Package 'readxl'

December 20, 2018

Title Read Excel Files

Version 1.2.0

Description Import excel files into R. Supports '.xls' via the embedded 'libxls' C library <https://github.com/evanmiller/libxls> and '.xlsx' via the embedded 'RapidXML' C++ library <http://rapidxml.sourceforge.net>. Works on Windows, Mac and Linux without external dependencies.

License GPL-3

URL http://readxl.tidyverse.org, https://github.com/tidyverse/readxl

BugReports https://github.com/tidyverse/readxl/issues

Imports cellranger, progress, Rcpp (>= 0.12.18), tibble (>= 1.3.1), utils

Suggests covr, knitr, rmarkdown, rprojroot (>= 1.1), testthat

LinkingTo progress, Rcpp

VignetteBuilder knitr

Encoding UTF-8

LazyData true

Note libxls-SHA 53cd37b

RoxygenNote 6.1.1

NeedsCompilation yes

Author Hadley Wickham [aut] (<https://orcid.org/0000-0003-4757-117X>), Jennifer Bryan [aut, cre] (<https://orcid.org/0000-0002-6983-2759>), RStudio [cph, fnd] (Copyright holder of all R code and all C/C++ code without explicit copyright attribution), Marcin Kalicinski [ctb, cph] (Author of included RapidXML code), Komarov Valery [ctb, cph] (Author of included libxls code), Christophe Leitienne [ctb, cph] (Author of included libxls code), Bob Colbert [ctb, cph] (Author of included libxls code), David Hoerl [ctb, cph] (Author of included libxls code), Evan Miller [ctb, cph] (Author of included libxls code)

Maintainer Jennifer Bryan <jenny@rstudio.com>

Repository CRAN

Date/Publication 2018-12-19 23:50:03 UTC

R topics documented:

`cell-specification` *Specify cells for reading*

Description

The `range` argument of `read_excel()` provides many ways to limit the read to a specific rectangle of cells. The simplest usage is to provide an Excel-like cell range, such as `range = "D12:F15"` or `range = "R1C12:R6C15"`. The cell rectangle can be specified in various other ways, using helper functions. In all cases, cell range processing is handled by the cellranger package, where you can find full documentation for the functions used in the examples below.

See Also

The cellranger package has full documentation on cell specification and offers additional functions for manipulating "A1:D10" style spreadsheet ranges. Here are the most relevant:

- `cellranger::cell_limits()`
- `cellranger::cell_rows()`
- `cellranger::cell_cols()`
- `cellranger::anchored()`

Examples

```
path <- readxl_example("geometry.xls")
## Rows 1 and 2 are empty (as are rows 7 and higher)
## Column 1 aka "A" is empty (as are columns 5 of "E" and higher)

# By default, the populated data cells are "shrink-wrapped" into a
# minimal data frame
read_excel(path)

# Specific rectangle that is subset of populated cells, possibly improper
```

```
read_excel(path, range = "B3:D6")
read_excel(path, range = "C3:D5")

# Specific rectangle that forces inclusion of unpopulated cells
read_excel(path, range = "A3:D5")
read_excel(path, range = "A4:E5")
read_excel(path, range = "C5:E7")

# Anchor a rectangle of specified size at a particular cell
read_excel(path, range = anchored("C4", dim = c(3, 2)), col_names = FALSE)

# Specify only the rows or only the columns
read_excel(path, range = cell_rows(3:6))
read_excel(path, range = cell_cols("C:D"))
read_excel(path, range = cell_cols(2))

# Specify exactly one row or column bound
read_excel(path, range = cell_rows(c(5, NA)))
read_excel(path, range = cell_rows(c(NA, 4)))
read_excel(path, range = cell_cols(c("C", NA)))
read_excel(path, range = cell_cols(c(NA, 2)))

# General open rectangles
# upper left = C4, everything else unspecified
read_excel(path, range = cell_limits(c(4, 3), c(NA, NA)))
# upper right = D4, everything else unspecified
read_excel(path, range = cell_limits(c(4, NA), c(NA, 4)))
```

`excel_format` *Determine file format*

Description

Determine if files are xls or xlsx (or from the xlsx family).

`excel_format(guess = TRUE)` is used by `read_excel()` to determine format. It draws on logic from two lower level functions:

- `format_from_ext()` attempts to determine format from the file extension.
- `format_from_signature()` consults the file signature or "magic number".

File extensions associated with xlsx vs. xls:

- xlsx: `.xlsx`, `.xlsm`, `.xltx`, `.xltm`
- xls: `.xls`

File signatures (in hexadecimal) for xlsx vs xls:

- xlsx: First 4 bytes are `50 4B 03 04`
- xls: First 8 bytes are `D0 CF 11 E0 A1 B1 1A E1`

Usage

```
excel_format(path, guess = TRUE)

format_from_ext(path)

format_from_signature(path)
```

Arguments

path	Path to the xls/xlsx file.
guess	Logical. If the file extension is absent or not recognized, this controls whether we attempt to guess format based on the file signature or "magic number".

Value

Character vector with values "xlsx", "xls", or NA.

