Package ‘gtxlsx’

November 7, 2026

Type Package

Title Write 'gt' and HTML Tables into 'openxlsx2' Workbooks

Version 0.5.0

Description Turns a 'gt' table into a range of cells in an 'openxlsx2' workbook, keeping the heading, column spanners, row groups, stub, summary rows, footnotes and the styling set through 'gt'. Numbers stay numbers wherever a spreadsheet number format can reproduce what 'gt' shows. A second entry point does the same for a plain HTML table, so output from other table packages can be written to a worksheet as well; that path needs nothing beyond 'openxlsx2'.

License MIT + file LICENSE

URL https://janmarvin.github.io/gtxlsx/, https://github.com/JanMarvin/gtxlsx

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

Encoding UTF-8

Language en-US

Depends R (>= 3.6.0)

Imports grDevices, openxlsx2 (>= 1.0), utils

Suggests gt (>= 0.8.0), lt (>= 0.2), markdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Jan Marvin Garbuszus [aut, cre]

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

Repository CRAN

Date/Publication 2026-11-07 19:50:07 UTC

Contents

gtxlsx_extract 2
wb_add_gt 3
wb_add_html 5
wb_add_lt 8
wb_to_gt 9

Index **11**

`gtxlsx_extract` *Look at the pieces of a built gt table*

Description

Runs gt's own build step and hands back the result as plain data frames and lists: the rendered body, the column definitions, the stub, the row groups, the spanners, the styles, the footnotes and the table options.

Usage

```
gtxlsx_extract(x, context = "html")
```

Arguments

`x`	A `gt_tbl` object.
`context`	Render context handed to gt's builder. `"html"` is what `wb_add_gt()` uses and the only value that has been exercised here.

Details

This is the input `wb_add_gt()` works from. It is exported mainly so you can see why a table came out the way it did, or check what a `gt` feature leaves behind before it reaches the worksheet.

Value

A named list with the elements `body`, `data`, `boxhead`, `stub`, `groups_rows`, `row_groups`, `spanners`, `heading`, `stubhead`, `styles`, `footnotes`, `source_notes`, `summary` and `options`.

Examples

```
library(gt)

tbl <- gt(data.frame(a = 1:2, b = c(1.5, 2.5)))
tbl <- fmt_number(tbl, columns = "b", decimals = 1)

g <- gtxlsx_extract(tbl)
g$body
g$boxhead$var
```

wb_add_gt *Write a gt table into a worksheet*

Description

Lays a gt table out as cells: the heading, the column spanners, the column labels, the stub, the row groups, the body, any summary rows, the footnotes and the source notes, one after another in a single rectangular block starting at dims.

Usage

```
wb_add_gt(
  wb,
  x,
  sheet = current_sheet(),
  dims = "A1",
  numeric = TRUE,
  convert_date = TRUE,
  col_widths = "auto",
  row_heights = NULL,
  ignore_errors = TRUE,
  gap = 1L,
  features = TRUE,
  freeze = FALSE,
  ...
)
```

Arguments

wb	A wbWorkbook object, as returned by openxlsx2::wb_workbook().
x	A gt_tbl object, or a gt_group as returned by gt::gt_group() or gt::gt_split(). A group is written one table after another down the sheet.
sheet	The worksheet to write to. Defaults to the current sheet.
dims	Cell reference of the top left corner of the table, for example "B2".
numeric	Write numbers as numbers where the displayed format can be reproduced. Set to FALSE to write every number as text.
convert_date	Write dates and times as spreadsheet dates. A column is converted when its text is exactly what one of gt's date or time styles gives for the stored value, with the matching spreadsheet format; month and weekday names then follow the language of the spreadsheet software. With FALSE, or where no style reproduces the text, the printed text is written as it is.
col_widths	"auto" measures the rendered text and sizes the columns to fit it, a numeric vector sets the widths directly, and NULL leaves them alone. Widths set with gt::cols_width() always win.

