.

To look up the local formatting of a given cell, take the cell's `local_format_id` value (`my_cells$Sheet1[1,"local_forma`
and use it as an index into the format structure. E.g. to look up the font size, `my_formats$local$font$size[local_forma`
To see all available formats, type `str(my_formats$local)`.

Strings can be formatted within a cell, so that a single cell can contain substrings with different formatting. This in-cell formatting is available in the column `character_formatted`, which is a list-column of data frames. Each row of each data frame describes a substring and its formatting. For cells without a character value, `character_formatted` is `NULL`, so for further processing you might need to filter out the `NULL`s first.

Value

A data frame with the following columns.

- `sheet` The worksheet that the cell is from.
- `address` The cell address in A1 notation.
- `row` The row number of a cell address (integer).
- `col` The column number of a cell address (integer).
- `is_blank` Whether or not the cell has a value
- `data_type` The type of a cell, referring to the following columns: error, logical, numeric, date, character, blank.
- `error` The error value of a cell.
- `logical` The boolean value of a cell.
- `numeric` The numeric value of a cell.
- `date` The date value of a cell.
- `character` The string value of a cell.
- `formula` The formula in a cell (see 'Details').
- `is_array` Whether or not the formula is an array formula.
- `formula_ref` The address of a range of cells group to which an array formula or shared formula applies (see 'Details').
- `formula_group` The formula group to which the cell belongs (see 'Details').
- `comment` The text of a comment attached to a cell.
- `height` The height of a cell's row, in Excel's units.
- `width` The width of a cell's column, in Excel's units.
- `style_format` An index into a table of style formats `x$formats$style` (see 'Details').
- `local_format_id` An index into a table of local cell formats `x$formats$local` (see 'Details').

Cell formatting is returned in `xlsx_formats()`. There are two types or scopes of formatting: 'style' formatting, such as Excel's built-in styles 'normal', 'bad', etc., and 'local' formatting, which overrides particular elements of the style, e.g. by making it bold. Both types are returned, in the $style and $local sublists of `xlsx_formats()`, with identical structures. To look up the local formatting of a given cell, take the cell's 'local_format_id' value (`my_cells$data$Sheet1[1,"local_format_id"]`), and use it as an index into the format structure. E.g. to look up the font size, `my_formats$local$font$size[local_format_`
To see all available formats, type `str(my_formats$local)`.

Examples

```
examples <- system.file("extdata/examples.xlsx", package = "tidyxl")

# All sheets
str(xlsx_cells(examples))

# Specific sheet either by position or by name
str(xlsx_cells(examples, 2))
```

```
str(xlsx_cells(examples, "Sheet1"))

# The formats of particular cells can be retrieved like this:

Sheet1 <- xlsx_cells(examples, "Sheet1")
formats <- xlsx_formats(examples)

formats$local$font$bold[Sheet1$local_format_id]
formats$style$font$bold[Sheet1$style_format]

# To filter for cells of a particular format, first filter the formats to get
# the relevant indices, and then filter the cells by those indices.
bold_indices <- which(formats$local$font$bold)
Sheet1[Sheet1$local_format_id %in% bold_indices, ]

# In-cell formatting is available in the `character_formatted` column as a
# data frame, one row per substring.
xlsx_cells(examples)$character_formatted[77]
```

`xlsx_color_standard` *Names and RGB values of Excel standard colours*

Description

A dataset containing the names and RGB colour values of Excel's standard palette.

Usage

```
xlsx_color_standard

xlsx_colour_standard
```

Format

A data frame with 10 rows and 2 variables:

- `name` Name of the colour
- `rgb` RGB value of the colour

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 10 rows and 2 columns.

xlsx_color_theme *Import theme color definitions from xlsx (Excel) files*

Description

`xlsx_color_theme()` (alias `xlsx_colour_theme()` returns the names and RGB values of theme colours defined in xlsx (Excel) files. For example, `"accent6"` is the name of a theme colour in Excel, which could resolve to any RGB colour defined by the author of the file. Themes are often defined to comply with corporate standards.