Examples

```
files <- c(
  "a.xlsx",
  "b.xls",
  "c.png",
  file.path(R.home("doc"), "html", "logo.jpg"),
  readxl_example("clippy.xlsx"),
  readxl_example("deaths.xls")
)
excel_format(files)
```

---

excel_sheets *List all sheets in an excel spreadsheet*

Description

List all sheets in an excel spreadsheet

Usage

```
excel_sheets(path)
```

Arguments

path	Path to the xls/xlsx file.

Examples

```
excel_sheets(readxl_example("datasets.xlsx"))
excel_sheets(readxl_example("datasets.xls"))

# To load all sheets in a workbook, use lapply
path <- readxl_example("datasets.xls")
lapply(excel_sheets(path), read_excel, path = path)
```

---

readxl_example *Get path to readxl example*

Description

readxl comes bundled with some example files in its `inst/extdata` directory. This function make them easy to access.

Usage

```
readxl_example(path = NULL)
```

Arguments

`path`	Name of file. If `NULL`, the example files will be listed.

Examples

```
readxl_example()
readxl_example("datasets.xlsx")
```

---

readxl_progress *Determine whether to show progress spinner*

Description

By default, readxl displays a progress spinner **unless** one of the following is `TRUE`:

- The spinner is explicitly disabled by setting `options(readxl.show_progress = FALSE)`.
- The code is run in a non-interactive session (`interactive()` is `FALSE`).
- The code is run by knitr / rmarkdown.
- The code is run in an RStudio notebook chunk. readxl uses the progress package under-the-hood and therefore is also sensitive to any options that it consults.

Usage

```
readxl_progress()
```

read_excel *Read xls and xlsx files*

Description

Read xls and xlsx files

`read_excel()` calls `excel_format()` to determine if `path` is xls or xlsx, based on the file extension and the file itself, in that order. Use `read_xls()` and `read_xlsx()` directly if you know better and want to prevent such guessing.

Usage

```
read_excel(path, sheet = NULL, range = NULL, col_names = TRUE,
  col_types = NULL, na = "", trim_ws = TRUE, skip = 0,
  n_max = Inf, guess_max = min(1000, n_max),
  progress = readxl_progress(), .name_repair = "unique")

read_xls(path, sheet = NULL, range = NULL, col_names = TRUE,
  col_types = NULL, na = "", trim_ws = TRUE, skip = 0,
  n_max = Inf, guess_max = min(1000, n_max),
  progress = readxl_progress(), .name_repair = "unique")

read_xlsx(path, sheet = NULL, range = NULL, col_names = TRUE,
  col_types = NULL, na = "", trim_ws = TRUE, skip = 0,
  n_max = Inf, guess_max = min(1000, n_max),
  progress = readxl_progress(), .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.
`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. By default, readxl ensures column names are not empty and are unique. If the tibble package version is recent enough, there is full support for `.name_repair` as documented in `tibble::tibble()`. If an older version of tibble is present, readxl falls back to name repair in the style of tibble v1.4.2.

Value

A tibble

See Also

cell-specification for more details on targetting cells with the `range` argument

Examples

```
datasets <- readxl_example("datasets.xlsx")
read_excel(datasets)

# Specify sheet either by position or by name
read_excel(datasets, 2)
read_excel(datasets, "mtcars")

# Skip rows and use default column names
read_excel(datasets, skip = 148, col_names = FALSE)

# Recycle a single column type
read_excel(datasets, col_types = "text")

# Specify some col_types and guess others
```

```
read_excel(datasets, col_types = c("text", "guess", "numeric", "guess", "guess"))

# Accomodate a column with disparate types via col_type = "list"
df <- read_excel(readxl_example("clippy.xlsx"), col_types = c("text", "list"))
df
df$value
sapply(df$value, class)

# Limit the number of data rows read
read_excel(datasets, n_max = 3)

# Read from an Excel range using A1 or R1C1 notation
read_excel(datasets, range = "C1:E7")
read_excel(datasets, range = "R1C2:R2C5")

# Specify the sheet as part of the range
read_excel(datasets, range = "mtcars!B1:D5")

# Read only specific rows or columns
read_excel(datasets, range = cell_rows(102:151), col_names = FALSE)
read_excel(datasets, range = cell_cols("B:D"))

# Get a preview of column names
names(read_excel(readxl_example("datasets.xlsx"), n_max = 0))

if (utils::packageVersion("tibble") > "1.4.2") {
  ## exploit full .name_repair flexibility from tibble

  ## "universal" names are unique and syntactic
  read_excel(
    readxl_example("deaths.xlsx"),
    range = "arts!A5:F15",
    .name_repair = "universal"
  )

  ## specify name repair as a built-in function
  read_excel(readxl_example("clippy.xlsx"), .name_repair = toupper)

  ## specify name repair as a custom function
  my_custom_name_repair <- function(nms) tolower(gsub("[.]", "_", nms))
  read_excel(
    readxl_example("datasets.xlsx"),
    .name_repair = my_custom_name_repair
  )

  ## specify name repair as an anonymous function
  read_excel(
    readxl_example("datasets.xlsx"),
    sheet = "chickwts",
    .name_repair = ~ substr(.x, start = 1, stop = 3)
  )
}
```

Index