

Package ‘lt’

October 17, 2026

Type Package

Title Lightweight Tables via JSON Specs and JavaScript

Version 0.5

Description A lightweight grammar of tables. Build a table by declaring a JSON spec (titles, spanners, row groups, footnotes, formatting functions, etc.); a tiny vanilla JavaScript runtime builds the HTML table from the spec on page load. Cells can hold inline plots (error bars, dot plots, mini line and bar charts) drawn as SVG in the browser from the numbers alone. Tables can optionally be made interactive (sorting, searching, filtering, pagination, column hiding, expandable row detail, and CSV download) via a separate extension loaded only when requested. No 'sass', no 'V8', no 'htmlwidgets' — just base R and 'xfun' ('htmltools' is used only for the optional Shiny binding and for embedding tables in other HTML output).

License MIT + file LICENSE

URL <https://github.com/yihui/lt>

BugReports <https://github.com/yihui/lt/issues>

Depends R (>= 4.1.0)

Encoding UTF-8

Imports grDevices, xfun (>= 0.62)

Suggests chromote, htmltools, knitr, litedown, magick, repr, shiny, testit

Config/lt.js 1.14.110

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Yihui Xie [aut, cre, cph] (ORCID:
<<https://orcid.org/0000-0003-0645-5666>>, URL: <https://yihui.org>)

Maintainer Yihui Xie <xie@yihui.name>

Repository CRAN

Date/Publication 2026-10-17 03:30:02 UTC

Contents

format.lt_tbl	2
lt	3
lt_align	4
lt_css	5
lt_date	6
lt_dependency	7
lt_dotplot	8
lt_errorbar	10
lt_export	12
lt_footnote	14
lt_format	15
lt_group	16
lt_header	17
lt_hide	18
lt_html	18
lt_indent	19
lt_interactive	20
lt_label	22
lt_merge	23
lt_move	24
lt_note	25
lt_output	25
lt_spanner	26
lt_sparkline	27
lt_spec	28
lt_style	29
lt_sub	30
lt_width	31
lt_wrap	32
print.lt_tbl	32
Index	34

format.lt_tbl	<i>Render an lt_tbl to HTML</i>
---------------	---------------------------------

Description

Emits the CSS+JS runtime and a script block carrying the table’s JSON spec. Multiple tables on the same page only need the runtime once.

Usage

```
## S3 method for class 'lt_tbl'  
format(x, fragment = TRUE, inline_assets = TRUE, assets = TRUE, ...)
```

Arguments

<code>x</code>	An <code>lt_tbl</code> object.
<code>fragment</code>	If TRUE (default), return an HTML fragment suitable for embedding. If FALSE, wrap in a minimal <code><html><body></code> document.
<code>inline_assets</code>	If TRUE (default), inline the CSS/JS as text. If FALSE, emit <code><link>/<script src=...></code> tags (assets must be served alongside the HTML).
<code>assets</code>	Which runtime assets to include: TRUE (default) for both CSS and JS, FALSE for neither, or a character vector subset of <code>c("css", "js")</code> for selective inclusion. A table with <code>lt_interactive()</code> enabled also gets the interactivity extension: its stylesheet with "css", its script with "js".
<code>...</code>	Reserved for future use.

Value

A character scalar containing HTML.

Examples

```
tbl = lt(head(mtcars))
html = format(tbl)
format(tbl, fragment = FALSE, inline_assets = FALSE)
```

<code>lt</code>	<i>Create a table specification</i>
-----------------	-------------------------------------

Description

Entry point of the lightweight grammar of tables. Returns an object (a list) that records the data plus a list of table-modifying operations. The object is rendered to HTML by `format()` (called automatically by the print method).

Usage

```
lt(data, ...)

## Default S3 method:
lt(data, auto_format = TRUE, auto_label = TRUE, ...)
```

Arguments

<code>data</code>	A data frame (or anything coercible to one).
<code>...</code>	Arguments passed to methods.
<code>auto_format</code>	Whether to automatically format numeric columns (rounding, thousand separators, percentage detection). Set to FALSE to disable for the whole table; use <code>lt_format()</code> on specific columns to disable selectively.
<code>auto_label</code>	Whether to automatically clean column names for display by replacing separators (<code>.</code> and <code>_</code>) with spaces. Set to FALSE to show raw column names.

Value

A table object that can be piped into `lt_*`() functions.

HTML in cells

By default all cell values and text (titles, labels, footnotes, ...) are HTML-escaped, so special characters like `<`, `>`, and `&` render as literal text. To emit content as raw HTML instead:

- For body cells, mark whole columns with `lt_html()`.
- For a title, subtitle, column label, spanner, footnote, or note, wrap the text in `I()`. For example, `lt_header(x, I("<b>Report</b>"))` renders the title as raw HTML, while `lt_header(x, "<b>Report</b>")` shows the literal `<b>` tags.

Interactivity

When a cell value has been formatted (e.g., by auto-formatting or `lt_format()`), the original value is stored in the cell's title attribute and shown as a tooltip on hover. Additionally:

- Alt + Click on a table toggles display of all raw values in that table.
- Alt + Double-Click on a table toggles raw values globally (all tables on the page).

Raw values are shown as (value) after the formatted text, highlighted with a light yellow background.

Missing values

By default, missing (NA) cells display an em dash (–) so they are not confused with empty strings. Change the table-wide default with the `lt.missing` global option, e.g. `options(lt.missing = "n/a")`; set it to the empty string (`options(lt.missing = "")`) to leave NA cells blank. Use `lt_sub()` with `missing =` to override the text for specific columns.

Examples

```
lt(head(mtcars[, 1:4]))
```

lt_align	Set column alignment
----------	----------------------

Description

Override the auto-detected alignment for specific columns. By default, numeric columns are right-aligned and character columns are left-aligned.

Usage

```
lt_align(x, columns, align = c("left", "center", "right"))
```

Arguments

x	An <code>lt()</code> object.
columns	Columns to select: a character vector of column names, integer positions, or a one-sided formula. A formula whose right-hand side references the pronoun <code>.</code> (the character vector of column names) is evaluated as a predicate, e.g. <code>~endsWith(., "_time")</code> selects all columns whose names end with <code>_time</code> ; otherwise the bare names on the right-hand side are used, e.g. <code>~ x + y</code> .
align	One of "left", "center", or "right".

Value

x with the alignment recorded.

Examples

```
lt(head(mtcars)) |> lt_align(~ cyl + gear, "center")
```

lt_css

Attach custom CSS

Description

Add user-supplied stylesheets or inline rules that render after the built-in CSS, so rules can override the defaults.

Usage

```
lt_css(x, ...)
```

Arguments

x	An <code>lt()</code> object.
...	Unnamed arguments are stylesheet paths or URLs. A bare filename (no directory component) that does not exist in the working directory is resolved against the stylesheets shipped with lt, so e.g. <code>lt_css(x, "lt-gt.css")</code> uses the bundled gt-like theme. Named arguments define inline CSS rules scoped to <code>.lt-table</code>. Names are selectors (e.g., <code>.na</code>, <code>td.highlight</code>) and values are either a CSS string or a named list of properties: <code>lt_css(x, .na = "background: #eee")</code> or <code>lt_css(x, .na = list(background = "#eee"))</code>. The selectors and rules can also be given as a single list(selectors, rules) of two parallel vectors (see <code>lt_label()</code>).