row_heights	NULL, the default, leaves the spreadsheet software to size the rows. "gt" sets each row from the padding gt would have used, and a numeric vector sets the heights directly. Both also center the text vertically, since spreadsheet software aligns to the bottom of a cell and gt pads evenly. Rows with wrapped text keep the software's own sizing, which a fixed height would clip. Footnotes and source notes are merged across the table, and spreadsheet software does not fit the height of merged cells, so their height is estimated from the table width.
ignore_errors	Mark text cells whose content looks like a number or a date, so spreadsheet software stops flagging them.
gap	Blank rows left between the tables of a gt_group. Ignored for a single table.
features	What to write besides the values. TRUE, the default, is all of them; FALSE writes values only. Otherwise a character vector of any of "font", "fill", "border", "numfmt", "merge" and "link", so a table that goes wrong in one respect can still be written in every other.
freeze	Freeze panes so the heading and the stub stay in view while scrolling. TRUE freezes below the heading and beside the stub, a length-two vector c(row, col) freezes at a cell of your choosing, and FALSE, the default, leaves the sheet alone.
...	Currently unused.

Details

Everything gt applies before rendering is already in place when the cells are written, because gtxlsx reads the table gt has built rather than repeating the work: every fmt_*() and sub_*(), the cols_merge_*() family, text_transform(), data_color(), summary_rows() and the footnote marks. Styling set with gt::tab_style() and gt::tab_options() becomes fonts, fills, alignment and borders; markup inside a cell (bold, italic, superscripts, line breaks) becomes rich text.

Anything gt draws as a picture cannot be written to a cell. gt::fmt_image() and gt::cols_nanoplot() leave the cell empty, gt::fmt_icon() and gt::fmt_flag() fall back to their label text.

Value

The workbook, invisibly. The input workbook is not modified; a clone is returned, as elsewhere in openxlsx2.

Row striping

A striped table gets a fill on every body row, the striping color on one and table.background.color on the next. Spreadsheet software leaves an unfilled cell transparent, so filling only half the rows would show the banding as detached blocks rather than a continuous column.

The color comes from gt, and gt's default is white. On a worksheet with a colored background that white will cover the tint under the table. Set table.background.color to match, or turn striping off, if that matters.

Links

fmt_url() and fmt_email() leave an anchor in the cell, and that becomes a hyperlink on the cell. The text shown is whatever gt put there.

Numbers versus text

With `numeric = TRUE` a column is written as numbers whenever a spreadsheet number format can reproduce exactly what gt displays. `$1,234.50` becomes the value `1234.5` with the format `"$"#,##0.00`, so the sheet stays usable for arithmetic. Columns gt has scaled or suffixed (`1.2K` for `1200`) cannot be reproduced that way and stay text; those cells are marked so spreadsheet software does not flag them as numbers stored as text. The decision is made per column: if one cell cannot be reproduced, for example a cell with a footnote mark or one replaced with a dash by `text_transform()`, the whole column is written as text. Missing values shown with `sub_missing()` are the exception and leave the numbers alone.

See Also

`wb_add_html()` for tables that are already HTML, and `gtxlsx_extract()` to see the pieces `wb_add_gt()` works from.

Examples

```
library(gt)
library(openxlsx2)

tbl <- gt(data.frame(item = c("Cash", "Debt"), amount = c(1204.5, -3910)))
tbl <- fmt_currency(tbl, columns = "amount", decimals = 2)
tbl <- tab_header(tbl, title = "Balance")

wb <- wb_workbook()$add_worksheet()
wb <- wb_add_gt(wb, tbl, dims = "B2")

wb_to_df(wb, col_names = FALSE)
```

---

wb_add_html *Write an HTML table into a worksheet*

Description

Reads the first (or any) `<table>` of an HTML fragment or document and writes it as cells. `colspan` and `rowspan` become merged ranges, `<style>` rules and `style=` attributes become fills, fonts, alignment and borders, and markup inside a cell becomes rich text. Titles and notes that sit beside the table rather than inside it are picked up as well.

