The Family of *Table Option Functions*

With the `opt_*()` functions, we have an easy way to set commonly-used table options without having to use `tab_options()` directly. For instance, we can modify the set of marks to use with footnotes, turn on row striping, change the alignment of the table header, and much more.

`opt_footnote_marks()`: Modify the set of footnote marks

Alter the footnote marks for any footnotes that may be present in the table. Either a vector of marks can be provided (including Unicode characters), or, a specific keyword could be used to signify a preset sequence.

EXAMPLES

Use `sza` to create a **gt** table, adding three footnotes. Call `opt_footnote_marks()` to specify which footnote marks to use.

```
sza %>%
  dplyr::group_by(latitude, tst) %>%
  dplyr::summarize(
    SZA.Max = max(sza),
    SZA.Min = min(sza, na.rm = TRUE)
  ) %>%
  dplyr::ungroup() %>%
  dplyr::filter(latitude == 30, !is.infinite(SZA.Min)) %>%
  dplyr::select(-latitude) %>%
  gt(rowname_col = "tst") %>%
  tab_spanner_delim(delim = ".") %>%
  fmt_missing(
    columns = everything(),
    missing_text = "90+"
  ) %>%
  tab_stubhead("TST") %>%
  tab_footnote(
    footnote = "True solar time.",
    locations = cells_stubhead()
  ) %>%
  tab_footnote(
    footnote = "Solar zenith angle.",
    locations = cells_column_spanners(spanners = "SZA")
  ) %>%
  tab_footnote(
    footnote = "The Lowest SZA.",
    locations = cells_body(columns = everything(), rows = "1200")
  ) %>%
  opt_footnote_marks(marks = "standard")
```

```
## Warning in min(sza, na.rm = TRUE): no non-missing arguments to min; returning
## Inf

## Warning in min(sza, na.rm = TRUE): no non-missing arguments to min; returning
## Inf

## Warning in min(sza, na.rm = TRUE): no non-missing arguments to min; returning
## Inf

## Warning in min(sza, na.rm = TRUE): no non-missing arguments to min; returning
## Inf

## Warning in min(sza, na.rm = TRUE): no non-missing arguments to min; returning
## Inf

## Warning in min(sza, na.rm = TRUE): no non-missing arguments to min; returning
## Inf

## Warning in min(sza, na.rm = TRUE): no non-missing arguments to min; returning
## Inf

## Warning in min(sza, na.rm = TRUE): no non-missing arguments to min; returning
## Inf
```

```
## `summarise()` regrouping output by 'latitude' (override with `.groups` argument)
```

	SZA†	
TST*	Max	Min
0530	90+	84.7
0600	90+	78.7
0630	90+	72.5
0700	89.4	66.3
0730	83.7	60.0
0800	78.3	53.5
0830	73.2	47.1
0900	68.4	40.6
0930	64.1	34.1
1000	60.4	27.7

*True solar time.

†Solar zenith angle.

‡The Lowest SZA.

	SZA†	
TST*	Max	Min
1030	57.3	21.2
1100	55.0	15.1
1130	53.5	9.6
1200	53.0‡	6.9‡

*True solar time.

†Solar zenith angle.

‡The Lowest SZA.

`opt_row_striping()` : Option to add or remove row striping

By default, a **gt** table does not have row striping enabled. However, this function allows us to easily enable or disable striped rows in the table body.

EXAMPLE

Use `exibble` to create a **gt** table with a number of table parts added. Next, we add row striping to every second row with the `opt_add_row_striping()` function.

```
exibble %>%
  gt(rowname_col = "row", groupname_col = "group") %>%
  summary_rows(
    groups = "grp_a",
    columns = c(num, currency),
    fns = list(
      min = ~min(., na.rm = TRUE),
      max = ~max(., na.rm = TRUE)
    )) %>%
  grand_summary_rows(
    columns = currency,
    fns = list(total = ~sum(., na.rm = TRUE))
  ) %>%
  tab_source_note(source_note = "This is a source note.") %>%
  tab_footnote(
    footnote = "This is a footnote.",
    locations = cells_body(columns = 1, rows = 1)
  ) %>%
  tab_header(
    title = "The title of the table",
    subtitle = "The table's subtitle"
  ) %>%
  opt_row_striping()
```

The title of the table		
The table's subtitle		
This is a source note.		