Value

x with the stylesheets recorded.

Examples

```
tbl = lt(head(mtcars))
tbl |>
  lt_style("mpg", test = "v => v > 20", class = "high") |>
  lt_css(.high = list(background = "#cfc", fontWeight = "bold"))
```

lt_date	<i>Format date/time columns</i>
---------	---------------------------------

Description

Format date or datetime columns using JavaScript’s native Date methods. The underlying data should be Date or POSIXt in R (serialized as new Date(...) for the browser).

Usage

```
lt_date(x, columns, method = NULL, locale = NULL, options = NULL)
```

Arguments

x	An <code>lt()</code> object.
columns	Columns to select: a character vector of column names, integer positions, or a one-sided formula. A formula whose right-hand side references the pronoun <code>.</code> (the character vector of column names) is evaluated as a predicate, e.g. <code>~endsWith(., "_time")</code> selects all columns whose names end with <code>_time</code> ; otherwise the bare names on the right-hand side are used, e.g. <code>~ x + y</code> .
method	A JS Date method name to call (e.g., <code>"toLocaleDateString"</code> , <code>"toISOString"</code> , <code>"toDateString"</code> , <code>"toLocaleString"</code> , <code>"toLocaleTimeString"</code>). If <code>NULL</code> (the default), <code>toLocaleDateString()</code> is used with <code>locale</code> and <code>options</code> . See https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date#instance_methods for all available methods.
locale	A BCP 47 locale string (e.g., <code>"en-US"</code> , <code>"de-DE"</code> , <code>"ja-JP"</code>). Only used when <code>method</code> is <code>NULL</code> . See https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Intl#locales_argument for details.
options	A named list of <code>Intl.DateTimeFormat</code> options. Common fields include <code>year</code> (<code>"numeric"</code> , <code>"2-digit"</code>), <code>month</code> (<code>"numeric"</code> , <code>"2-digit"</code> , <code>"long"</code> , <code>"short"</code> , <code>"narrow"</code>), <code>day</code> (<code>"numeric"</code> , <code>"2-digit"</code>), <code>hour</code> , <code>minute</code> , <code>second</code> (<code>"numeric"</code> , <code>"2-digit"</code>), <code>weekday</code> (<code>"long"</code> , <code>"short"</code> , <code>"narrow"</code>), and <code>timeZoneName</code> (<code>"long"</code> , <code>"short"</code>). Only used when <code>method</code> is <code>NULL</code> .

Value

`x` with the date formatting recorded.

Note

The formatted date may differ from the input date depending on the viewer's local timezone. JavaScript's new `Date("2024-01-15")` parses date-only strings as UTC midnight, but `toLocaleDateString()` converts to the local timezone. For example, 2024-01-15 will display as 2024-01-14 for a viewer at GMT-6. To avoid this, pass `options = list(timeZone = "UTC")`.

Examples

```
d = data.frame(
  event = c("Launch", "Update"),
  date = as.Date(c("2024-01-15", "2024-06-30"))
)
lt(d) |> lt_date(~ date)
lt(d) |> lt_date(~ date, options = list(year = "numeric", month = "short"))
```

lt_dependency

*HTML dependency for lt tables***Description**

The `htmltools::htmlDependency()` bundling lt's runtime assets, for embedding lt tables in other HTML output (e.g. another `htmlwidget` or a `reactable` cell) and rendering them client-side with `LT.render()` / `LT.buildHtml()` on a spec from `lt_spec()`. Requires the **htmltools** package.

Usage

```
lt_dependency(interactive = FALSE, plot = FALSE, shiny = FALSE)
```

Arguments

<code>interactive</code>	Whether to also include the interactivity extension (the assets behind <code>lt_interactive()</code>): its script and stylesheet.
<code>plot</code>	Whether to also include the graphics module (the assets behind <code>lt_errorbar()</code> and other inline-plot cells): its script and stylesheet. Its script loads before the core runtime so plot cells render.
<code>shiny</code>	Whether to include the Shiny output binding (only needed by <code>lt_output()</code> / <code>render_lt()</code>).

Value

An `html_dependency` object.

Examples

```
if (requireNamespace('htmltools', quietly = TRUE))
  lt_dependency(interactive = TRUE)
```

lt_dotplot

*Draw an inline dot plot in a column***Description**

Render each row's values across one or more columns as inline dots on a shared horizontal scale: N columns give N dots per row, laid out by value. It is lightweight like `lt_errorbar()` and `lt_sparkline()` – only the numbers travel to the client and the SVG is drawn in the browser, so an interactive table (`lt_interactive()`) draws only the rows on the current page.

Usage

```
lt_dotplot(
  x,
  columns,
  limits = NULL,
  color = FALSE,
  labels = NULL,
  stagger = FALSE,
  into = NULL,
  width = 160,
  height = NULL,
  hide = TRUE,
  axis = TRUE
)
```

Arguments

x	An <code>lt()</code> object.
columns	The value column(s): one dot is drawn per column, read left to right across the row, all on a scale shared across the columns. The plot is drawn in the first column's cells, or into <code>into</code> when given. A one-sided formula (<code>~ a + b + c</code>), names, or integer positions.
limits	Numeric <code>c(min, max)</code> for the shared horizontal scale. Defaults to the range of all the columns' finite values. Values outside the scale are clamped to the edges.
color	Series colors. <code>FALSE</code> (default) draws every series in one default color; <code>TRUE</code> assigns colors from the current palette (<code>grDevices::palette()</code>); a character vector of CSS colors sets them explicitly (recycled to the number of series). When colored, a legend is drawn below the plot in the footer.
labels	Legend labels, one per series, used when colored. Defaults to the series' value-column names.
stagger	If <code>TRUE</code> , spread the dots onto separate vertical tracks (one per column) instead of a single baseline, so near-equal values do not overlap. The vertical position is only cosmetic separation; the value is still read horizontally. The cell is made taller to fit the tracks (unless <code>height</code> is set). <code>FALSE</code> (default) draws all dots on one line.

into	Optional name of a dedicated column to draw the plot into, instead of the first series' value column. The series columns (...) are still read for their values and stay visible unless hide drops them, so you can show every value column as text <i>and</i> a plot in its own column. A column of this name is created when it does not already exist (added as a new, extra column – use <code>lt_move()</code> to position it). When NULL (default), the plot is drawn into the first series' value column.
width, height	Pixel size of each cell's SVG. width may instead be a relative CSS width (e.g. "100%"), or <code>list(min =, width =)</code> , to auto-fit the column (see <code>lt_errorbar()</code>). height defaults to a size that fits the tracks when stagger is set, else a single line.
hide	If TRUE (default), the columns the plot reads are hidden, since their values are drawn into it: all but the one holding the plot, or – when into names a separate target column – all of them. Set to FALSE to keep them visible.
axis	Draw a shared horizontal axis (a baseline with tick marks and labels at "nice" round values) in the table footer under the plot column, plus faint vertical grid-lines inside each cell at the same tick positions to help read the bars. TRUE draws the axis; a character string draws the axis with that string as a caption below it (e.g. "Risk difference"); FALSE (default) draws no axis.