Usage

```
wb_add_html(
  wb,
  x,
  sheet = current_sheet(),
  dims = "A1",
  which = 1L,
```

```
  numeric = TRUE,
  convert_date = TRUE,
  dec_mark = NULL,
  big_mark = NULL,
  col_widths = "auto",
  ignore_errors = TRUE,
  context = TRUE,
  features = TRUE,
  freeze = FALSE,
  ...
)
```

Arguments

wb	A wbWorkbook object.
x	HTML: a string, a file path, an already parsed document, or anything with an as.character() method that returns HTML. That includes what rvest and xml2 hand back, so a scraped page or a single <table> node can be passed straight in.
sheet	The worksheet to write to. Defaults to the current sheet.
dims	Cell reference of the top left corner.
which	Which table in the document to write, when there is more than one.
numeric	Write cells as numbers where the text is plainly a number. Only symbol prefixes and suffixes such as $ or % are converted, so a label like "458 Speciale" stays text, and so does a number that the text would not give back, such as 007.
convert_date	Write cells as spreadsheet dates when the text is a whole day in one of gt's date styles or in ISO form, such as 31 January 2026 or 2026-06-05, with or without a time, and the matching format gives the text back exactly. With FALSE the text is written as it is.
dec_mark, big_mark	The characters the table uses for the decimal point and for grouping thousands, for example dec_mark = "," for a table written as 1.234,50. With both left at NULL they are worked out from the whole table: 1,234,567, 1.234,50 or 12 345 show which marks it uses, and a table that uses two different ones is not converted at all, with a warning. A number that could still be read two ways stays text: 1.234 is not guessed to be either 1.234 or 1234. big_mark = "" is a table without grouping.
col_widths	"auto" measures the rendered text, a numeric vector sets the widths directly, NULL leaves them alone.
ignore_errors	Mark text cells that look numeric, so spreadsheet software does not flag them.
context	Pick up block elements sitting beside the table, such as a heading above it or a note below, and write them as merged rows. Set to FALSE to write the table on its own.
features	What to write besides the values. TRUE, the default, is all of them; FALSE writes values only. Otherwise a character vector of any of "font", "fill", "border", "numfmt", "merge" and "link". A page whose CSS or links go wrong in one respect can still be written in every other.

freeze	Freeze panes so the header rows and any leading <th> column stay in view while scrolling. TRUE works them out from the table, a length-two vector c(row, col) freezes at a cell of your choosing, and FALSE, the default, leaves the sheet alone.
...	Currently unused.

Details

This is the general path for tables that are already HTML, whatever produced them. openxlsx2 is the only thing it needs; gt is a suggestion and nothing on this path uses it.

Old fashioned presentational markup is understood too: bgcolor, align, valign, width, nowrap and <table border>.

Value

The workbook, invisibly.

How much CSS is understood

Enough for tables, not enough to call it a browser. A selector is matched by walking its components against the cell and the elements above it, so table.report td.total, thead td and div > td all mean what they say. Nested rules, :is(), custom properties and !important are handled, as are the positional pseudo-classes :first-child, :last-child, :only-child and :nth-child().

An <a href> inside a cell becomes a hyperlink on that cell, and the whole cell is what becomes clickable: a spreadsheet has no way to link part of a cell's text. The first usable anchor is taken, since a cell holds one target, and a link that only points at a fragment of the source page is skipped. Either of those produces a warning naming how many were dropped.

What is not: sibling combinators (+, ~), state pseudo-classes and ::before cause a rule to be skipped rather than guessed at, attribute selectors match on the tag alone, @media conditions are ignored, and stylesheets pulled in with <link> are not fetched.

Properties with no spreadsheet equivalent, such as gradients, letter spacing and rounded corners, are dropped. <img> and <svg> leave an empty cell.

See Also

wb_add_gt(), which goes straight from a gt object and keeps more of the structure.

