Package: ksTFL (via r-universe)

June 19, 2026

Type Package

Title Framework for Clinical Tables, Figures, and Listings

Version 0.11.7

Author Igor Aleschenkov [aut, cre, cph], Vladimir Larchenko [aut, cph]

Maintainer Igor Aleschenkov <igor.aleschenkov@gmail.com>

Description Library provides framework for generation of clinical TFLs. The purpose is to provide functionality to prepare metadata that describes TFL and pass this metadata together with data to the C++ rendering engine to generate final DOCX documents with styles and deterministic pagination. Bundled third-party components include vendored FreeType, HarfBuzz, minizip, and nlohmann/json code plus Liberation fonts under their respective compatible licenses; see LICENSE for redistribution details.

License GPL-3 + file LICENSE

URL https://crow16384.github.io/ksTFL/

BugReports https://github.com/crow16384/ksTFL/issues

Encoding UTF-8

Imports cli, checkmate, jsonlite, purrr, rlang, tidyselect, utils, digest, htmltools, rstudioapi, Rcpp (>= 1.0.0)

LinkingTo Rcpp

SystemRequirements C++20, GNU make

Suggests testthat (>= 3.0.0), knitr, rmarkdown, ggplot2, shiny, colourpicker, sortable, shinyFiles

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

RoxygenNote 8.0.0

Config/pak/sysreqs make

Repository https://crow16384.r-universe.dev

Date/Publication 2026-06-19 17:36:29 UTC

RemoteUrl https://github.com/crow16384/ksTFL

RemoteRef HEAD

RemoteSha eb4993a748bf811db55fbdb2a4a60162faa6f692

Contents

`add_body_text` *Add body text*

Description

Add body text (e.g., when no data to display)

Usage

```
add_body_text(
  spec = NULL,
  text = NULL,
  id = NULL,
  styleRef = NULL,
  order = NULL
)
```

Arguments

`spec`	Spec object (dispatches on class)
`text`	Character vector of body text lines
`id`	Body text identifier (auto-generated if NULL)

styleRef	Character vector of style names or result of f_combine(). Merged with last-win strategy.
order	Order of body text group (auto-assigned if NULL)

Details

Generic function to add body text. Dispatches to class-specific methods, allowing different spec classes to implement their own body text handling.

Multiple calls add multiple text groups. Calling with the same ID merges with last-win strategy.

Value

Updated spec object

Examples

```
## Not run:
# Text-only spec with body text
spec <- create_text() |>
  set_document(hasData = FALSE) |>
  add_body_text("No data available for the specified criteria.")

# Table spec with body text as fallback when no data rows
spec <- create_table(empty_df) |>
  set_document(hasData = FALSE) |>
  add_body_text("No adverse events were reported.",
                styleRef = f_combine("b", "fc_red"))

## End(Not run)
```

---

add_body_text.default *Default method for add_body_text*

Description

Default method for add_body_text

Usage

```
## Default S3 method:
add_body_text(spec, text = NULL, id = NULL, styleRef = NULL, order = NULL)
```

Arguments

spec	Spec object
text	Character vector of body text lines
id	Body text identifier

styleRef	Character vector of style names or result of f_combine(). Merged with last-win strategy.
order	Order of body text group

Value

Error if no method found

---

add_body_text.TFL_options

Add body text for TFL_options

Description

When adding body text to TFL options (global settings), automatically removes any existing default body text entries (__default_NNN) and starts adding new ones from __default_002 onwards.

Usage

```
## S3 method for class 'TFL_options'
add_body_text(spec, text = NULL, id = NULL, styleRef = NULL, order = NULL)
```

Arguments

spec	TFL_options object
text	Character vector of body text lines
id	Body text identifier (auto-generated as __default_NNN if NULL)
styleRef	Character vector of style names or result of f_combine(). Merged with last-win strategy.
order	Order of body text group (auto-assigned if NULL)

Value

Updated options object

add_body_text.TFL_spec *Add body text for TFL_spec*

Description

When adding body text to a spec that has default body text entries (IDs starting with *_default*), they are automatically removed to avoid mixing defaults with user-defined content.

Usage

```
## S3 method for class 'TFL_spec'
add_body_text(spec, text = NULL, id = NULL, styleRef = NULL, order = NULL)
```

Arguments

spec	TFL_spec object
text	Character vector of body text lines
id	Body text identifier (auto-generated if NULL)
styleRef	Character vector of style names or result of `f_combine()`. Merged with last-win strategy.
order	Order of body text group (auto-assigned if NULL)

Value

Updated spec object

add_footer *Add a footer row*

Description

Generic function to add footer rows. Dispatches to class-specific methods, allowing footers to be added to both spec objects and global options.

Usage

```
add_footer(spec = NULL, ..., level = NULL)
```

Arguments

spec	Spec object (dispatches on class)
...	Up to 3 character strings (left, center, right) • Positional parts represent left, center and right header/footer cells respectively. • Supply fewer than 3 parts if some cells should be empty; use empty string "" for explicit empties. • Each call appends one header/footer row; use the level parameter to replace an existing row.
level	Optional numeric index. If provided, replaces footer at that row. If NULL, appends next row.

Details

Each call adds a new footer row. Per schema, footers are arrays of arrays (each call = one row).

Value

Updated spec object

Examples

```
## Not run:
# Add to a spec object
spec <- create_text() |>
  add_footer("Company Name", "", "Page {PAGE} of {NUMPAGES}") |>
  add_footer("", "Confidential", "")

# Add to global options
options <- tfl_get_options()
options <- add_footer(options, "Company Name", "", "Page {PAGE} of {NUMPAGES}")

## End(Not run)
```

---

add_footer.default *Default method for add_footer*

Description

Default method for add_footer

Usage

```
## Default S3 method:
add_footer(spec, ...)
```

Arguments

spec	Spec object
...	Footer parts • Up to 3 positional character strings: left, center, right. • Use an empty string "" to represent an empty cell. • The level parameter can be used to replace an existing row instead of appending.

Value

Error if no method found

---

add_footer.TFL_options

Add a footer row for TFL_options

Description

Adds a footer row to the global TFL options, which can be used as defaults for all spec objects.

Usage

```
## S3 method for class 'TFL_options'
add_footer(spec, ..., level = NULL)
```

Arguments

spec	TFL_options object
...	Up to 3 character strings (left, center, right)
level	Optional numeric index. If provided, replaces footer at that row. If NULL, appends next row.

Value

Updated options object

add_footer.TFL_spec *Add a footer row for TFL_spec*

Description

Add a footer row for TFL_spec

Usage

```
## S3 method for class 'TFL_spec'
add_footer(spec, ..., level = NULL)
```

Arguments

spec	TFL_spec object
...	Up to 3 character strings (left, center, right) • Positional parts represent left, center and right header/footer cells respectively. • Supply fewer than 3 parts if some cells should be empty; use empty string "" for explicit empties. • Each call appends one header/footer row; use the level parameter to replace an existing row.
level	Optional numeric index. If provided, replaces footer at that row. If NULL, appends next row.

Value

Updated spec object

add_footnote *Add a footnote*

Description

Add a footnote to the specification. Multiple calls add multiple footnote groups. Calling with the same ID merges with last-win strategy.

Usage

```
add_footnote(spec, text, id = NULL, styleRef = NULL, order = NULL)
```

Arguments

spec	TFL spec object
text	Character vector of footnote text lines
id	Footnote identifier (auto-generated if NULL)
styleRef	Character vector of style names or result of f_combine(). Merged with last-win strategy.
order	Order of footnote group (auto-assigned if NULL)

Value

Updated spec object

Examples

```
## Not run:
spec <- create_table(mtcars) |>
  add_footnote("Data source: Clinical database lock 2025-12-01") |>
  add_footnote("Missing values displayed as 'N/A'",
               styleRef = f_combine("footnote_style", "font_courier_new"))

# Multiple footnote lines in one group
spec <- create_table(mtcars) |>
  add_footnote(c("a. Treatment group A", "b. Treatment group B"))

## End(Not run)
```

---

add_header *Add a header row*

Description

Generic function to add header rows. Dispatches to class-specific methods, allowing headers to be added to both spec objects and global options.