Details

Since the cell shows every named column, the column header defaults to all their names joined by " / " (e.g. "Before / After"); override it with a later `lt_label()` on that column.

Value

x with the dot-plot column recorded.

Examples

```
d = data.frame(
  group = c("A", "B", "C"), before = c(3, 5, 4), after = c(6, 7, 5)
)
# one dot per column, colored and keyed by a footer legend, with an axis
lt(d) |> lt_dotplot(~ before + after, color = TRUE)
# custom colors and labels
lt(d) |> lt_dotplot(
  ~ before + after, color = c("#999", "#1a9641"),
  labels = c("Baseline", "Follow-up")
)
# draw into a dedicated column, keeping the value columns visible as text
lt(d) |> lt_dotplot(~ before + after, into = "plot", hide = FALSE)
```

lt_errorbar

*Draw an inline error-bar plot in a column***Description**

Render a numeric column's cells as a small inline SVG: a point at the estimate and a horizontal bar (with end caps) from the lower to the upper bound, drawn on a scale shared across the column so rows are comparable at a glance (handy for a forest plot of effect sizes with confidence intervals). It is lightweight: only the numbers travel to the client, and the SVG is drawn in the browser — so for an interactive table ([lt_interactive\(\)](#)) only the rows on the current page are drawn.

Usage

```
lt_errorbar(
  x,
  ...,
  limits = NULL,
  ref = NULL,
  into = NULL,
  width = 160,
  height = NULL,
  hide = TRUE,
  axis = FALSE,
  color = FALSE,
  labels = NULL
)
```

Arguments

<code>x</code>	An lt() object.
<code>...</code>	One or more error-bar series, each naming the point estimate, the lower bound, and the upper bound, in that order: either a two-sided formula value $\sim$ lower + upper (estimate on the left, bounds on the right) or a length-3 character vector / integer positions <code>c(value, lower, upper)</code> . Give several to stack several series per cell. The plot is drawn in the first series' value column's cells, or into <code>into</code> when given.
<code>limits</code>	Numeric <code>c(min, max)</code> for the shared horizontal scale. Defaults to the range of all the columns' finite values. Values outside the scale are clamped to the edges.
<code>ref</code>	Optional value at which to draw a vertical reference line (e.g. 0 for a risk difference, 1 for an odds ratio).
<code>into</code>	Optional name of a dedicated column to draw the plot into, instead of the first series' value column. The series columns (...) are still read for their values and stay visible unless <code>hide</code> drops them, so you can show every value column as text <i>and</i> a plot in its own column. A column of this name is created when it does not already exist (added as a new, extra column — use lt_move() to position it). When <code>NULL</code> (default), the plot is drawn into the first series' value column.

width, height	Pixel size of each cell's SVG. width may instead be a relative CSS width such as "100%" or "80%", in which case the plot auto-fits its column: it takes that fraction of the column's width and is redrawn when the width changes (e.g. the window or a column is resized), never shrinking below the default pixel width. Use <code>list(min =, width =)</code> to set that minimum explicitly, e.g. <code>list(min = 200, width = "100%")</code> . height defaults to a size that fits the number of series (taller for more).
hide	If TRUE (default), the columns the plot reads are hidden, since their values are drawn into it: all but the one holding the plot, or – when <code>into</code> names a separate target column – all of them. Set to FALSE to keep them visible.
axis	Draw a shared horizontal axis (a baseline with tick marks and labels at "nice" round values) in the table footer under the plot column, plus faint vertical grid-lines inside each cell at the same tick positions to help read the bars. TRUE draws the axis; a character string draws the axis with that string as a caption below it (e.g. "Risk difference"); FALSE (default) draws no axis.
color	Series colors. FALSE (default) draws every series in one default color; TRUE assigns colors from the current palette (<code>grDevices::palette()</code>); a character vector of CSS colors sets them explicitly (recycled to the number of series). When colored, a legend is drawn below the plot in the footer.
labels	Legend labels, one per series, used when colored. Defaults to the series' value-column names.

Details

Several series can share one cell: pass one triple per series and each row shows one point-and-bar per series, stacked vertically on the same shared scale (e.g. a risk difference for each of several treatment arms). Color the series and a legend is drawn in the footer, as for `lt_dotplot()`.

Value

x with the error-bar column recorded.

Examples

```
d = data.frame(
  term = c("A", "B", "C"), est = c(0.2, -0.1, 0.4),
  lo = c(0.0, -0.3, 0.1), hi = c(0.4, 0.1, 0.7)
)
# a two-sided formula names the estimate (LHS) and the bounds (RHS)
lt(d) |> lt_errorbar(est ~ lo + hi, ref = 0)
# keep the bounds columns visible and add a labeled axis
lt(d) |> lt_errorbar(est ~ lo + hi, hide = FALSE, axis = "Effect")
# draw into a dedicated column, keeping every value column visible as text
lt(d) |> lt_errorbar(est ~ lo + hi, into = "forest", hide = FALSE)
# auto-fit the plot to its column (grows/shrinks as the column is resized)
lt(d) |> lt_errorbar(est ~ lo + hi, into = "forest", width = "100%")
# several series stacked in each cell, colored with a footer legend
d2 = data.frame(
  term = c("A", "B"), e1 = c(0.2, 0.4), l1 = c(0, 0.2), u1 = c(0.4, 0.6),
```

```

    e2 = c(-0.1, 0.1), l2 = c(-0.3, -0.1), u2 = c(0.1, 0.3)
  )
  lt(d2) |> lt_errorbar(
    e1 ~ l1 + u1, e2 ~ l2 + u2, ref = 0, color = TRUE,
    labels = c("Drug A", "Drug B")
  )

```

lt_export

Export an lt table to a file

Description

Save a table to disk. The output format is chosen from the file extension of output: .html writes an HTML table, .pdf writes a vector PDF, and any other extension writes a PNG. PDF and PNG are produced by rendering the table in a headless Chromium browser (via `xfun::browser_print()`).

Usage

```

lt_export(
  x,
  output = "lt.html",
  method = c("auto", "node", "browser", "raw"),
  css = TRUE,
  fragment = FALSE,
  tidy = FALSE,
  crop = TRUE,
  width = NULL,
  padding = 8,
  browser = NULL,
  ...
)

```

Arguments

x	An <code>lt_tbl</code> object.
output	Output file path. Its extension selects the format: .html, .pdf, or (otherwise) PNG. If NA, the HTML is returned as a string instead of being written to a file.
method	How to produce the HTML <table>: "auto", "node", "browser", or "raw" (see Details). Applies only to .html output.
css	Whether to include the lt.css runtime stylesheet in the HTML output. User CSS from <code>lt_css()</code> is always included. Applies only to .html output.
fragment	If FALSE (default), wrap the HTML in a full HTML document; if TRUE, return only the table fragment. Applies only to .html output.
tidy	Whether to pretty-print the baked <table> with line breaks and indentation. Applies only to .html output baked by "node" or "browser" (ignored for method = "raw", which is a JavaScript spec).

crop	Whether to crop the PDF/PNG tightly to the table, removing the surrounding page whitespace. This adds a preliminary browser pass to measure the rendered table. Set to FALSE for the default full page. Cropping PNG output requires the magick package; without it, PNG falls back to the full page (with a warning).
width	The width of the table in CSS pixels. By default (NULL) it shrinks to the table's natural width. A smaller width wraps cell content; a larger one pads the table.
padding	Padding in CSS pixels to keep around the table when cropping. A single value (all sides) or a length-two vector c(vertical, horizontal).
browser	Path to the Chromium-based browser; passed to <code>xfun::browser_print()</code> . NULL (default) auto-detects.
...	Passed to <code>xfun::browser_print()</code> for PDF/PNG output. To add extra Chromium flags via its args, keep "default" in the vector (e.g. <code>args = c("default", "--force-device-scale-factor=2")</code>); "default" is what makes <code>browser_print()</code> add the headless and print flags, so dropping it opens the file in a desktop browser and writes nothing.