Examples

```
library(openxlsx2)

html <- paste0(
  "<style>th { background-color: #204060; color: white; }</style>",
  "<table><tr><th>Account</th><th>Change</th></tr>",
  "<tr><td>Cash</td><td>1,204.50</td></tr></table>"
)

wb <- wb_workbook()$add_worksheet()
wb <- wb_add_html(wb, html, dims = "A1")
```

```
wb_to_df(wb, col_names = FALSE)
```

---

wb_add_lt *Write an lt table into a worksheet*

Description

The `lt` package builds its HTML in JavaScript when the page is viewed, so there is no table to read on the R side. This helper asks `lt` to bake the table to static HTML first and then hands the result to `wb_add_html()`.

Usage

```
wb_add_lt(wb, x, sheet = current_sheet(), dims = "A1", method = "auto", ...)
```

Arguments

`wb`	A `wbWorkbook` object.
`x`	An `lt_tbl` object.
`sheet`	The worksheet to write to.
`dims`	Cell reference of the top left corner.
`method`	How `lt` should bake the table: `"node"` or `"browser"` to force a renderer, `"auto"` to use whichever is available.
`...`	Passed on to `wb_add_html()`.

Details

Baking needs Node.js or a Chromium based browser on the machine.

Value

The workbook, invisibly.

Examples

```
# needs the lt package and a Node.js or browser install
## Not run:
library(openxlsx2)

tbl <- lt::lt(data.frame(a = c("x", "y"), n = c(1234.5, 67.89)))
tbl <- lt::lt_format(tbl, ~ n, decimals = 2, big_mark = ",")

wb <- wb_workbook()$add_worksheet()
wb <- wb_add_lt(wb, tbl, dims = "B2")

## End(Not run)
```

wb_to_gt *Turn a worksheet range back into a gt table (experimental)*

Description

Reads a range of cells and builds a `gt` object from it: full width merged rows at the top become the heading, partly merged rows above the labels become spanners, full width merged rows at the bottom become source notes, and per-cell fills, fonts and alignment are translated into `gt::tab_style()` calls.

Usage

```
wb_to_gt(
  wb,
  sheet = current_sheet(),
  dims = NULL,
  styles = TRUE,
  structure = TRUE,
  ...
)
```

Arguments

`wb`	A `wbWorkbook` object.
`sheet`	The worksheet to read.
`dims`	Range to read. Defaults to the used range of the sheet.
`styles`	Translate cell styles into `gt::tab_style()` calls. This is done cell by cell, so it is slow on large ranges.
`structure`	Read merged cells as heading, spanners and source notes. With `FALSE` the range is taken as a plain table.
`...`	Passed on to `openxlsx2::wb_to_df()`.

Value

A `gt_tbl` object.

Please read this before using it

This function is a development toy, not a finished feature. It exists because the reverse direction was interesting to try, and it has had only light testing: a handful of sheets, no round trip guarantees. Treat its output as a starting point you will edit, not as a faithful copy, and expect the details to change or the function to be withdrawn.

A worksheet simply does not record most of what a `gt` table knows. Row groups, the stub, footnote marks and number formats do not come back: groups arrive as ordinary rows, the stub as a column named after its letter, footnote marks glued to the text they mark, and `$115,900` as the bare number `115900` with no `fmt_currency()` behind it. Column names are made unique, so repeated labels gain a suffix.

Examples

```
library(openxlsx2)

wb <- wb_workbook()$add_worksheet()
wb$add_data(x = data.frame(a = c("x", "y"), b = c(1, 2)))

tbl <- wb_to_gt(wb, dims = "A1:B3", styles = FALSE)
class(tbl)
```

Index

gtxlsx_extract, 2
gtxlsx_extract(), *5*

openxlsx2::wb_to_df(), *9*
openxlsx2::wb_workbook(), *3*

wb_add_gt, 3
wb_add_gt(), *2*, *7*
wb_add_html, 5
wb_add_html(), *5*, *8*
wb_add_lt, 8
wb_to_gt, 9