The title of the table The table's subtitle							
	num	char	fctr	date	time	datetime	currency
grp_a							
row_1	1.111e-01[1]	apricot	one	2015-01-15	13:35	2018-01-01 02:22	49.950
row_2	2.222e+00	banana	two	2015-02-15	14:40	2018-02-02 14:33	17.950
row_3	3.333e+01	coconut	three	2015-03-15	15:45	2018-03-03 03:44	1.390
row_4	4.444e+02	durian	four	2015-04-15	16:50	2018-04-04 15:55	65100.000
min	0.11	—	—	—	—	—	1.39
max	444.40	—	—	—	—	—	65,100.00
grp_b							
row_5	5.550e+03	NA	five	2015-05-15	17:55	2018-05-05 04:00	1325.810
row_6	NA	fig	six	2015-06-15	NA	2018-06-06 16:11	13.255
row_7	7.770e+05	grapefruit	seven	NA	19:10	2018-07-07 05:22	NA
row_8	8.880e+06	honeydew	eight	2015-08-15	20:20	NA	0.440
total	—	—	—	—	—	—	66,508.79

[1] This is a footnote.

This is a source note.

`opt_align_table_header()` : Option to align the table header

By default, a table header added to a **gt** table has center alignment for both the title and the subtitle elements. This function allows us to easily set the horizontal alignment of the title and subtitle to the left or right by using the `"align"` argument.

EXAMPLE

Use `exibble` to create a **gt** table with a number of table parts added. The header (consisting of the title and the subtitle) are to be aligned to the left with the `opt_align_table_header()` function.

```
exibble %>%
  gt(rowname_col = "row", groupname_col = "group") %>%
  summary_rows(
    groups = "grp_a",
    columns = c(num, currency),
    fns = list(
      min = ~min(., na.rm = TRUE),
      max = ~max(., na.rm = TRUE)
    )
  ) %>%
  grand_summary_rows(
    columns = currency,
    fns = list(total = ~sum(., na.rm = TRUE))
  ) %>%
  tab_source_note(source_note = "This is a source note.") %>%
  tab_footnote(
    footnote = "This is a footnote.",
    locations = cells_body(columns = 1, rows = 1)
  ) %>%
  tab_header(
    title = "The title of the table",
    subtitle = "The table's subtitle"
  ) %>%
  opt_align_table_header(align = "left")
```

The title of the table

The table's subtitle

	num	char	fctr	date	time	datetime	currency
grp_a							
row_1	1.111e-01[1]	apricot	one	2015-01-15	13:35	2018-01-01 02:22	49.950
row_2	2.222e+00	banana	two	2015-02-15	14:40	2018-02-02 14:33	17.950
row_3	3.333e+01	coconut	three	2015-03-15	15:45	2018-03-03 03:44	1.390
row_4	4.444e+02	durian	four	2015-04-15	16:50	2018-04-04 15:55	65100.000
min	0.11	—	—	—	—	—	1.39
max	444.40	—	—	—	—	—	65,100.00
grp_b							
row_5	5.550e+03	NA	five	2015-05-15	17:55	2018-05-05 04:00	1325.810
row_6	NA	fig	six	2015-06-15	NA	2018-06-06 16:11	13.255
row_7	7.770e+05	grapefruit	seven	NA	19:10	2018-07-07 05:22	NA

This is a source note.

The title of the table

The table's subtitle

row_8	8.880e+06	honeydew	eight	2015-08-15	20:20	NA	0.440
total	—	—	—	—	—	—	66,508.79

[1] This is a footnote.

This is a source note.

`opt_all_caps()`: Option to use all caps in select table locations

Sometimes an all-capitalized look is suitable for a table. With the `opt_all_caps()` function, we can transform characters in the column labels, the stub, and in all row groups in this way (and there's control over which of these locations are transformed).

EXAMPLE

Use `exibble` to create a **gt** table with a number of table parts added. All text in the column labels, the stub, and in all row groups is to be transformed to all caps using `opt_all_caps()`.

```
exibble %>%
  gt(rowname_col = "row", groupname_col = "group") %>%
  summary_rows(
    groups = "grp_a",
    columns = c(num, currency),
    fns = list(
      min = ~min(., na.rm = TRUE),
      max = ~max(., na.rm = TRUE)
    )) %>%
  grand_summary_rows(
    columns = currency,
    fns = list(total = ~sum(., na.rm = TRUE))
  ) %>%
  tab_source_note(source_note = "This is a source note.") %>%
  tab_footnote(
    footnote = "This is a footnote.",
    locations = cells_body(columns = 1, rows = 1)
  ) %>%
  tab_header(
    title = "The title of the table",
    subtitle = "The table's subtitle"
  ) %>%
  opt_all_caps()
```

The title of the table

The table's subtitle

	NUM	CHAR	FCTR	DATE	TIME	DATETIME	CURRENCY
GRP_A							

This is a source note.