Usage

```
add_header(spec = NULL, ..., level = NULL)
```

Arguments

spec	Spec object (dispatches on class)
...	Up to 3 character strings (left, center, right) • Positional parts represent left, center and right header/footer cells respectively. • Supply fewer than 3 parts if some cells should be empty; use empty string "" for explicit empties.

- Each call appends one header/footer row; use the `level` parameter to replace an existing row.

`level` Optional numeric index. If provided, replaces header at that row. If NULL, appends next row.

Details

Each call adds a new header row. Per schema, headers are arrays of arrays (each call = one row).

Value

Updated spec object

Examples

```
## Not run:
# Add to a spec object
spec <- create_text() |>
  add_header("Study ABC-123", "CONFIDENTIAL", "Page {PAGE}") |>
  add_header("Protocol v2.0", "", paste("Date:", format(Sys.Date(), "%Y-%m-%d")))

# Add to global options
options <- tfl_get_options()
options <- add_header(options, "Study ABC-123", "CONFIDENTIAL", "Page {PAGE}")

## End(Not run)
```

`add_header.default` *Default method for add_header*

Description

Default method for add_header

Usage

```
## Default S3 method:
add_header(spec, ...)
```

Arguments

`spec` Spec object

`...` Header parts

- Up to 3 positional character strings: left, center, right.
- Use an empty string "" to represent an empty cell.
- The `level` parameter can be used to replace an existing row instead of appending.

Value

Error if no method found

`add_header.TFL_options` *Add a header row for TFL_options*

Description

Adds a header row to the global TFL options, which can be used as defaults for all spec objects.

Usage

```
## S3 method for class 'TFL_options'
add_header(spec, ..., level = NULL)
```

Arguments

`spec`	TFL_options object
`...`	Up to 3 character strings (left, center, right) • Positional parts represent left, center and right header/footer cells respectively. • Supply fewer than 3 parts if some cells should be empty; use empty string "" for explicit empties. • Each call appends one header/footer row; use the `level` parameter to replace an existing row.
`level`	Optional numeric index. If provided, replaces header at that row. If NULL, appends next row.

Value

Updated options object

`add_header.TFL_spec` *Add a header row for TFL_spec*

Description

Add a header row for TFL_spec

Usage

```
## S3 method for class 'TFL_spec'
add_header(spec, ..., level = NULL)
```

Arguments

spec	TFL_spec object
...	Up to 3 character strings (left, center, right) • Positional parts represent left, center and right header/footer cells respectively. • Supply fewer than 3 parts if some cells should be empty; use empty string "" for explicit empties. • Each call appends one header/footer row; use the level parameter to replace an existing row.
level	Optional numeric index. If provided, replaces header at that row. If NULL, appends next row.

Value

Updated spec object

---

add_span_header *Add stub (spanning) column definition*

Description

Define a spanning header that covers multiple columns. Can be used to create multi-level headers by specifying different stubOrder values.

Usage

```
add_span_header(
  spec,
  cols,
  label,
  stubOrder = NULL,
  id = NULL,
  labelStyleRef = NULL
)
```

Arguments

spec	TFL spec object
cols	Columns to span using tidyselect syntax. Accepts: • Named columns: c("age", "sex") • Column ranges: age:sex • Helper functions: starts_with("age_"), contains("_pct") • Negation: -id or !matches("^temp")
label	Spanning header label

stubOrder	Order of stub header (auto-generated if NULL). Used to create multi-level headers: lower numbers appear above higher numbers. Multiple stubs at the same order are allowed if their column sets do not overlap.
id	Stub column identifier (auto-generated if NULL)
labelStyleRef	List of style names to be applied. Provided styles will be merged with last-win strategy for report

Details

Column Overlap Rules:

Stubs at the **same** stubOrder cannot share columns (to avoid ambiguous headers). However, stubs at **different** stubOrder values can overlap freely.

This allows hierarchical header structures:

- stubOrder = 1: first-level grouping above the column headers
- stubOrder = 2: next-level grouping above the first-level etc..

Multiple calls with the same stubOrder are allowed as long as their column sets do not overlap. This enables building complex header structures incrementally.

Value

Updated spec object

Examples

```
## Not run:
data <- data.frame(
  id = 1:10,
  age = rnorm(10, 45, 10),
  sex = sample(c("M", "F"), 10, TRUE),
  weight = rnorm(10, 70, 10),
  height = rnorm(10, 170, 10)
)

# Example 1: Single-level spanning header
spec <- create_table(data) |>
  add_span_header(
    cols = c(age, sex),  # Using tidyselect (unquoted column names)
    label = "Demographics",
    labelStyleRef = c("stub_label_style", "bold")
  )

# Example 2: Two-level header hierarchy
spec <- create_table(data) |>
 add_span_header(cols = c(age, sex, weight, height), label = "All Measurements", stubOrder = 0) |>
  add_span_header(cols = c(age, sex), label = "Demographics", stubOrder = 1) |>
  add_span_header(cols = c(weight, height), label = "Physical", stubOrder = 1)

# Example 3: Three-level header hierarchy
spec <- create_table(data) |>
```

```
  add_span_header(cols = starts_with("a") | starts_with("w") | starts_with("h"),
                  label = "Main Data", stubOrder = 0) |>
  add_span_header(cols = c(age, sex), label = "Demographics", stubOrder = 1) |>
  add_span_header(cols = c(weight, height), label = "Physical", stubOrder = 1) |>
  add_span_header(cols = age, label = "Age Details", stubOrder = 2)

# Example 4: Using tidyselect helpers
spec <- create_table(data) |>
  add_span_header(cols = contains("age"), label = "Age-related", stubOrder = 0) |>
  add_span_header(cols = matches("^w"), label = "Weight", stubOrder = 0)

# Example 5: Negation to exclude columns
spec <- create_table(data) |>
  add_span_header(cols = -id, label = "Measurements", stubOrder = 0)

# Example 6: Multiple non-overlapping stubs at same level
spec <- create_table(data) |>
  add_span_header(cols = c(age, sex), label = "Group1", stubOrder = 1) |>
 add_span_header(cols = c(weight, height), label = "Group2", stubOrder = 1)  # OK: no overlap

## End(Not run)
```

`add_style` *Add or update a style definition*

Description

Generic function to add or update style definitions. Dispatches to class-specific methods, allowing different spec classes to implement their own style handling.

Usage

```
add_style(spec, id, ...)
```

Arguments

`spec`	Spec object (dispatches on class)
`id`	Style identifier (auto-generated if NULL)
`...`	Style modifiers created with s_* functions

- `s_font` — font properties.
- `s_paragraph` — paragraph-level formatting (may include nested `s_spacing` and `s_indents`).
- `s_table_style` — table-cell styling (may include nested `s_borders` / `s_border`).

Details

Define styling for various document elements. Multiple calls to the same modifier function will merge with last-win strategy.

Available modifiers inside this function:

- s_font - Font properties
- s_paragraph - Paragraph formatting
- s_table_style - Table cell styling

Value

Updated spec object

Examples

```
## Not run:
spec <- create_text() |>
  add_style(id = "header",
    s_font(font_name = "Arial", font_size = "14pt", bold = TRUE),
    s_paragraph(alignment = "center"),
    s_table_style(background_color = "#D9D9D9")
  ) |>
  # Multiple calls merge with last-win
  add_style(id = "header",
    s_font(color = "#FF0000")  # Adds color, keeps other font properties
  )

## End(Not run)
```

---

add_style.default *Default method for add_style*

Description

Default method for add_style

Usage

```
## Default S3 method:
add_style(spec, id = NULL, ...)
```

Arguments

spec	Spec object
id	Style identifier
...	Style modifiers • Functions created with s_*() helpers (e.g., s_font(), s_paragraph()). • These modifiers are evaluated in the add_style() context and merged into the style definition.

Value

Error if no method found

`add_style.TFL_options` *Add or update a style definition for TFL_options*

Description

Add or update a style definition for TFL_options

Usage

```
## S3 method for class 'TFL_options'
add_style(spec, id = NULL, ...)
```

Arguments

`spec`	TFL_options style branch object
`id`	Style identifier (name)
`...`	Style modifiers created with s_* functions • Allowed modifiers: `s_font()`, `s_paragraph()`, `s_table_style()`. • `s_paragraph()` may itself contain nested modifiers `s_spacing()` and `s_indents()`. • Modifiers are merged into the named style using a last-win strategy.