Usage

```
xlsx_color_theme(path, check_filetype = TRUE)

xlsx_colour_theme(path, check_filetype = TRUE)
```

Arguments

`path`	Path to the xlsx file.
`check_filetype`	Logical. Whether to check that the filetype is xlsx (or xlsm) by looking at the file itself, rather than using the filename extension.

Value

A data frame, one row per colour, with the following columns.

- `name` The name of the theme.
- `rgb` The RGB colour that has been set for the theme in this file.

Examples

```
examples <- system.file("extdata/examples.xlsx", package = "tidyxl")
xlsx_color_theme(examples)
xlsx_colour_theme(examples)
```

xlsx_formats *Import xlsx (Excel) formatting definitions.*

Description

`xlsx_formats()` imports formatting definitions from spreadsheets. The structure is a nested list, e.g. `bold` is a vector within the list `font`, which is within the list `local`, which is within the list returned by `xlsx_formats()`. You can look up a cell's formatting by indexing the bottom-level vectors. See 'Details' for examples.

Usage

```
xlsx_formats(path, check_filetype = TRUE)
```

Arguments

path	Path to the xlsx file.
check_filetype	Logical. Whether to check that the filetype is xlsx (or xlsm) by looking at the file itself, rather than using the filename extension.

Details

There are two types of formatting: 'style' formatting, such as Excel's built-in styles 'normal', 'bad', etc., and 'local' formatting, which overrides the style. These are returned in the $style and $local sublists of `xlsx_formats()`, with identical structures.

To look up the local formatting of a given cell, take the cell's `local_format_id` value (`my_cells$Sheet1[1,"local_format`
and use it as an index into the format structure. E.g. to look up the font size, `my_formats$local$font$size[local_format_`
To see all available formats, type `str(my_formats$local)`.

Colours may be recorded in any of three ways: a hexadecimal RGB string with or without alpha, an 'indexed' colour, and an index into a 'theme'. `xlsx_formats()` dereferences 'indexed' and 'theme' colours to their hexadecimal RGB string representation, and standardises all RGB strings to have an alpha channel in the first two characters. The 'index' and the 'theme' name are still provided. To filter by an RGB string, you could look up the RGB values in a spreadsheet program (e.g. Excel, LibreOffice, Gnumeric), and use the `grDevices::rgb()` function to convert these to a hexadecimal string.

```
A <- 1; R <- 0.5; G <- 0; B <- 0
rgb(A, R, G, B)
# [1] "#FF800000"
```

Value

A nested list of vectors, beginning at the top level with $style and $local, then drilling down to the vectors that hold the definitions. E.g. `my_formats$local$font$size`.

Examples

```
examples <- system.file("extdata/examples.xlsx", package = "tidyxl")
str(xlsx_formats(examples))

# The formats of particular cells can be retrieved like this:

cells <- xlsx_cells(examples)
formats <- xlsx_formats(examples)

formats$local$font$bold[cells$local_format_id]
formats$style$font$bold[cells$style_format]

# To filter for cells of a particular format, first filter the formats to get
# the relevant indices, and then filter the cells by those indices.
```

```
bold_indices <- which(formats$local$font$bold)
cells[cells$local_format_id %in% bold_indices, ]
```

`xlsx_names` *Import named formulas from xlsx (Excel) files*

Description

`xlsx_names()` returns the names and definitions of named formulas (aka named ranges) in xlsx (Excel) files.

Most names refer to ranges of cells, but they can also be defined as formulas. `xlsx_names()` tells you whether or not they are a range, using `is_range()` to work this out.

Names are scoped either globally (used only once in the file), or locally to each sheet (can be reused with different definitions in different sheets). For sheet-scoped names, `xlsx_names()` provides the name of the sheet.

Usage

```
xlsx_names(path, check_filetype = TRUE)
```

Arguments

`path`	Path to the xlsx file.
`check_filetype`	Logical. Whether to check that the filetype is xlsx (or xlsm) by looking at the file itself, rather than using the filename extension.

Value

A data frame, one row per name, with the following columns.