Details

For .html output, method controls how the <table> is produced: "raw" writes the JavaScript-spec HTML, so the table is built in the browser by the lt.js runtime when the file is viewed; the other methods bake a static <table> up front by running lt.js once (via "node" in Node.js or "browser" in a headless Chromium browser; "auto" picks Node if available, else the browser), so the saved file needs no JavaScript to view. method, css, fragment, and tidy apply only to .html output.

Value

The output path, or (when output is NA) the HTML as a string.

Global option

When the option `lt.lt_static` is set to a list of arguments (e.g., `options(lt.lt_static = list(css = FALSE))`), the `knit_print` and `record_print` methods emit the same static HTML table as `lt_export(x, "*.html")` (using those arguments as method/css/fragment) instead of the default JavaScript-based spec. This is useful for output formats that support raw HTML but cannot run JavaScript (e.g., GitHub Flavored Markdown).

Examples

```
tbl = lt(head(mtcars))

# HTML with the JavaScript spec (table built by lt.js when viewed)
f1 = tempfile(fileext = '.html')
lt_export(tbl, f1, method = 'raw') # file output

# Bake a static <table> (needs Node.js or a headless browser).
if (lt:::can_bake())
  lt_export(tbl, NA, method = 'auto', fragment = TRUE, css = FALSE, tidy = TRUE)

# PDF / PNG are rendered in a headless browser and cropped to the table.
```

```
f2 = tempfile(fileext = '.pdf')
f3 = tempfile(fileext = '.png')
if (lt::has_browser()) {
  lt_export(tbl, f2)
  lt_export(tbl, f3, width = 400)
}

unlink(c(f1, f2, f3))
```

lt_footnote

Add a footnote

Description

Attaches a footnote text to a table region. Footnotes are numbered automatically in the order they are added (de-duplicated by text).

Usage

```
lt_footnote(x, text, where, columns = NULL, rows = NULL, match = NULL)
```

Arguments

x	An <code>lt()</code> object.
text	Footnote text.
where	One of 'title', 'subtitle', 'column', 'spanner', 'group', or 'body'.
columns	Character names, integer positions, or a one-sided formula (for 'column' or 'body'); predicate formulas such as <code>~ endsWith(., "_time")</code> are also allowed (see <code>lt_align()</code>). For 'group' with <code>match = "starts_with"</code> , a single prefix string.
rows	Integer vector of 1-based row indices (for 'body'; NULL means all rows).
match	For where = "group": one of "exact" (default), "starts_with", or "all".

Value

x with the footnote recorded.

Examples

```
lt(head(mtcars)) |>
  lt_footnote("Source: 1974 Motor Trend US magazine.", "title")
# raw HTML footnote (wrap in I())
lt(head(mtcars)) |>
  lt_footnote(I("See <a href='#'>docs</a>."), "title")

# a footnote for each of the six locations
d = data.frame(
```

```

  Arm   = c("Treatment", "Treatment", "Control", "Control"),
  Stat  = c("n", "Mean", "n", "Mean"),
  Value = c(31, 6.8, 30, 4.2),
  SE    = c(NA, 0.42, NA, 0.38)
)
lt(d) |>
  lt_group(~ Arm) |>
  lt_header(title = "Study Results", subtitle = "Two arms") |>
  lt_spanner(Summary ~ Value + SE) |>
  lt_footnote("Source: simulated trial data.", "title") |>
  lt_footnote("Intention-to-treat population.", "subtitle") |>
  lt_footnote("Standard error of the mean.", "column", "SE") |>
  lt_footnote("Per-arm summary statistics.", "spanner", "Summary") |>
  lt_footnote("Received active drug.", "group", "Treatment") |>
  lt_footnote("SE undefined for counts.", "body", "SE", rows = c(1, 3))

```

lt_format

*Format numeric columns***Description**

Control the number of decimal places and thousands separator for numeric columns. Columns passed to this function are excluded from automatic formatting (see the `auto_format` argument of `lt()`). To disable auto-format for a column without otherwise changing its display, call `lt_format(x, ~col)` with no other arguments.

Usage

```

lt_format(
  x,
  columns,
  decimals = NULL,
  sig_digits = NULL,
  big_mark = NULL,
  percent = NULL,
  prefix = NULL,
  suffix = NULL
)

```

Arguments

<code>x</code>	An <code>lt()</code> object.
<code>columns</code>	Columns to select: a character vector of column names, integer positions, or a one-sided formula. A formula whose right-hand side references the pronoun <code>.</code> (the character vector of column names) is evaluated as a predicate, e.g. <code>~endsWith(., "_time")</code> selects all columns whose names end with <code>_time</code> ; otherwise the bare names on the right-hand side are used, e.g. <code>~ x + y</code> .
<code>decimals</code>	Number of decimal places (default <code>NULL</code> means no change).

sig_digits	Number of significant digits (e.g., 3). Cannot be combined with decimals. Unlike decimals, which fixes the number of places after the decimal point, this targets the total number of meaningful digits, which is useful for columns spanning several orders of magnitude.
big_mark	Thousands separator (e.g., ", "). NULL or "" means none.
percent	If TRUE, multiply values by 100 and append "%". If "%", only append "%" without multiplying (for values already in percent scale).
prefix	String prepended to each formatted value (e.g., "\$").
suffix	String appended to each formatted value (e.g., " kg").

Value

x with the formatting recorded.

Examples

```
lt(head(mtcars)) |> lt_format(~ mpg + wt, decimals = 1, big_mark = ",")
d = data.frame(Item = c("A", "B"), Price = c(1234.5, 678.9))
lt(d) |> lt_format(~ Price, decimals = 2, big_mark = ", ", prefix = "$")
lt(d) |> lt_format(~ Price, sig_digits = 3)
```

lt_group	<i>Define row groups</i>
----------	--------------------------

Description

Partition rows into labeled groups. Pass column names to group by those columns' values (the columns are removed from the body and rendered as rowspan cells on the left). Use sep = TRUE to render groups as full-width separator rows instead of rowspan.