Value

Updated spec object

`add_style.TFL_spec` *Add or update a style definition for TFL_spec*

Description

Add or update a style definition for TFL_spec

Usage

```
## S3 method for class 'TFL_spec'
add_style(spec, id, ...)
```

Arguments

spec	TFL_spec object
id	Style identifier (auto-generated if NULL)
...	Style modifiers created with s_* functions • Allowed modifiers: s_font(), s_paragraph(), s_table_style(). • s_paragraph() may itself contain nested modifiers s_spacing() and s_indents(). • Modifiers are merged into the named style using a last-win strategy.

Value

Updated spec object

---

add_subtitle *Add a subtitle*

Description

Add a subtitle to the specification. Multiple calls add multiple subtitle groups. Calling with the same ID merges with last-win strategy.

Usage

```
add_subtitle(
  spec,
  text,
  id = NULL,
  styleRef = NULL,
  order = NULL,
  toclevel = NULL
)
```

Arguments

spec	TFL spec object.
text	Character vector of subtitle text lines. May contain #ByGroup1, #ByGroup2, ... placeholders that are replaced at render time with the current value of the first, second, ... grouping/paging column on each page.
id	Subtitle identifier (auto-generated if NULL).
styleRef	Character vector of style names to apply. Styles are merged with last-win strategy.
order	Integer ordering key (auto-assigned if NULL).

`toclevel` Optional integer 1–9. When set, this subtitle is marked as a Table of Contents entry at the given level.

Static subtitles (no `#ByGroupX` placeholders): the TC entry is emitted only on the **first page** of the spec, producing a single TOC entry.

Dynamic subtitles (containing `#ByGroupX`): a TC entry is emitted on **every page**, so each distinct group value gets its own TOC entry. The resolved (substituted) text is used as the TOC entry text.

In both cases, multi-line subtitles are concatenated with a space and inline styling tags are stripped for the TOC entry text. Use together with `add_title(toclevel = )` and `tfl_set_options(insertTOC = TRUE)` or `save_report(insertTOC = TRUE)`.

Value

Updated spec object.

Examples

```
## Not run:
# Static subtitle — one TOC entry for the whole report
spec <- create_table(adsl) |>
  add_title("Table 1: Demographics", toclevel = 1) |>
  add_subtitle("Safety Analysis Set", toclevel = 2) |>
  add_subtitle("Data Cutoff: 2025-12-14")

# Dynamic subtitle — one TOC entry per group value (e.g. one per visit)
spec <- create_table(advs) |>
  add_title("Table 2: Vital Signs by Visit and Parameter", toclevel = 1) |>
  add_subtitle("#ByGroup1 - #ByGroup2", toclevel = 2)

# Generate the TOC page
report <- create_report(spec)
save_report(report, docFileName = "tables.docx", insertTOC = TRUE)
# Open tables.docx in Word, click the TOC placeholder, press F9 to update.

## End(Not run)
```

---

add_title *Add a title*

Description

Add a title to the specification. Multiple calls add multiple title groups. Calling with the same ID merges with last-win strategy.

Usage

```
add_title(
  spec,
  text,
  id = NULL,
  styleRef = NULL,
  order = NULL,
  toclevel = NULL
)
```

Arguments

spec	TFL spec object.
text	Character vector of title text lines. Multiple elements are rendered as separate lines within the same title paragraph.
id	Title identifier (auto-generated if NULL).
styleRef	Character vector of style names to apply. Styles are merged with last-win strategy.
order	Integer ordering key (auto-assigned if NULL).
toclevel	Optional integer 1–9. When set, the **first page** occurrence of this title is marked as a Table of Contents entry at the given level. Multi-line titles are concatenated with a space for the TOC entry text; inline styling tags (e.g. <b>, <i>) are stripped automatically. To generate a TOC page, set toclevel here and either call tfl_set_options(insertTOC = TRUE) for the whole session or pass insertTOC = TRUE to save_report(). The renderer will prepend a "Table of Contents" page with a { TOC \f \h \z } field. Open the generated document in Word, click inside the TOC area, and press **F9** to populate it.

Value

Updated spec object.

Examples

```
## Not run:
# Basic multi-line title (no TOC)
spec <- create_table(adsl) |>
  add_title(c("Study ABC-123", "Table 1: Demographics")) |>
  add_title("Full Analysis Set", styleRef = "subtitle_style")

# Title marked for TOC at level 1 — renderer will emit a TC field on the first page
spec <- create_table(adsl) |>
  add_title("Table 1: Demographics", toclevel = 1)

# Full TOC workflow across a multi-spec report
t1 <- create_table(adsl) |>
  add_title("Table 1: Demographics", toclevel = 1) |>
```

```
  set_document(hasData = TRUE)

t2 <- create_table(advs) |>
  add_title("Table 2: Vital Signs", toclevel = 1) |>
  set_document(hasData = TRUE)

report <- create_report(t1, t2)
save_report(report, docFileName = "tables.docx", insertTOC = TRUE)
# Open tables.docx in Word, click the TOC placeholder, press F9 to update.

## End(Not run)
```

`c_addrow` *Insert Additional Row in Conditional Rows*

Description

Declares an additional row to be inserted above or below rows matching the parent `compute_cols()` condition. Content can optionally be copied from a specified column; if omitted, creates an empty separator row.

Usage

```
c_addrow(pos, value_from = NULL, styleRef = NULL)
```

Arguments

`pos`	Character. Position for insertion: "above" or "below".
`value_from`	Character or unquoted column name. Optional source column for the inserted row's content. If NULL or missing, creates an empty separator row.
`styleRef`	Character vector or result of `f_combine()`. Optional style to apply to the inserted row. If NULL, no special styling.

Details

Must be called inside `compute_cols()`.

Behavior:

- Multiple `c_addrow()` calls in one `compute_cols()` accumulate
- Order of appearance is preserved
- Coexists with other actions on same row
- If `value_from` is provided, must exist in spec columns (data_env reference)
- If `value_from` is NULL or missing, creates an empty separator row

Stackable Actions: Actions within a single `compute_cols()` call execute **sequentially in the order specified**. This means `c_addrow()` can see values modified by earlier `c_glue()` actions, allowing you to build compound cell values (e.g., "PARAM: VISIT") before using them in inserted rows. Multiple `c_glue()` and `c_addrow()` calls can be interleaved as needed.

Value

Quosure structure (internal use within `compute_cols()`)

See Also

`compute_cols()` for conditional row actions, `c_style()`, `c_merge()` for other action types

Examples

```
## Not run:
 compute_cols(spec, lastOf(treatment), c_addrow(pos = "below", value_from = "treatment"))
  compute_cols(spec, firstOf(visit), c_addrow(pos = "above"))

  # Stackable: addrow sees glued value
  compute_cols(spec, PARAM == "ALT",
    c_glue(PARAM, "after", glue_col = VISIT, separator = ": "),
    c_addrow("above", value_from = PARAM)  # Uses "ALT: Week 2"
  )

## End(Not run)
```

---

c_clear *Clear Cell Content in Conditional Rows*

Description

Declares a clear action that blanks the display text of specified cells in rows matching the parent `compute_cols()` condition. The cells are rendered empty without affecting their column structure, width, or styling.

Usage

```
c_clear(cols)
```

Arguments

`cols`	Tidyselect expression for the target columns to blank (e.g., `col1`, `c(col1, col2)`, `starts_with("x")`). Must resolve to visible (non-hidden) report columns only.

Details

Must be called inside `compute_cols()`.

Behavior:

- Sets the rendered cell text to `""` for matching rows.
- Processed before `c_merge()` and `c_glue()` in the rendering chain, so the cleared state participates in subsequent actions. In particular: combining `c_clear()` + `c_glue()` on the same column effectively *replaces* the original cell content with the glued value.

- Compatible with `c_style()`: styling is independent of text content.
- When `c_merge()` targets a cleared leader cell, the merged span renders as a blank merged cell.

Value

Quosure-style marker (internal use within `compute_cols()`)

See Also

`compute_cols()`, `c_style()`, `c_merge()`, `c_glue()`

Examples

```
## Not run:
  # Blank label column in total rows
  spec <- compute_cols(spec, is_total,
    c_clear(label))

  # Clear then replace with a value from another column (full replacement)
  spec <- compute_cols(spec, condition,
    c_clear(display_col),
    c_glue(display_col, "after", glue_col = replacement_col))

## End(Not run)
```

---

c_glue *Concatenate a Value to Cell Text in Conditional Rows*

Description

Declares a glue action to concatenate a value — from a data column or a literal string — to the display text of specified cells in rows matching the parent `compute_cols()` condition.