- `sheet` If the name is defined only for a specific sheet, the name of the sheet. Otherwise `NA` for names defined globally.
- `name`
- `formula` Usually a range of cells, but sometimes a whole formula, e.g. `MAX(A2,1)`.
- `comment` A description given by the spreadsheet author.
- `hidden` Whether or not the name is visible to the user in spreadsheet applications. Hidden names are usually ones that were created automatically by the spreadsheet application.
- `is_range` Whether or not the `formula` is a range of cells. This is handy for joining to the set of cells referred to by a name. In this context, commas between cell addresses are always regarded as union operators – this differs from `xlex()`, see that help file for details.

Examples

```
examples <- system.file("extdata/examples.xlsx", package = "tidyxl")
xlsx_names(examples)
```

xlsx_sheet_names *List sheets in an xlsx (Excel) file*

Description

`xlsx_sheets()` returns the names of the sheets in a workbook, as a character vector. They are in the same order as they appear in the spreadsheet when it is opened with a spreadsheet application like Excel or LibreOffice.

Usage

```
xlsx_sheet_names(path, check_filetype = TRUE)
```

Arguments

`path`	Path to the xlsx file.
`check_filetype`	Logical. Whether to check that the filetype is xlsx (or xlsm) by looking at the file itself, rather than using the filename extension.

Value

A character vector of the names of the worksheets in the file.

Examples

```
examples <- system.file("extdata/examples.xlsx", package = "tidyxl")
xlsx_sheet_names(examples)
```

xlsx_validation *Import data validation rules of cells in xlsx (Excel) files*

Description

`xlsx_validation()` returns the data validation rules applied to cells in xlsx (Excel) files. Data validation rules control what constants can be entered into a cell, e.g. any whole number between 0 and 9, or one of several values from another part of the spreadsheet.

Usage

```
xlsx_validation(path, sheets = NA)
```

Arguments

`path`	Path to the xlsx file.
`sheets`	Sheets to read. Either a character vector (the names of the sheets), an integer vector (the positions of the sheets), or NA (default, all sheets).

Value

A data frame with the following columns.

- `sheet` The worksheet that a validation rule cell is from.
- `ref` Comma-delimited cell addresses to which the rules apply, e.g. `A106` or A115,A121:A122‘.
- type Data type of input, one of whole, decimal, list, date, time, textLength, custom, and whole‘.
- `operator` Unless `type` is `list` or `custom`, then `operator` is one of `between`, `notBetween`, `equal`, `notEqual`, `greaterThan`, `lessThan`, `greaterThanOrEqual`, `lessthanOrEqual`.
- `formula1` If `type` is `list`, then a range of cells whose values are allowed by the rule. If `type` is `custom`, then a formula to determine allowable values. Otherwise, a cell address or constant, coerced to character. Dates and times are formatted like "2017-01-27 13:30:45". Times without dates are formatted like "13:30:45".
- `formula2` If `operator` is `between` or `notBetween`, then a cell address or constant as with formula1, otherwise NA.
- `allow_blank` Boolean, whether or not the rule allows blanks.
- `show_input_message` Boolean, whether or not the rule shows a message when the user begins entering a value.
- `prompt_title` Text to appear in the title bar of a popup message box when the user begins entering a value.
- `prompt_body` Text to appear in a popup message box when the user begins entering a value. When `NA`, then some default text is shown.
- `show_error_message` Boolean, whether or not the rule shows a message when the user has entered a forbidden value. When `NA`, then some default text is shown.
- `error_title` Text to appear in the title bar of a popup message box when the user enters a forbidden value. When `NA`, then some default text is shown.
- `error_body` Text to appear in a popup message box when the user enters a forbidden value. When `NA`, then some default text is shown.
- `error_symbol` Name of a symbol to appear in the popup error message when the user enters a forbidden value.

Examples

```
examples <- system.file("extdata/examples.xlsx", package = "tidyxl")
xlsx_validation(examples)
xlsx_validation(examples, 1)
xlsx_validation(examples, "Sheet1")
```

Index