Usage

```
lt_group(x, ..., sep = "auto", sort = TRUE)
```

Arguments

x	An <code>lt()</code> object.
...	A column name, integer position, or formula (e.g., <code>~col</code> or <code>~col1 + col2</code>) to group by column values, or named arguments of the form <code>"Label" = rows</code> (integer vector of 1-based row indices) for manual groups. Unnamed character strings reorder previously defined groups.
sep	If TRUE, render groups as full-width separator rows instead of the default rowspan style. Only supports a single grouping column. The default 'auto' uses separator rows when there is a single character grouping column with any value longer than 20 characters (numeric columns always use rowspan).
sort	If TRUE (default), sort rows by group columns so that identical group values are contiguous. Set to FALSE to preserve the original row order.

Value

x with the row groups recorded.

Examples

```
# Group by a column (rowspan, default)
d = data.frame(arm = c("Placebo", "Placebo", "Treatment", "Treatment"),
               stat = c("n", "Mean", "n", "Mean"), value = c(30, 4.2, 31, 6.8))
lt(d) |> lt_group(~ arm)

# Separator-row style
lt(d) |> lt_group(~ arm, sep = TRUE)

# Manual groups (always separator rows)
lt(head(mtcars)) |>
  lt_group("First three" = 1:3, "Last three" = 4:6)
```

lt_header

Add a title and subtitle

Description

Add a title and subtitle

Usage

```
lt_header(x, title = NULL, subtitle = NULL)
```

Arguments

x	An <code>lt()</code> object.
title	A character scalar. Wrap in <code>I()</code> to treat as raw HTML.
subtitle	A character scalar. Wrap in <code>I()</code> to treat as raw HTML.

Value

x with the header recorded.

Examples

```
lt(head(mtcars)) |> lt_header("Motor Trend Cars", "First 6 rows")
# raw HTML title
lt(head(mtcars)) |> lt_header(I("<em>Motor Trend</em> Cars"))
```

lt_hide	<i>Hide columns</i>
---------	---------------------

Description

Drop columns from the rendered table without removing them from the data. The values still travel to the client in the spec, so they remain available to things that read the whole row, such as an interactive table's row detail (see the `detail` argument of `lt_interactive()`).

Usage

```
lt_hide(x, columns)
```

Arguments

<code>x</code>	An <code>lt()</code> object.
<code>columns</code>	Columns to select: a character vector of column names, integer positions, or a one-sided formula. A formula whose right-hand side references the pronoun <code>.</code> (the character vector of column names) is evaluated as a predicate, e.g. <code>~endsWith(., "_time")</code> selects all columns whose names end with <code>_time</code> ; otherwise the bare names on the right-hand side are used, e.g. <code>~ x + y</code> .

Value

`x` with the hidden columns recorded.

Examples

```
lt(head(mtcars)) |> lt_hide(~ gear + carb)
```

lt_html	<i>Render column cells as raw HTML</i>
---------	--

Description

Mark whole columns so their body cells are emitted as raw HTML instead of being escaped. By default every cell is HTML-escaped (so `<`, `>`, and `&` render as literal text); use this to embed links, emphasis, or other markup in a column's cells.

Usage

```
lt_html(x, columns)
```

Arguments

`x` An `lt()` object.

`columns` Columns whose cells are raw HTML: character names, integer positions, or a one-sided formula (predicate formulas such as `~endsWith(., "_time")` are also allowed; see `lt_align()`). If missing, all columns are treated as raw HTML.

Details

To emit raw HTML in other regions (title, column labels, footnotes, ...), wrap the text in `I()` in the corresponding `lt_*`() function instead.

Value

`x` with the raw-HTML columns recorded.

Examples

```
d = data.frame(
  name = c("<a href='#a'>A</a>", "<a href='#b'>B</a>"), n = 1:2
)
lt(d) |> lt_html(~ name)
```

lt_indent	<i>Indent rows</i>
-----------	--------------------

Description

Add hierarchical indentation to the first column of specified rows.

Usage

```
lt_indent(x, rows, level = 1)
```

Arguments

`x` An `lt()` object.

`rows` Integer vector of 1-based row indices to indent.

`level` Indent level (default 1). Each level adds one unit of left padding.

Value

`x` with the indentation recorded.

Examples

```
d = data.frame(label = c("Overall", "Male", "Female"), n = c(100, 55, 45))
lt(d) |> lt_indent(2:3)
```

lt_interactive	<i>Enable interactive table features</i>
----------------	--

Description

Make a table interactive in the browser: pagination, a table-wide search box, click-to-sort column headers, per-column filter boxes, and resizable columns. The features are handled client-side by an opt-in JavaScript extension, loaded only for tables that call this function.

Usage

```
lt_interactive(
  x,
  sort = TRUE,
  search = TRUE,
  filter = FALSE,
  pager = c(10, 25, 50, 100),
  resize = FALSE,
  hide = NULL,
  detail = NULL,
  download = FALSE
)
```

Arguments

x	An <code>lt()</code> object.
sort	Whether clicking a column header sorts the table by that column (cycling ascending, descending, then unsorted); shift-clicking adds a column as a further tie-breaker, so several columns can sort at once. Sorting uses the raw values, so numeric columns sort numerically. Can also request an initial sort by one or more columns, given either as a character vector of column names or as a one-sided formula (e.g. <code>~ x + -y</code>); a name prefixed with <code>-</code> (or a negated formula term) sorts that column descending, e.g. <code>c('g', '-x')</code> or <code>~ g + -x</code> .
search	Whether to show a table-wide search box. A row is kept when any cell matches the term. Besides plain substring matching, a term that references the cell variable <code>x</code> (e.g. <code>x > 5</code> or <code>x != "A"</code>) is evaluated as a JavaScript expression against each cell's value.
filter	The per-column filters. <code>TRUE</code> shows a filter box under every column header, matching terms the same way as <code>search</code> but against that column only (a row is kept when it passes every filter and the search). A character vector of column names restricts the boxes to those columns. A named list gives each named column its own filter: <code>TRUE</code> for a plain box, <code>"select"</code> for a value dropdown, <code>"range"</code> for a two-thumb range slider, or <code>"checklist"</code> for a popover of checkboxes that keeps the rows whose value is checked (the dropdown's / checklist's choices and the slider's ends are taken from the column's data). For a custom label or hand-set options, pass a list instead of the string, e.g. <code>list(type =</code>