Usage

```
c_glue(cols, position, glue_col = NULL, text = NULL, separator = NULL)
```

Arguments

`cols`	Tidyselect expression for the target columns (e.g., `col1`, `c(col1, col2)`, `starts_with("x")`). Must resolve to visible (non-hidden) report columns only.
`position`	Character. Concatenation side: `"before"` prepends the glue value to the existing cell text; `"after"` appends it.
`glue_col`	Optional. Unquoted column name whose formatted value is concatenated onto the target cells. The column may be hidden (not in the visible column list). Mutually exclusive with `text`.

text	Optional. A single literal character string to concatenate onto the target cells. Mutually exclusive with glue_col.
separator	Character string inserted between the existing cell text and the glued value when both sides are non-empty. Defaults to "" (direct concatenation). When either side is empty, no separator is inserted.

Details

Must be called inside compute_cols().

Constraints:

- Exactly one of glue_col or text must be provided.
- cols resolves only visible report columns via tidyselect.
- glue_col can reference any data column, including hidden ones.
- glue_col must not overlap with cols.

Behavior:

- When the glue value (from glue_col or text) is empty or NA, the action is silently skipped for that row/cell.
- When a target cell was suppressed by deduplication (dedupe = TRUE on that column), the glue is silently skipped to preserve the visual suppression of repeated values.
- When a target cell is suppressed by a concurrent c_merge() action (i.e., it is a non-leader merged cell), the glue is silently skipped.
- The merge leader cell is glued normally when c_glue() targets a column involved in c_merge() as the first column.
- Multiple c_glue() calls on the same column accumulate in call order.

Interaction with other actions:

- c_style(): Fully compatible — styling and text modification are independent.
- c_merge(): Compatible. Glue is processed after merge in the renderer. Non-leader (suppressed) merge cells are skipped; the merge-leader cell is glued normally.
- c_addrow(): **Stackable** — when used together in the same compute_cols(), c_addrow() sees glued values. This allows building compound cell values (e.g., "PARAM: VISIT") before using them in inserted rows.
- c_pageBreak(): Fully compatible.

Stackable Actions: Actions within a single compute_cols() call execute **sequentially in the order specified**. This means c_glue() modifications are visible to subsequent c_addrow() actions in the same call, enabling complex multi-step transformations.

Value

Quosure-style marker (internal use within compute_cols())

See Also

compute_cols(), c_style(), c_merge(), c_addrow()

Examples

```
## Not run:
  # Append a unit from a hidden column (e.g. unit_col is not in spec cols)
  spec <- compute_cols(spec, !is.na(value),
    c_glue(value, "after", glue_col = unit, separator = " "))

  # Prepend a literal marker to a label column
  spec <- compute_cols(spec, is_total,
    c_glue(label, "before", text = ">> "))

  # Combine with c_style() — independent operations
  spec <- compute_cols(spec, firstOf(group),
    c_glue(label, "before", text = "> "),
    c_style(label, styleRef = "bold"))

  # Stackable with c_addrow() — addrow sees glued values
  spec <- compute_cols(spec, PARAM == "ALT",
    c_glue(PARAM, "after", glue_col = VISIT, separator = ": "),
    c_addrow("above", value_from = PARAM)  # Uses "ALT: Week 2"
  )

## End(Not run)
```

---

c_merge *Merge Adjacent Columns in Conditional Rows*

Description

Declares adjacent columns to be merged in rows matching the parent compute_cols() condition. Merged columns appear as a single spanned cell.

Usage

```
c_merge(cols, styleRef = NULL)
```

Arguments

cols	Tidyselect expression for column selection. Must resolve to at least 2 columns that are consecutive in the report column order.
styleRef	Character vector or result of f_combine(). Optional style to apply to the merged cell. If NULL, no special styling.

Details

Must be called inside `compute_cols()`. Columns must be adjacent in the final report column order.

Value

Quosure structure (internal use within `compute_cols()`)

See Also

`compute_cols()` for conditional row actions, `c_style()`, `c_addrow()` for other action types

Validation:

- Immediate: columns exist and are consecutive (error if not)
- Deferred: overlapping merge ranges from multiple `compute_cols()` calls (warning if resolvable, error if ambiguous)

Behavior:

- Multiple merge actions in one row: all applied if non-overlapping
- Overlapping merges from different `compute_cols()` blocks: raises warning/error

Examples

```
## Not run:
  spec <- create_table(mtcars) |>
    add_style("group_header", s_table_style(background_color = "#D9D9D9"))

  # Merge multiple columns for group header
  spec <- compute_cols(spec, group == "A",
    c_merge(c(col1, col2, col3), styleRef = "group_header"))

  # Merge without style
  spec <- compute_cols(spec, group == "B", c_merge(c(disp, hp)))

## End(Not run)
```

---

c_pageBreak *Insert a page break at the matching row*

Description

Used inside `compute_cols()` to signal the renderer to start a new page at every row matching the condition. Takes no arguments.

Usage

```
c_pageBreak()
```

Value

Action marker (internal use within `compute_cols()`)

See Also

`compute_cols()`, `c_style()`, `c_addrow()`, `c_merge()`

Examples

```
## Not run:
data <- data.frame(
  group = c("A", "A", "B", "B", "C"),
  value = c(1, 2, 3, 4, 5)
)

# Force a new page at the start of each group
spec <- create_table(data) |>
  compute_cols(firstOf(group), c_pageBreak())

## End(Not run)
```

---

c_style *Apply Style to Columns in Conditional Rows*

Description

Declares a style or combination of styles to be applied to specified columns in rows matching the parent `compute_cols()` condition.

Usage

```
c_style(cols, styleRef)
```

Arguments

`cols`	Tidyselect expression for column selection (e.g., `c(col1, col2)`, `everything()`, `starts_with("x")`)
`styleRef`	Character. Name of the style to apply (defined via `add_style()`). Can be a single style name (e.g., `"bold"`) or result of `f_combine()` for combining multiple styles (e.g., `f_combine("bold", "red")`). When multiple styles are provided via `f_combine()`, they are merged in the order listed (last wins for conflicting properties).

Details

Must be called inside `compute_cols()`. Columns are resolved using tidyselect syntax against the table data.

Value

Quosure structure (internal use within `compute_cols()`)

See Also

`compute_cols()` for conditional row actions, `c_merge()`, `c_addrow()` for other action types

Behavior:

- Same column styled multiple times in one row: last style wins, warning issued
- Same column styled from different `compute_cols()` calls on same row: automatic style combination (merged via `create_report()`)
- Multiple columns in one call: all receive the same style(s)

Style Combination:

- Use `f_combine("style1", "style2")` to apply multiple styles together
- During `create_report()`, combined styles are consolidated into a single hash
- Consolidation only happens for new specs (not pre-processed reports)

Examples

```
## Not run:
  spec <- create_table(mtcars) |>
    add_style("bold", s_font(bold = TRUE)) |>
    add_style("red", s_font(color = "red"))

  # Single style on one column
  spec <- compute_cols(spec, cyl == 6, c_style(mpg, styleRef = "bold"))

  # Single style on multiple columns
  spec <- compute_cols(spec, hp > 100, c_style(c(mpg, wt), styleRef = "red"))

  # Combined styles on columns
  spec <- compute_cols(spec, cyl == 8, c_style(mpg, styleRef = f_combine("bold", "red")))

## End(Not run)
```

---

clean_reports *Clean Obsolete and Orphaned JSON Files from a Meta Folder*

Description

Removes two categories of stale files from a meta folder:

1. **Obsolete spec JSONs** - older versions of a document when multiple spec JSONs exist for the same `doc_file`. Only the most-recent spec per document is kept.
2. **Orphaned data JSONs** - data JSON files that are no longer referenced by any surviving spec JSON.