The title of the table							
The table's subtitle							
ROW_1	1.111e-01[1]	apricot	one	2015-01-15	13:35	2018-01-01 02:22	49.950
ROW_2	2.222e+00	banana	two	2015-02-15	14:40	2018-02-02 14:33	17.950
ROW_3	3.333e+01	coconut	three	2015-03-15	15:45	2018-03-03 03:44	1.390
ROW_4	4.444e+02	durian	four	2015-04-15	16:50	2018-04-04 15:55	65100.000
min	0.11	—	—	—	—	—	1.39
max	444.40	—	—	—	—	—	65,100.00
GRP_B							
ROW_5	5.550e+03	NA	five	2015-05-15	17:55	2018-05-05 04:00	1325.810
ROW_6	NA	fig	six	2015-06-15	NA	2018-06-06 16:11	13.255
ROW_7	7.770e+05	grapefruit	seven	NA	19:10	2018-07-07 05:22	NA
ROW_8	8.880e+06	honeydew	eight	2015-08-15	20:20	NA	0.440
total	—	—	—	—	—	—	66,508.79

[1] This is a footnote.

This is a source note.

`opt_table_lines()`: Option to set table lines to different extents

The `opt_table_lines()` function sets table lines in one of three possible ways: (1) all possible table lines drawn (`"all"`), (2) no table lines at all (`"none"`), and (3) resetting to the default line styles (`"default"`). This is great if you want to start off with lots of lines and subtract just a few of them with `tab_options()` or `tab_style()` . Or, use it to start with a completely lineless table, adding individual lines as needed.

EXAMPLE

Use `exibble` to create a gt table with a number of table parts added. Then, use the `opt_table_lines()` function to haves lines everywhere there can possibly be lines.

```
exibble %>%
  gt(rowname_col = "row", groupname_col = "group") %>%
  summary_rows(
    groups = "grp_a",
    columns = c(num, currency),
    fns = list(
      min = ~min(., na.rm = TRUE),
      max = ~max(., na.rm = TRUE)
    )) %>%
  grand_summary_rows(
    columns = currency,
    fns = list(total = ~sum(., na.rm = TRUE))
  ) %>%
  tab_source_note(source_note = "This is a source note.") %>%
  tab_footnote(
    footnote = "This is a footnote.",
    locations = cells_body(columns = 1, rows = 1)
  ) %>%
  tab_header(
    title = "The title of the table",
    subtitle = "The table's subtitle"
  ) %>%
  opt_table_lines()
```

The title of the table

The table's subtitle

	num	char	fctr	date	time	datetime	currency
grp_a							
row_1	1.111e-01[1]	apricot	one	2015-01-15	13:35	2018-01-01 02:22	49.950
row_2	2.222e+00	banana	two	2015-02-15	14:40	2018-02-02 14:33	17.950
row_3	3.333e+01	coconut	three	2015-03-15	15:45	2018-03-03 03:44	1.390
row_4	4.444e+02	durian	four	2015-04-15	16:50	2018-04-04 15:55	65100.000
min	0.11	—	—	—	—	—	1.39
max	444.40	—	—	—	—	—	65,100.00
grp_b							
row_5	5.550e+03	NA	five	2015-05-15	17:55	2018-05-05 04:00	1325.810
row_6	NA	fig	six	2015-06-15	NA	2018-06-06 16:11	13.255
row_7	7.770e+05	grapefruit	seven	NA	19:10	2018-07-07 05:22	NA

This is a source note.

The title of the table The table's subtitle							
row_8	8.880e+06	honeydew	eight	2015-08-15	20:20	NA	0.440
total	—	—	—	—	—	—	66,508.79

[1] This is a footnote.

This is a source note.

`opt_table_outline()` : Option to wrap an outline around the entire table

This function puts an outline of consistent `style` , `width` , and `color` around the entire table. It'll write over any existing outside lines so long as the `width` is larger that of the existing lines. The default value of `style` (`"solid"`) will draw a solid outline, whereas a value of `"none"` will remove any present outline.

EXAMPLE

Use `exibble` to create a **gt** table with a number of table parts added. Have an outline wrap around the entire table by using `opt_table_outline()` .

```
exibble %>%
  gt(rowname_col = "row", groupname_col = "group") %>%
  summary_rows(
    groups = "grp_a",
    columns = c(num, currency),
    fns = list(
      min = ~min(., na.rm = TRUE),
      max = ~max(., na.rm = TRUE)
    )) %>%
  grand_summary_rows(
    columns = currency,
    fns = list(total = ~sum(., na.rm = TRUE))
  ) %>%
  tab_source_note(source_note = "This is a source note.") %>%
  tab_footnote(
    footnote = "This is a footnote.",
    locations = cells_body(columns = 1, rows = 1)
  ) %>%
  tab_header(
    title = "The title of the table",
    subtitle = "The table's subtitle"
  ) %>%
  opt_table_outline()
```

The title of the table The table's subtitle						
num	char	fctr	date	time	datetime	currency

This is a source note.