	"select", label = "Cylinders") or <code>list(type = "range", min = 0, max = 100)</code>. One <i>unnamed</i> entry in the list is the default applied to every other visible column, so <code>list(TRUE, cyl = "select")</code> boxes every column but gives <code>cyl</code> a dropdown. A column with nothing to filter (every value blank) gets no box, even under that default. A typed filter renders as a funnel + popover (holding the widget <i>and</i> an expression box, kept in sync) under its column header, or — when the column is hidden from the table (e.g. with <code>lt_hide()</code>) — as a chip in the control bar, a natural way to offer a control for a value you don't show. Add <code>box = FALSE</code> to a typed spec's list to drop the expression box, restricting the column to the widget's own choices (e.g. a single-select dropdown with no custom set). The widget hides itself whenever a hand-typed expression outruns it (a value it cannot represent), so it never shows a value out of step with the box.
pager	The page sizes to offer, as a vector of row counts; the first one is used initially. Paging shows that many of the filtered and sorted rows at a time, with a pager (first, previous, next, last), the row range, and (for more than one size) a selector below the table. <code>Inf</code> is a valid size (every row on one page), offered as an infinity symbol in the selector. The selector is omitted when the table has no more rows than the smallest page size (no size would split it), though the pager's nav and row-count readout stay, the latter being useful as a filter narrows the rows. Use <code>FALSE</code> or <code>NULL</code> to show all rows with no pager at all.
resize	Whether to let the reader drag a column's right edge to resize it (double-clicking the edge fits the column to its content). Widening a column widens the table, leaving the other columns as they are. Initial widths can be set with <code>lt_width()</code>.
hide	Column-visibility control: an eye button at the start of the search row opens a checklist of every column, where unchecking a column hides it (header and cells) and re-checking restores it. <code>TRUE</code> adds the menu with all columns shown; column names (a character vector or a one-sided formula, e.g. <code>~ x + y</code>) adds it with those columns hidden to begin with (every column is still listed, so any can be toggled). <code>NULL</code> (the default) adds no menu.
detail	Row detail (drill-down). An expand caret is added to each row; clicking it reveals a table built from that row below it. Give the columns to show, as a character vector of names or a one-sided formula (e.g. <code>~ gear + carb</code>); their displayed (formatted) values are laid out as a one-row table. The columns can be ones hidden from the main table with <code>lt_hide()</code>, a natural way to tuck extra fields into the detail. For full control, pass a JavaScript callback instead, wrapped in <code>js()</code>: it is called as <code>(row, index, display)</code> — <code>row</code> the raw values, <code>display</code> the formatted text, each keyed by column name — and returns a table spec, which may carry its own interactive field to make the detail table interactive. <code>NULL</code> (the default) adds no row detail.
download	Whether to add a button that downloads the table as a CSV file. The file holds the current view (every row the filters and search keep, in sort order, across all pages) of the table's columns. <code>TRUE</code> adds the button with a default file name; a string sets the file name (a <code>.csv</code> extension is added if missing). For more control, pass a list with any of <code>file</code> (the file name) and <code>mode</code>: <code>"display"</code> (the default) exports the displayed cell text, while <code>"raw"</code> exports the underlying unformatted values (full precision, no thousands separators or percent signs), handier to re-import for analysis. Row detail (<code>detail</code>) is never included. <code>FALSE</code> (the default) adds no button.

Details

Sorting and filtering respect the row structure, happening *within* each group or subtree rather than across it. Separator row groups (`lt_group()` with `sep = TRUE`, or manual groups) keep their order, and a group whose rows all filter out drops its header. Indentation (`lt_indent()`) sorts siblings under each parent, and an indented child keeps its ancestors visible for context. Rowspan row groups (`lt_group()` with its default rowspan rendering) sort and filter within each run and clicking a group header reorders its blocks; with several group columns the view stays nested (an inner column reorders only within its parent block), and the spanning cells are redrawn to match.

Value

x with interactivity enabled.

Examples

```
lt(head(mtcars)) |> lt_interactive()
# search only, no click-to-sort or pagination
lt(head(mtcars)) |> lt_interactive(sort = FALSE, pager = FALSE)
# 5 rows at a time, or all of them, with a filter box on one column
lt(mtcars) |> lt_interactive(filter = 'cyl', pager = c(5, Inf))
# resizable columns
lt(head(mtcars)) |> lt_interactive(resize = TRUE)
# a column-visibility menu (all columns shown, or with some hidden to start)
lt(head(mtcars)) |> lt_interactive(hide = TRUE)
lt(head(mtcars)) |> lt_interactive(hide = ~ hp + drat)
# an initial sort by cyl, then mpg descending within each
lt(mtcars) |> lt_interactive(sort = ~ cyl + -mpg)
# expandable row detail (drill-down): name the columns to show. Here gear and
# carb are hidden from the main table but revealed in the detail that appears
# when a row is expanded.
lt(head(mtcars)) |>
  lt_hide(~ gear + carb) |>
  lt_interactive(detail = ~ gear + carb)
# for full control, pass a JavaScript callback yourself: (raw row, index,
# displayed row); here showing the row number and mpg as the table displays it
lt(head(mtcars)) |> lt_interactive(
  detail = js('(row, i, d) => ({ data: { metric: ["row", "mpg"],
    value: [i, d.mpg] } })')
)
# a button to download the current view as a CSV file
lt(head(mtcars)) |> lt_interactive(download = TRUE)
lt(head(mtcars)) |> lt_interactive(download = 'cars.csv')
# export the raw (unformatted) values instead of the displayed text
lt(head(mtcars)) |> lt_interactive(download = list(file = 'cars.csv', mode = 'raw'))
```

lt_label

Rename column labels

Description

Override column headers without modifying the underlying data frame.

Usage

```
lt_label(x, ...)
```

Arguments

x An `lt()` object.

... Named arguments of the form `col_name = "Display Label"`. Wrap a label in `I()` to treat it as raw HTML. Alternatively, pass a single named list or named character vector mapping column names to labels, which is convenient when the labels are computed programmatically, e.g. `lt_label(x, c(mpg = "Miles/Gallon", cyl = "Cylinders"))`, or as a single list(`names`, `labels`) of two parallel vectors.

Value

`x` with the column label overrides recorded.

Examples

```
lt(head(mtcars)) |> lt_label(mpg = "Miles/Gallon", cyl = "Cylinders")
# raw HTML label (wrap in I())
lt(head(mtcars)) |> lt_label(mpg = I("Miles/Gallon<sup>*</sup>"))
# a single named list or vector of labels
lt(head(mtcars)) |> lt_label(c(mpg = "Miles/Gallon", cyl = "Cylinders"))
# a single list(names, labels) of two parallel vectors
lt(head(mtcars)) |> lt_label(list(c("mpg", "cyl"), c("Miles/Gallon", "Cylinders")))
```

lt_merge

Merge columns

Description

Combine values from multiple columns into a single display column using a pattern. Source columns (all except the first) are hidden by default.

Usage

```
lt_merge(x, columns, pattern = NULL, hide = TRUE)
```

Arguments

x	An <code>lt()</code> object.
columns	Character names, integer positions, or a one-sided formula (predicate formulas such as <code>~ endsWith(., "_time")</code>) are also allowed; see <code>lt_align()</code> . The first column is the target (receives merged content); the rest are sources.
pattern	A glue-style template using <code>\{1\}</code> , <code>\{2\}</code> , etc. to refer to columns by position. Wrap sections in <code><<</code> and <code>>></code> for conditional NA handling: <code>"\{1\}<< (\{2\})>>"</code> drops the wrapped portion when any referenced value is missing/empty. If <code>NULL</code> , columns are concatenated separated by spaces.
hide	If <code>TRUE</code> (default), source columns (all but the first) are automatically hidden.

Value

x with the merge recorded.

Examples

```
d = data.frame(stat = c("Mean", "SD"), value = c(4.2, 1.1), ci = c("(2.0, 6.4)", "(0.5, 1.7)"))
lt(d) |> lt_merge(~ value + ci, pattern = "{1} {2}")
```

<code>lt_move</code>	<i>Move columns</i>
----------------------	---------------------

Description

Rearrange column display order without modifying the data frame.