Usage

```
clean_reports(
  meta_dir = tfl_get_option("meta_directory"),
  keep_versions = 1L,
  dry_run = TRUE
)
```

Arguments

meta_dir	Character string. Path to the meta folder. Defaults to tfl_get_option("meta_directory"). An error is raised when neither the argument nor the option is set.
keep_versions	Integer. Number of most-recent spec versions to keep per document. Default 1 (keep only the latest). Set to 2 to keep the latest and one previous version for rollback.
dry_run	Logical. If TRUE (default), only report what would be deleted without removing anything.

Details

By default the function runs in **dry-run** mode and only reports what would be deleted. Pass dry_run = FALSE to actually delete files.

Value

Invisibly returns a list with elements obsolete_specs, orphaned_data, and deleted (character vectors of filenames).

Examples

```
## Not run:
# Preview what would be removed
clean_reports("path/to/meta")

# Actually delete, keeping 1 version per document
clean_reports("path/to/meta", dry_run = FALSE)

# Keep 2 versions per document (latest + one rollback)
clean_reports("path/to/meta", keep_versions = 2, dry_run = FALSE)

## End(Not run)
```

---

compute_cols *Define Conditional Row Actions for Tables*

Description

Declares a set of styling, merging, and row insertion actions to be applied to rows matching a condition. Actions are captured as unevaluated expressions and evaluated later during create_report(). Supports complex conditions using data columns and helper functions.

Usage

```
compute_cols(spec, cond, ...)
```

Arguments

spec	A TFL_spec object (must have docType = "Table")
cond	An unquoted logical expression to be evaluated in the data environment. Can reference: • Data columns directly (e.g., col1 > 10, Parameter == "Pulse") • Helper functions (e.g., firstOf(), lastOf(), uniqueOf()) Must return a logical vector of length equal to nrow(data).
...	Action function calls: c_style(), c_merge(), c_addrow(), c_glue(), c_clear(). Multiple actions allowed, including duplicates. Actions are captured unevaluated.

Details

Execution Timeline:

1. compute_cols() captures condition and actions (no evaluation)
2. Appends to spec$.metadata$compute_cols list
3. During create_report(), conditions are evaluated and matched rows identified
4. Actions are applied to matching rows (styling, merging, row insertion)
5. StyleRows are serialized to JSON

Constraints:

- Only allowed for docType = "Table"
- Multiple compute_cols() calls accumulate on the same spec
- Actions on the same row from different compute_cols() blocks are aggregated
- Conditions must be deterministic (no NA values allowed)

Action Functions (used inside compute_cols()):

- c_style(cols, styleRef): Apply style(s) to columns in matching rows
- c_merge(cols, styleRef = NULL): Merge adjacent columns in matching rows
- c_addrow(pos, value_from = NULL, styleRef = NULL): Insert row above/below matching rows
- c_pageBreak(): Insert a page break at the matching row (no args)
- c_glue(cols, position, glue_col = NULL, text = NULL, separator = NULL): Concatenate a data column value or literal text to matching cell text
- c_clear(cols): Clear the display text of matching cells (render as blank)

Value

Modified spec object with appended action metadata in spec$.metadata$compute_cols. Invisibly returns the updated spec to enable piping workflows.

See Also

c_style(), c_merge(), c_addrow() for action functions used within compute_cols()

Examples

```
## Not run:
  spec <- create_table(mtcars)
  spec <- add_style(spec, id = "bold", s_font(bold = TRUE))
  spec <- add_style(spec, id = "red", s_font(color = "red"))
  spec <- add_style(spec, id = "highlight", s_table_style(background_color = "yellow"))

  # Style columns in rows where cyl is first occurrence
  spec <- compute_cols(spec, firstOf(cyl), c_style(c(mpg, hp), styleRef = "bold"))

  # Style and merge columns in rows with high hp
  spec <- compute_cols(spec, hp > 200,
    c_style(hp, styleRef = "red"),
    c_merge(c(wt, qsec), styleRef = "highlight")
  )

  # Add empty separator row above first occurrence
  spec <- compute_cols(spec, firstOf(cyl), c_addrow(pos = "above"))

  # Add row with content from a column
  spec <- compute_cols(spec, lastOf(cyl), c_addrow(pos = "below", value_from = "mpg"))

## End(Not run)
```

---

create_figure *Create a Figure Specification*

Description

Create and initialize a TFL specification for embedding a figure. Accepts either a **file path** to an existing image or a **ggplot2 object** that is rendered automatically to a temporary SVG file.

Usage

```
create_figure(plot_or_path, dpi = 300L)
```

Arguments

plot_or_path One of:

- A **character string** — path to an existing, readable image file (.png, .jpeg/.jpg, or .svg).

- A **ggplot2 object** (class "gg" or "ggplot") — the plot is rendered to a temporary file via ggplot2::ggsave(). Use dpi and package options (figureWidth, figureHeight, figureDevice) to control output dimensions and format.

dpi — Integer. Resolution (dots per inch) when plot_or_path is a ggplot2 object. Ignored for file paths. Default: 300.

Details

When a ggplot2 object is passed:

1. The plot is rendered via ggplot2::ggsave() to a temporary file in tempdir().
2. The temporary file path is stored in spec$.metadata$filePath.
3. save_report() copies the file (prefixed with dataRef) into metaPath, where the C++ renderer reads it.
4. The temporary file persists for the duration of the R session.

The C++ renderer natively supports .png, .jpeg/.jpg, and .svg formats.

Value

A TFL_spec object with docType = "Figure".

See Also

create_table(), create_text(), set_document()

Examples

```
## Not run:
## From file path (existing behaviour)
spec <- create_figure("inst/images/example.png")

## From a ggplot2 object
library(ggplot2)
p <- ggplot(mtcars, aes(x = wt, y = mpg)) + geom_point()
spec <- create_figure(p)

## Control figure defaults via options
tfl_set_options(figureWidth = "8in", figureHeight = "5in", figureDevice = "svg")
spec <- create_figure(p, dpi = 150)

## Override per figure
spec <- create_figure(p) |>
  set_document(figureDevice = "jpeg", figureScaleMode = "fitWidth")

## Full pipeline
spec <- create_figure(p) |>
  add_title("Weight vs MPG") |>
  add_footnote("Source: Motor Trend, 1974.")
report <- create_report(spec)
```

```
write_doc(report, name = "fig01", outDir = "output", metaPath = tempdir())

## End(Not run)
```

`create_report` *Combine Multiple TFL Specifications and/or Reports into a Single Report*

Description

This function takes multiple TFL specification objects and/or previously created TFL report objects and combines them into a single report object matching the spec_schema_v2 structure. Each spec is keyed by a combination of its variable name and metadata hash (for direct specs) or preserves original keys (for specs from reports).

Usage

```
create_report(...)
```

Arguments

`...`	One or more objects of class `TFL_spec` or `TFL_report`, or plain `lists` whose elements are `TFL_spec` / `TFL_report` objects. • `TFL_spec` objects produced by `create_table()`, `create_text()` or `create_figure()`. • `TFL_report` objects produced by previous calls to `create_report()`. • `list` arguments are expanded in place; for each `TFL_spec` element the list slot name (if any) is used as the key basis. Unnamed slots fall back to `<outer_arg_name>_<i>` (or `spec_<i>` when the outer argument is itself a literal `list(...)` call). Nested lists are not supported. • Arguments are processed in order; each new `TFL_spec` is keyed by the argument name combined with the spec metadata hash. • `TFL_report` objects are flattened and their spec keys are preserved.

Details

The function performs the following operations:

1. Flattens all inputs (expands `list` arguments and extracts specs from `TFL_report` objects).
2. Auto-disambiguates duplicate keys among newly-added specs by appending a numeric suffix (_2, _3, ...) to the name basis; duplicate keys originating from `TFL_report` arguments still trigger a hard error.
3. Consolidates styles within newly-added `TFL_spec` objects only (specs from `TFL_report` are already consolidated).
4. Assigns a global `docOrder` integer (1, 2, 3, ...) based on final position.
5. Preserves existing `dataRef` values and warns if duplicates detected.
6. Returns a named list keyed by `<variable_name>_<hash>` or original report keys.

Value

A named list where each element is a TFL_spec object, keyed by the pattern <variable_name>_<hash> for direct specs, or original keys for specs extracted from input reports. Result has class TFL_report.