The title of the table The table's subtitle							
grp_a							
row_1	1.111e-01[1]	apricot	one	2015-01-15	13:35	2018-01-01 02:22	49.950
row_2	2.222e+00	banana	two	2015-02-15	14:40	2018-02-02 14:33	17.950
row_3	3.333e+01	coconut	three	2015-03-15	15:45	2018-03-03 03:44	1.390
row_4	4.444e+02	durian	four	2015-04-15	16:50	2018-04-04 15:55	65100.000
min	0.11	—	—	—	—	—	1.39
max	444.40	—	—	—	—	—	65,100.00
grp_b							
row_5	5.550e+03	NA	five	2015-05-15	17:55	2018-05-05 04:00	1325.810
row_6	NA	fig	six	2015-06-15	NA	2018-06-06 16:11	13.255
row_7	7.770e+05	grapefruit	seven	NA	19:10	2018-07-07 05:22	NA
row_8	8.880e+06	honeydew	eight	2015-08-15	20:20	NA	0.440
total	—	—	—	—	—	—	66,508.79

[1] This is a footnote.

This is a source note.

`opt_table_font()` : Option to define a custom font for the table

The `opt_table_font()` function makes it possible to define a custom font for the entire **gt** table. The standard fallback fonts are still set by default but the font defined here will take precedence. You could still have different fonts in select locations in the table, and for that you would need to use `tab_style()` in conjunction with the `cell_text()` helper function.

EXAMPLES

Use `sp500` to create a small **gt** table, using `fmt_currency()` to provide a dollar sign for the first row of monetary values; then, set a larger font size for the table and use the 'Merriweather' font (from Google Fonts, via `google_font()`) with two font fallbacks ('Cochin' and the catchall 'Serif' group).

```
sp500 %>%
  dplyr::slice(1:10) %>%
  dplyr::select(-volume, -adj_close) %>%
  gt() %>%
  fmt_currency(
    columns = 2:5,
    rows = 1,
    currency = "USD",
    use_seps = FALSE
  ) %>%
  tab_options(table.font.size = px(18)) %>%
  opt_table_font(
    font = list(
      google_font(name = "Merriweather"),
      "Cochin", "Serif"
    )
  )
```

date	open	high	low	close
2015-12-31	$2060.59	$2062.54	$2043.62	$2043.94
2015-12-30	2077.34	2077.34	2061.97	2063.36
2015-12-29	2060.54	2081.56	2060.54	2078.36
2015-12-28	2057.77	2057.77	2044.20	2056.50
2015-12-24	2063.52	2067.36	2058.73	2060.99
2015-12-23	2042.20	2064.73	2042.20	2064.29
2015-12-22	2023.15	2042.74	2020.49	2038.97
2015-12-21	2010.27	2022.90	2005.93	2021.15
2015-12-18	2040.81	2040.81	2005.33	2005.55
2015-12-17	2073.76	2076.37	2041.66	2041.89

Use `sza` to create an eleven-row table. Within `opt_table_font()`, set up a preferred list of sans-serif fonts that are commonly available in macOS and Windows (using part of the `default_fonts()` vector as a fallback).

```
sza %>%
  dplyr::filter(
    latitude == 20 &
      month == "jan" &
      !is.na(sza)
  ) %>%
  dplyr::select(-latitude, -month) %>%
  gt() %>%
  opt_table_font(
    font = c(
      "Helvetica Neue", "Segoe UI",
      default_fonts()[-c(1:3)]
    )
  ) %>%
  opt_all_caps()
```

TST	SZA
0700	84.9
0730	78.7
0800	72.7
0830	66.1
0900	61.5
0930	56.5
1000	52.1
1030	48.3
1100	45.5
1130	43.6
1200	43.0

`opt_css()`: Option to add custom CSS for the table

The `opt_css()` function makes it possible to add CSS to a **gt** table. This CSS will be added after the compiled CSS that **gt** generates automatically when the object is transformed to an HTML output table. You can supply `css` as a vector of lines or as a single string.

EXAMPLE

Use `exibble` to create a **gt** table and format the data in both columns. With `opt_css()`, insert CSS rulesets as a string and be sure to set the table ID explicitly (here as “one”).

```
exibble %>%
  dplyr::select(num, currency) %>%
  gt(id = "one") %>%
  fmt_currency(
    columns = currency,
    currency = "HKD"
  ) %>%
  fmt_scientific(columns = num) %>%
  opt_css(
    css = "
  #one .gt_table {
    background-color: skyblue;
  }
  #one .gt_row {
    padding: 20px 30px;
  }
  #one .gt_col_heading {
    text-align: center !important;
  }
  "
  )
```

num	currency
1.11×10^{-1}	HK$49.95
2.22	HK$17.95
3.33×10^{1}	HK$1.39
4.44×10^{2}	HK$65,100.00
5.55×10^{3}	HK$1,325.81
NA	HK$13.26
7.77×10^{5}	NA
8.88×10^{6}	HK$0.44