Usage

```
lt_move(x, columns, after = NULL)
```

Arguments

x	An <code>lt()</code> object.
columns	Columns to select: a character vector of column names, integer positions, or a one-sided formula. A formula whose right-hand side references the pronoun <code>.</code> (the character vector of column names) is evaluated as a predicate, e.g. <code>~ endsWith(., "_time")</code> selects all columns whose names end with <code>_time</code> ; otherwise the bare names on the right-hand side are used, e.g. <code>~ x + y</code> .
after	Column name or integer position after which to place the moved columns. Use <code>NULL</code> to move to the start.

Value

x with the column move recorded.

Examples

```
lt(head(mtcars)) |> lt_move(~ gear + carb, after = "mpg")
```

lt_note	<i>Add a note</i>
---------	-------------------

Description

Notes are rendered in the table footer below numbered footnotes.

Usage

```
lt_note(x, text)
```

Arguments

x An `lt()` object.

text Note text. Wrap in `I()` to treat as raw HTML.

Value

x with the note recorded.

Examples

```
lt(head(mtcars)) |> lt_note("CI = confidence interval.")
# raw HTML note (wrap in I())
lt(head(mtcars)) |> lt_note(I("CI = <em>confidence interval</em>."))
```

lt_output	<i>Shiny bindings for lt</i>
-----------	------------------------------

Description

`lt_output()` creates a UI placeholder; `render_lt()` supplies the table spec from the server. Together they render an `lt()` table as a custom Shiny output — no `renderUI()` involved.

Usage

```
lt_output(outputId, ...)
```

```
render_lt(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

outputId	Output variable name to read the table from.
...	Reserved for future use.
expr	An expression that returns an <code>lt()</code> object.
env	Environment in which to evaluate <code>expr</code> .
quoted	Whether <code>expr</code> is already quoted.

Value

`lt_output()` returns a Shiny UI element; `render_lt()` returns a render function.

Examples

```
if (interactive()) {
  library(shiny)
  ui = fluidPage(lt_output("tbl"))
  server = function(input, output) {
    output$tbl = render_lt(lt(head(mtcars)) |> lt_header("Motor Trend"))
  }
  shinyApp(ui, server)
}
```

lt_spanner	<i>Add a column spanner</i>
------------	-----------------------------

Description

A spanner is a label rendered above a contiguous group of column headers.

Usage

```
lt_spanner(x, label, columns, sep = "[._]")
```

Arguments

x	An <code>lt()</code> object.
label	A character scalar — the spanner text. Alternatively, a two-sided formula <code>Label ~ col1 + col2</code> providing both the label (LHS) and columns (RHS). When missing, spanners are inferred from column names.
columns	Columns to span: character names, integer positions, or a one-sided formula (predicate formulas such as <code>~ endsWith(., "_time")</code> are also allowed; see <code>lt_align()</code>). When missing, inferred from column names.
sep	Separator pattern for auto-inference (default <code>"[._]"</code>).

Details

When called with no label or columns, infers spanners from column names by splitting on the first . or _ separator. Contiguous columns sharing a prefix are grouped under that prefix, and column labels are shortened to the suffix.

Value

x with the spanner recorded.

Note

The columns must be contiguous in the body of the table.

Examples

```
tbl = lt(head(iris))
# Explicit spanner
tbl |> lt_spanner(Sepal ~ Sepal.Length + Sepal.Width)
# Auto-infer from column names
tbl |> lt_spanner()
# raw HTML label (wrap in I())
tbl |> lt_spanner(I("<em>Sepal</em>"), ~ Sepal.Length + Sepal.Width)
```

lt_sparkline

Draw an inline sparkline in a column

Description

Render a per-row series of numbers as a small inline SVG line or bar chart (a sparkline). The series for each row can come from a single list-column (each element a numeric vector) or from several numeric columns read across the row. Like [lt_errorbar\(\)](#), it is lightweight: only the numbers travel to the client and the SVG is drawn in the browser, so an interactive table ([lt_interactive\(\)](#)) draws only the rows on the current page.

Usage

```
lt_sparkline(
  x,
  columns,
  type = c("line", "bar"),
  limits = NULL,
  color = NULL,
  width = 120,
  height = 20,
  hide = TRUE
)
```

Arguments

x	An <code>lt()</code> object.
columns	The column(s) holding each row's series. Either a single list-column (a one-sided formula <code>~ col</code> , a name, or a position) whose cells are numeric vectors, or several numeric columns (<code>~ a + b + c</code>) whose values are read left to right as the series. The sparkline is drawn in the first column's cells.
type	"line" (default) for a line chart or "bar" for a bar chart.
limits	Numeric <code>c(min, max)</code> for a shared vertical scale across all rows (so rows are comparable). The default (NULL) scales each row to its own range.
color	CSS color for the line or bars (defaults to the stylesheet's).
width, height	Pixel size of each cell's SVG. width may instead be a relative CSS width (e.g. "100%"), or <code>list(min =, width =)</code> , to auto-fit the column (see <code>lt_errorbar()</code>).
hide	If TRUE (default) and the series spans several columns, those columns after the first are hidden (their values are drawn into the plot); set to FALSE to keep them visible. Has no effect for a single list-column.

Value

x with the sparkline column recorded.

Examples

```
# a list-column: each cell is a numeric vector
d = data.frame(city = c("A", "B", "C"))
d$temp = list(c(3, 5, 8, 6, 9), c(1, 2, 2, 4, 3), c(7, 6, 5, 6, 8))
lt(d) |> lt_sparkline(~ temp)
# several columns read across the row, as bars on a shared scale
m = data.frame(name = c("X", "Y"), q1 = c(2, 5), q2 = c(4, 3),
  q3 = c(6, 7), q4 = c(5, 2))
lt(m) |> lt_sparkline(~ q1 + q2 + q3 + q4, type = "bar", limits = c(0, 8))
```

lt_spec

Extract a table's render-ready spec

Description

Return the JSON-ready spec for an `lt()` table: the raw data plus the declarative ops (and any `lt_interactive()` options), prepared exactly as the JavaScript runtime expects. This is the payload the runtime consumes, so another widget can embed an lt table by shipping this spec (e.g. `xfun::tojson(lt_spec(x))`) and rendering it client-side on demand with `LT.render(container, spec)` (or `LT.buildHtml(spec)`). Styling from `lt_css()` and CSS rules is dropped; attach the runtime assets separately (see `lt_dependency()`).

Usage

```
lt_spec(x)
```

Arguments

x An `lt_tbl` object.

Value

A named list of spec fields (data, ops, and any others set on the table), ready to serialize to JSON.

Examples

```
spec = lt_spec(lt(head(mtcars)) |> lt_interactive())
str(spec, max.level = 1)
xfun::tojson(spec) # ship this, then LT.render(container, spec) in the browser
```

lt_style	<i>Style cells</i>
----------	--------------------

Description

Apply CSS styling to specific cells. Target cells by column, row, or both. When `test` is provided, styles are applied conditionally based on cell values (evaluated in JavaScript).

Usage