Examples

```
## Not run:
spec1 <- create_table(mtcars)
spec2 <- create_text()
final_report <- create_report(spec1, spec2)

# Combining with a previous report
spec3 <- create_figure("path/to/image.png")
combined <- create_report(final_report, spec3)

# Passing a named list of specs
out <- list(t1 = spec1, t2 = spec2)
report_from_list <- create_report(out)

## End(Not run)
```

---

create_table *Create a Table Specification*

Description

Create and initialize a TFL specification for tabular output. This wrapper captures the tidyselect cols expression and forwards it to the internal initializer. Use create_table() when you have a data frame that should be rendered as a table.

Usage

```
create_table(data = NULL, cols = everything())
```

Arguments

data	A data frame to build the table from (required).
cols	Tidyselect expression indicating which columns from data to include in the report. Defaults to everything().

Details

Column Width Initialization: Initial column widths are automatically calculated based on data values and their types and sum to 100%. To lock specific columns and trigger automatic recalculation of others, use define_cols() with the colWidth parameter (when autoColWidth = TRUE in the tfl_options, the default).

Example workflow:

- Create table: widths auto-distributed
- `define_cols(id, colWidth="20%")`: locks id at 20%, others recalculated to fill 80% keeping initially detected proportions
- `define_cols(age, colWidth="2cm")`: locks age at fixed 2cm width, other relative columns recalculated to fill remaining space

Value

A `TFL_spec` object with `docType = "Table"`.

See Also

`create_text()`, `create_figure()`, `define_cols()`, `add_title()`, `create_report()`

Examples

```
## Not run:
## Basic usage with the built-in `mtcars` dataset
spec <- create_table(mtcars)

## Select specific columns using tidyselect
spec <- create_table(mtcars, cols = c(cyl, mpg, hp))

## or using ranges:
spec <- create_table(mtcars, cyl:hp)

## or by excluding columns
spec <- create_table(mtcars, cols = -c(gear, carb))

## or simple by names
spec <- create_table(mtcars, cols = c("cyl", "mpg", "hp"))

## End(Not run)
```

---

create_text *Create a Text Document Specification*

Description

Create and initialize a TFL specification for narrative (text) documents. This is a user-facing wrapper around the internal `.tfl_init()` initializer and provides a clear, intention-revealing name for creating text-only specifications. Text documents do not accept `data` and will have `docType = "Text"` set on the resulting spec.

Usage

```
create_text()
```

Value

A TFL_spec object with docType = "Text".

See Also

create_table(), create_figure(), add_body_text()

Examples

```
## Not run:
# Create a text-only spec and add narrative content
spec <- create_text() |>
  add_title("Listing of Adverse Events") |>
  set_document(hasData = FALSE) |>
  add_body_text("No adverse events were reported during the study.")

## End(Not run)
```

---

define_cols *Define or modify column properties*

Description

Modify properties of existing columns. Can modify single column or batch update multiple columns. All parameters support 1-to-many recycling: provide a single value to apply to all columns, or a vector matching the length of cols for one-to-one mapping. Multiple calls merge with last-win strategy.

Usage

```
define_cols(
  spec,
  cols,
  label = NULL,
  isID = NULL,
  isVisible = NULL,
  isGrouping = NULL,
  isPaging = NULL,
  labelStyleRef = NULL,
  isColBreak = NULL,
  dedupe = NULL,
  blankAfter = NULL,
  type = NULL,
  format = NULL,
  missings = NULL,
  colWidth = NULL,
  valueStyleRef = NULL
)
```

Arguments

spec	TFL spec object (must be initialized with create_table)
cols	Columns to modify using tidyselect syntax. Accepts: • Named columns: c("age", "group") • Column ranges: age:group • Helper functions: starts_with("age_"), contains("_pct") • Negation: -id or !matches("^temp")
label	Column label (length 1 or length of cols; NA skips that position)
isID	Whether column is identifier (length 1 or length of cols; NA skips that position)
isVisible	Whether column is visible in report output (length 1 or length of cols; NA skips that position). When set to FALSE, column is hidden from output and automatically assigned width "0.0cm". Invisible columns do NOT participate in width recalculation; only visible columns are included. Cannot set colWidth for invisible columns (error raised if attempted).
isGrouping	Whether column defines groups (length 1 or length of cols; NA skips that position)
isPaging	Whether column defines pages (length 1 or length of cols; NA skips that position)
labelStyleRef	List of style names to be applied. Provided styles will be merged with last-win strategy for report. Can be: single string (recycled), character vector from f_combine (recycled), or list of f_combine results or NA sentinels (one-to-one mapping to columns). Use NA as a list element to skip updating labelStyleRef for that column.
isColBreak	Whether column triggers page break (length 1 or length of cols; NA skips that position)
dedupe	Whether to deduplicate values (length 1 or length of cols; NA skips that position)
blankAfter	Whether to add blank after value change (length 1 or length of cols; NA skips that position)
type	Data type for column format: "string" or "numeric" (length 1 or length of cols; NA skips that position). Optional; omit to preserve existing.
format	Format string for numeric data (sprintf style), e.g. "%.1f" (length 1 or length of cols; NA skips that position). Optional.
missings	How to display missing values in columns (length 1 or length of cols; NA skips that position). Optional.
colWidth	Column width, e.g. "2in", "5cm", "20%" (length 1 or length of cols; NA skips that position — the column's width is left unchanged). Optional. When at least one non-NA value is specified, affected columns are marked as LOCKED and automatic recalculation of remaining unlocked columns is triggered if autoColWidth = TRUE in tfl_set_options(). Locked columns maintain their exact width while unlocked columns normalize to fill remaining available space.
valueStyleRef	Style names to apply to cell values. Provided styles will be merged with last-win strategy for report. Use NA as a list element to skip updating valueStyleRef for that column.

Details

Column Width Management:

Widths are managed through a LOCKED/UNLOCKED/VISIBLE partitioning system:

- VISIBLE columns: included in width calculations (isVisible != FALSE)
- LOCKED columns: exact width specified via colWidth parameter (any unit: %, cm, in, mm, pt)
- UNLOCKED columns: automatically recalculated to fill available space
- INVISIBLE columns: hidden from output (isVisible = FALSE), assigned width "0.0cm", excluded from calculations

When `colWidth` is specified (or visibility changes), columns are marked as LOCKED and remaining UNLOCKED columns are automatically recalculated (if `autoColWidth = TRUE`, the default):

- Only VISIBLE UNLOCKED columns participate in recalculation
- Locked columns (any unit) maintain their exact specified width
- Unlocked visible columns normalize proportionally to fill remaining available space
- All visible columns' widths sum to 100% with 1 decimal place precision
- Invisible columns stay at "0.0cm" and don't affect other widths

To disable auto-recalculation, use `tfl_set_options(autoColWidth = FALSE)`.

Invisible Column Behavior:

- `isVisible = FALSE` automatically sets `colWidth = "0.0cm"`
- Cannot set `colWidth` on invisible columns (raises error)
- Invisible columns are excluded from width recalculation entirely
- Shown with "hidden" flag in spec preview