```
lt_style(
  x,
  columns = NULL,
  rows = NULL,
  test = NULL,
  class = NULL,
  bold = NULL,
  italic = NULL,
  color = NULL,
  bg = NULL,
  ...
)
```

Arguments

x An `lt()` object.

columns Character names, integer positions, a one-sided formula (predicate formulas such as `~ endsWith(., "_time")` are also allowed; see `lt_align()`), or `NULL` for all.

rows Integer vector of 1-based row indices (or `NULL` for all).

test A JavaScript function as a string (e.g., `"v => v < 0"`) that receives the raw cell value and returns true to apply the style. When `NULL`, the style applies unconditionally.

class	CSS class name(s) to add to matching cells. Define the corresponding rules in an external stylesheet via <code>lt_css()</code> .
bold	Logical: apply bold weight?
italic	Logical: apply italic style?
color	Text color (any CSS color value, e.g., "red", "#06c").
bg	Background color.
...	Additional CSS properties as named arguments. Names can be camelCase (e.g., <code>borderLeft</code>) or quoted dash-case (e.g., <code>`border-left`</code>). Values are CSS strings.

Value

x with the style recorded.

Examples

```
tbl = lt(head(mtcars))
tbl |>
  lt_style("mpg", rows = 1L, bold = TRUE, borderBottom = "2px solid red")
tbl |>
  lt_style("mpg", test = "v => v > 20", class = "high") |>
  lt_css(.high = list(background = "#cfc"))
```

lt_sub	<i>Substitute cell values</i>
--------	-------------------------------

Description

Replace NA, zero, or small values with display text.

Usage

```
lt_sub(
  x,
  columns = NULL,
  missing = NULL,
  zero = NULL,
  small = NULL,
  small_text = NULL
)
```

Arguments

x	An <code>lt()</code> object.
columns	Character names, integer positions, a one-sided formula (predicate formulas such as <code>~ endsWith(., "_time")</code>) are also allowed; see <code>lt_align()</code> , or NULL for all.

missing	Replacement for NA cells in these columns (e.g., "n/a"). NULL (default) leaves them to the table-wide default, which is an em dash (–); see the <code>lt.missing</code> global option in <code>lt()</code> .
zero	Replacement for zero values (e.g., "–").
small	Threshold: values whose absolute value is below this are replaced by <code>small_text</code> .
small_text	Text shown for values below <code>small</code> (e.g., "<0.1").

Value

x with the substitution recorded.

Examples

```
d = data.frame(x = c(1, 0, NA, 0.001))
lt(d) |> lt_sub(missing = "n/a", zero = "–", small = 0.01, small_text = "<0.01")
```

lt_width	<i>Set column and table widths</i>
----------	------------------------------------

Description

Set column and table widths

Usage

```
lt_width(x, ...)
```

Arguments

x	An <code>lt()</code> object.
...	Named arguments of the form <code>col_name = "width"</code> set the widths of individual columns. A single unnamed scalar sets the width of the whole table. Widths can be any CSS value (e.g., "100px", "20%", "8em"). The column widths can instead be given as a single <code>list(names, widths)</code> of two parallel vectors.

Value

x with the widths recorded.

Examples

```
lt(head(mtcars)) |> lt_width(mpg = "100px", cyl = "50px")
# set the whole-table width, optionally with column widths
lt(head(mtcars)) |> lt_width("80%", mpg = "100px")
# a single list(names, widths) of two parallel vectors
lt(head(mtcars)) |> lt_width(list(c("mpg", "cyl"), c("100px", "50px")))
```

lt_wrap	<i>Set attributes on the table's container</i>
---------	--

Description

Set arbitrary HTML attributes on the `<div class="lt-wrap">` that wraps the rendered table (e.g. `style`, `id`, `data-*`). A class is appended to the base `lt-wrap` class rather than replacing it, and accumulates across repeated calls. Common uses are scoping custom CSS (e.g. attached with `lt_css()`) to this table alone, or giving the container a fixed width.

Usage

```
lt_wrap(x, ...)
```

Arguments

<code>x</code>	An <code>lt()</code> object.
<code>...</code>	Named HTML attributes for the container, e.g. <code>class = "striped"</code> , <code>style = "width:800px"</code> , <code>id = "tbl1"</code> . The names and values can also be given as a single list(names, values) of two parallel vectors (see <code>lt_label()</code>).

Value

`x` with the container attributes recorded.

Examples

```
lt(head(mtcars)) |> lt_wrap(class = "my-table", style = "width:400px")
```

print.lt_tbl	<i>Print an lt_tbl (opens in the viewer or browser)</i>
--------------	---

Description

Print an `lt_tbl` (opens in the viewer or browser)

Usage

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

Arguments

<code>x</code>	An <code>lt_tbl</code> object.
<code>...</code>	Passed to <code>format()</code> .

print.lt_tbl

33

Value

x, invisibly.

Examples

```
print(lt(head(mtcars)))
```

Index

`format()`, [3](#), [32](#)
`format.lt_tbl`, [2](#)

`grDevices::palette()`, [8](#), [11](#)

`htmltools::htmlDependency()`, [7](#)

`I()`, [4](#), [17](#), [19](#), [23](#), [25](#)

`js()`, [21](#)

`knit_print`, [13](#)

`lt`, [3](#)
`lt()`, [5](#), [6](#), [8](#), [10](#), [14–20](#), [23–26](#), [28–32](#)
`lt_align`, [4](#)
`lt_align()`, [14](#), [19](#), [24](#), [26](#), [29](#), [30](#)
`lt_css`, [5](#)
`lt_css()`, [12](#), [28](#), [30](#), [32](#)
`lt_date`, [6](#)
`lt_dependency`, [7](#)
`lt_dependency()`, [28](#)
`lt_dotplot`, [8](#)
`lt_dotplot()`, [11](#)
`lt_errorbar`, [10](#)
`lt_errorbar()`, [7–9](#), [27](#), [28](#)
`lt_export`, [12](#)
`lt_footnote`, [14](#)
`lt_format`, [15](#)
`lt_format()`, [3](#), [4](#)
`lt_group`, [16](#)
`lt_group()`, [22](#)
`lt_header`, [17](#)
`lt_hide`, [18](#)
`lt_hide()`, [21](#)
`lt_html`, [18](#)
`lt_html()`, [4](#)
`lt_indent`, [19](#)
`lt_indent()`, [22](#)
`lt_interactive`, [20](#)
`lt_interactive()`, [3](#), [7](#), [8](#), [10](#), [18](#), [27](#), [28](#)

`lt_label`, [22](#)
`lt_label()`, [5](#), [9](#), [32](#)
`lt_merge`, [23](#)
`lt_move`, [24](#)
`lt_move()`, [9](#), [10](#)
`lt_note`, [25](#)
`lt_output`, [25](#)
`lt_output()`, [7](#)
`lt_spanner`, [26](#)
`lt_sparkline`, [27](#)
`lt_sparkline()`, [8](#)
`lt_spec`, [28](#)
`lt_spec()`, [7](#)
`lt_style`, [29](#)
`lt_sub`, [30](#)
`lt_sub()`, [4](#)
`lt_width`, [31](#)
`lt_width()`, [21](#)
`lt_wrap`, [32](#)

`print.lt_tbl`, [32](#)

`record_print`, [13](#)
`render_lt (lt_output)`, [25](#)
`render_lt()`, [7](#)

`xfun::browser_print()`, [12](#), [13](#)