Value

Updated spec object

Examples

```
## Not run:
data <- data.frame(id = 1:10, age = rnorm(10, 45, 10), group = rep(c("A", "B"), 5))

# Single column with format
spec <- create_table(data) |>
  define_cols("age",
    label = "Age (years)",
    type = "numeric", format = "%.1f", colWidth = "10%"
  )

# Batch update with single value
spec <- create_table(data) |>
  define_cols(c("id", "age", "group"),
    isVisible = TRUE  # Applied to all three
```

```
  )

# Batch update with mapped values
spec <- create_table(data) |>
  define_cols(c("id", "age"),
    label = c("Subject ID", "Age (years)"),  # One-to-one mapping
    isID = c(TRUE, FALSE)
  )

# Batch format update with mixed recycling
spec <- create_table(data) |>
  define_cols(c("id", "age"),
    type = "numeric",  # Single value recycled to both columns
    colWidth = c("10%", "15%")  # Different widths for each column
  )

# Multiple calls merge
spec <- create_table(data) |>
  define_cols("age",
    label = "Age",
    type = "numeric", format = "%.0f"
  ) |>
  define_cols("age",
    label = "Age (years)",  # Overrides previous label
    colWidth = "15%"  # Merges with format, keeping type and format
  )

# Apply style references - single value recycled to all columns
spec <- create_table(data) |>
  define_cols(c("age", "id"),
    labelStyleRef = f_combine("label_style", "emphasis")
  )

# Apply different styles combinations to different columns
spec <- create_table(data) |>
  define_cols(c("id", "age", "group"),
    labelStyleRef = c(
      f_combine("id_label", "key"),
      f_combine("numeric_label", "emphasis"),
      f_combine("categorical_label")
    )
  )

# Column width auto-recalculation (when autoColWidth = TRUE, the default):
# Initial widths are automatically distributed (e.g., id=33.3%, age=33.3%, group=33.4%)
spec <- create_table(data) |>
  define_cols("id", colWidth = "20%")  # Lock id at 20%
  # Result: id=20%, age and group auto-recalculate to fill remaining 80%
  #
# Multiple colWidth calls preserve previous locks:
spec <- create_table(data) |>
  define_cols("id", colWidth = "20%") |>
  define_cols("age", colWidth = "15%")
```

```
  # Result: id=20% (locked), age=15% (locked), group=65% (fills remaining)

# Disable auto-recalculation to manage widths manually:
tfl_set_options(autoColWidth = FALSE)  # Turn off auto-recalculation
spec <- create_table(data) |>
  define_cols(c("id", "age", "group"), colWidth = c("25%", "30%", "45%"))
  # Widths stay exactly as specified, no automatic recalculation
tfl_set_options(autoColWidth = TRUE)  # Re-enable (restore default)


# Using tidyselect helpers
spec <- create_table(data) |>
  define_cols(starts_with("age"),
    label = "Age-related metric",
    type = "numeric", format = "%.1f"
  )

# Using negation with tidyselect
spec <- create_table(data) |>
  define_cols(-id,  # Exclude id column
    isVisible = TRUE
  )

# Using column range
spec <- create_table(data) |>
  define_cols(age:group,  # All columns from age to group
    labelStyleRef = "emphasis"
  )

# Hide column from output while keeping data for conditional logic
spec <- create_table(data) |>
  define_cols("age",
    isVisible = FALSE  # Automatically sets width to "0.0cm"
  )
  # Result: age column hidden, other columns recalculated to sum to 100%

## End(Not run)
```

`f_combine` *Combine multiple style names for explicit grouping*

Description

Helper function to group multiple style names together for explicit application to columns or elements. Useful when using `define_cols` or other functions with multiple columns and you want to either:

- Recycle the same group of styles to all columns: `labelStyleRef = f_combine("style1", "style2")`
- Create explicit one-to-one mappings: `labelStyleRef = c(f_combine("s1", "s2"), f_combine("s3"), "style4")`

Usage

```
f_combine(...)
```

Arguments

`...`	Character strings representing style names to combine • Each argument must be a single character string naming a style. • Returned object has class `tfl_style_combine` to signal grouped style application.

Value

Character vector with class "tfl_style_combine" containing all provided style names. This special class signals to style resolution functions that these styles should be applied together as a group (merged with last-win strategy during `create_report()`).

Examples

```
## Not run:
# Combine styles for recycling to all columns
styles <- f_combine("label_style", "emphasis", "bold")

# Use in define_cols for one-to-one mapping
spec <- create_table(data) |>
  define_cols(c("id", "age"),
    labelStyleRef = c(
      f_combine("id_label", "key"),
      f_combine("numeric_label")
    )
  )

## End(Not run)
```

`list_reports` *List Saved Reports in a Meta Folder*

Description

Scans a meta folder produced by `save_report` and returns a summary data frame with one row per spec JSON file. Uses `_index.json` when available (fast); falls back to scanning every JSON file otherwise.

Usage

```
list_reports(
  meta_dir = tfl_get_option("meta_directory"),
  sort_by = c("datetime", "doc_file", "spec_file")
)
```

Arguments

meta_dir	Character string. Path to the meta folder. Defaults to tfl_get_option("meta_directory"). An error is raised when neither the argument nor the option is set.
sort_by	Character string. Column to sort by: "datetime" (default, newest first), "doc_file", or "spec_file".

Value

A data frame with columns:

spec_file Hash-named spec JSON filename.

doc_file Target DOCX filename stored at save time.

datetime ISO-8601 timestamp of when save_report() was called.

n_specs Number of TFL specs inside the JSON.

is_latest Logical - TRUE for the most-recent entry per doc_file; older entries are FALSE (obsolete candidates).

data_refs Character vector of data JSON base-names referenced by this spec (without .json extension).

When no spec JSONs are found in meta_dir, returns invisible(empty_data_frame) and prints an informational message.

Examples

```
## Not run:
df <- list_reports("path/to/meta")
print(df[df$is_latest, c("doc_file", "datetime", "spec_file")])

## End(Not run)
```

---

p_margins *Define page margins*

Description

This function can only be used inside p_page.

Usage

```
p_margins(
  top = NULL,
  bottom = NULL,
  left = NULL,
  right = NULL,
  header = NULL,
  footer = NULL
)
```

Arguments

top	Top margin, e.g. "25mm"
bottom	Bottom margin, e.g. "25mm"
left	Left margin, e.g. "20mm"
right	Right margin, e.g. "20mm"
header	Header margin, e.g. "12mm"
footer	Footer margin, e.g. "12mm"

Value

A margins specification object

Examples

```
## Not run:
spec <- create_text() |>
  set_page_style(
    page = p_page(
      size = "A4",
      orientation = "landscape",
      margins = p_margins(
        top = "25mm", bottom = "25mm",
        left = "20mm", right = "20mm",
        header = "12mm", footer = "12mm"
      )
    )
  )

## End(Not run)
```

---

p_page *Define page settings*

Description

This function can only be used inside set_page_style.

Usage

```
p_page(size = NULL, orientation = NULL, margins = NULL)
```

Arguments

size	Page size: "A4", "A3", "Letter", "Legal", "Executive"
orientation	Page orientation: "portrait" or "landscape"
margins	Margins object created with p_margins or a list with keys: top, bottom, left, right, header, footer

Value

A page specification object

Examples

```
## Not run:
spec <- create_text() |>
  set_page_style(
    docTemplate = "CRO Example_default",
    page = p_page(
      size = "A4",
      orientation = "landscape",
      margins = p_margins(
        top = "25mm", bottom = "25mm",
        left = "20mm", right = "20mm",
        header = "12mm", footer = "12mm"
      )
    )
  )

## End(Not run)
```

---

print.TFL_spec *Print method for TFL specification objects*

Description

Provides a comprehensive, human-readable overview of a TFL specification object with formatted tables, colors, and legends. Shows document metadata, columns with types and formats, titles, headers, footers, and sample data.

Usage

```
## S3 method for class 'TFL_spec'
print(
  x,
  layout = c("full", "compact"),
  width = getOption("width", 80L),
  colors = TRUE,
  ...
)
```

Arguments

x	A TFL_spec object (required). The specification to print.
layout	Display mode as a character string:

- "compact": Brief overview (default)

- `"full"`: Detailed view with all sections

`width`	Integer. Terminal width for text wrapping (default: from `getOption("width", 80L)`). Used to calculate column widths and text truncation.
`colors`	Logical. If `TRUE` (default), uses ANSI color codes for visual distinction (section headers in cyan/yellow/green, column names in blue). If `FALSE`, plain text.
`...`	Additional arguments passed to print methods (ignored).

Details

The full layout displays:

- Document type and data availability
- Summary counts (columns, titles, headers, footers, etc.)
- Page settings (size, orientation)
- Headers and footers content
- Titles, subtitles, and footnotes
- Column metadata table (Name, Label, Type, Format, Missings, Width, Flags, Styles)
- Flag legend explaining special column properties
- Body text content
- Sample data (first 3 rows)

Flags Legend:

- `*` = ID column (primary identifier)
- `x` = Hidden column (not visible in output)
- `#` = Grouping column (used for grouping rows)
- `>` = Page break (trigger page break)
- `v` = Column break (trigger column break)
- `d` = Deduplicate (remove duplicate values)
- `_` = Blank after (insert blank row after value change)

Value

The spec object (invisibly). This allows print to be called on a spec object for side effects while preserving the spec in piped workflows.

Examples

```
## Not run:
data <- data.frame(id = 1:10, age = rnorm(10, 45, 10), group = rep(c("A", "B"), 5))
spec <- create_table(data) |>
  add_title("Motor Trend Study") |>
  add_subtitle("Vehicle Performance Analysis") |>
  add_header("ABC Research", "Confidential", "2024") |>
  add_footnote("Source: mtcars dataset")
```

```
# Print with full details (calls print.TFL_spec)
print(spec)

# Compact view
print(spec, layout = "compact")

# Without colors
print(spec, colors = FALSE)

## End(Not run)
```

`replay_report` *Re-render a DOCX from Stored JSON*

Description

Re-renders a DOCX document entirely from JSON files stored in the meta folder - no R spec objects or data frames required. Useful for reproducing outputs after code changes or on a different machine.

Usage

```
replay_report(
  spec_json,
  meta_dir = tfl_get_option("meta_directory"),
  output_path = NULL,
  template_json = NULL,
  overrideTemplate = NULL,
  insertTOC = NULL,
  tocTitle = NULL,
  verbose = FALSE
)
```

Arguments

`spec_json`	Character string or character vector. Either: • A full path to a spec JSON file, or • A `doc_file` name (e.g. `"test_01.docx"`) - the most recent spec for that document is used. Multiple entries are allowed for merging several documents into one.
`meta_dir`	Character string or character vector. Path(s) to the meta folder(s). Defaults to `tfl_get_option("meta_directory")`. An error is raised when neither the argument nor the option is set. • A single string is recycled for every element of `spec_json`. • A vector of the same length as `spec_json` provides a per-document meta folder.

Required when any spec_json entry is a doc_file name rather than a full path.

output_path — Character string. Override the output DOCX path. If NULL (default), the path stored in the spec's _metadata (outDir/docFileName) is used. **Required** when length(spec_json) > 1.

template_json — Character string. Override the template JSON path. If NULL, resolved automatically from the spec.

overrideTemplate — Character string. A bundled template name (e.g. "Navy_Pro") or file path to a custom styles JSON. When non-NULL, this takes precedence over template_json. See tfl_list_templates() for available names.

insertTOC — Logical. Insert a Table of Contents. NULL (default) inherits the value from the first document's metadata.

tocTitle — Character string. TOC heading text. NULL (default) inherits from the first document.

verbose — Logical. Print C++ pipeline diagnostics. Default FALSE.

Details

When a single spec_json is provided the function behaves exactly as before. When a character vector of length > 1 is given, the specs from every document are merged into one combined JSON and rendered into a single DOCX file. output_path is required in this case.

Value

Invisibly returns the path to the rendered DOCX file.

Examples

```
## Not run:
# By doc name (uses latest spec)
replay_report("test_01.docx", meta_dir = "path/to/meta")

# By spec hash (exact version)
replay_report("abc123def456.json", meta_dir = "path/to/meta")

# Override output location
replay_report("test_01.docx", meta_dir = "path/to/meta",
              output_path = "~/Desktop/test_01_replay.docx")

# Merge two documents from the same meta folder
replay_report(
  c("tables_01.docx", "listings_01.docx"),
  meta_dir    = "path/to/meta",
  output_path = "output/combined.docx"
)

# Merge documents from different meta folders
replay_report(
  c("path/to/meta_a/abc123.json", "path/to/meta_b/def456.json"),
```

```
  output_path = "output/combined.docx"
)

## End(Not run)
```

`run_replay_app` *Launch the Combined Replay Shiny App*

Description

Opens an interactive Shiny application for selecting, reordering, and replaying multiple saved reports into a single combined DOCX document. Reports can be loaded from one or more meta folders, reordered via drag-and-drop, and rendered with optional TOC settings.

Usage

```
run_replay_app(meta_dir = NULL, ...)
```

Arguments

`meta_dir`	Character string (optional). Default meta folder path to pre-populate in the app. If `NULL`, the user must enter a path manually.
`...`	Additional arguments passed to `shiny::runApp()`, such as `launch.browser = TRUE`.

Details

This function requires the `shiny`, `sortable`, and `shinyFiles` packages.

Value

Invisibly returns the result of `shiny::runApp()`.

See Also

`list_reports()`, `replay_report()`, `clean_reports()`

Examples

```
## Not run:
run_replay_app()
run_replay_app(meta_dir = "path/to/meta")

## End(Not run)
```

`run_styles_editor` *Launch the styles template editor Shiny app*

Description

Opens an interactive Shiny application for creating and editing ksTFL styles templates that conform to `styles_schema_v2.json`. Templates can be loaded from the bundled `inst/templates/` directory or uploaded from disk, then edited and downloaded as JSON for use with `set_page_style()` / `write_doc()`.

Usage

```
run_styles_editor(...)
```

Arguments

`...`	Additional arguments passed to `shiny::runApp()`, such as `launch.browser = TRUE` or `port = 4321`.

Details

This function requires the **shiny** package to be installed.

Value

Invisibly returns the result of `shiny::runApp()`.

See Also

`tfl_list_templates()`, `set_page_style()`

Examples

```
## Not run:
run_styles_editor()
run_styles_editor(launch.browser = TRUE)

## End(Not run)
```

s_border *Define border properties*

Description

This function can only be used inside s_borders.

Usage

```
s_border(color = NULL, width = NULL, line_style = NULL)
```

Arguments

color	Border color as hex (e.g., "#000000") or color name (e.g., "black", "red")
width	Border width, e.g. "1pt"
line_style	Line style: "single", "double", "dashed", "dotted", "thick", "none"

Value

A border specification object

Examples

```
## Not run:
# Cell-level borders (inside s_table_style)
spec <- create_text() |>
  add_style("my_style",
    s_table_style(
      borders = s_borders(
        bottom = s_border(color = "#000000", width = "2pt", line_style = "single")
      )
    )
  )

# Paragraph-level borders (inside s_paragraph)
spec <- create_table(mtcars) |>
  add_style("para_underline",
    s_paragraph(
      borders = s_borders(
        bottom = s_border(width = "1pt")
      )
    )
  )

## End(Not run)
```

s_borders *Define borders for table cells or paragraphs*

Description

This function can be used inside s_table_style (cell-level borders) or s_paragraph (paragraph-level borders).

Usage

```
s_borders(top = NULL, bottom = NULL, left = NULL, right = NULL)
```

Arguments

top	Top border created with s_border
bottom	Bottom border created with s_border
left	Left border created with s_border
right	Right border created with s_border

Value

A borders specification object

Examples

```
## Not run:
# Cell-level borders
spec <- create_text() |>
  add_style("my_style",
    s_table_style(
      borders = s_borders(
        top = s_border(width = "1pt"),
        bottom = s_border(width = "2pt", line_style = "double")
      )
    )
  )

# Paragraph-level borders (border follows the text, not the cell edge)
spec <- create_table(mtcars) |>
  add_style("span_underline",
    s_paragraph(
      alignment = "center",
      borders = s_borders(
        bottom = s_border(color = "#4472C4", width = "1pt")
      )
    )
  )

## End(Not run)
```

s_font *Define font properties for a style*

Description

This function can only be used inside add_style.

Usage

```
s_font(
  font_name = NULL,
  font_size = NULL,
  bold = NULL,
  italic = NULL,
  underline = NULL,
  strikethrough = NULL,
  color = NULL,
  highlight = NULL
)
```

Arguments

font_name	Font family name. One of: "Arial", "Courier New", "Times New Roman", "Georgia", "Verdana", "Trebuchet MS", "Liberation Sans"
font_size	Font size with units, e.g. "12pt"
bold	Logical, whether text is bold
italic	Logical, whether text is italic
underline	Logical, whether text is underlined
strikethrough	Logical, whether text has strikethrough
color	Text color as hex (e.g., "#000000") or color name (e.g., "red", "blue")
highlight	Background highlight color as hex (e.g., "#FFFF00") or color name (e.g., "yellow")

Value

A font specification object (for internal use)

See Also

add_style() for applying styles, s_paragraph(), s_table_style() for other style components

Examples

```
## Not run:
spec <- create_text() |>
  add_style("my_style",
    s_font(font_name = "Arial", font_size = "12pt", bold = TRUE)
  )

## End(Not run)
```

`s_indents` *Define indentation properties for paragraphs*

Description

This function can only be used inside `s_paragraph`.

Usage

```
s_indents(left = NULL, right = NULL, first_line = NULL)
```

Arguments

`left`	Left indent, e.g. "10mm"
`right`	Right indent, e.g. "10mm"
`first_line`	First line indent (negative for hanging), e.g. "-5mm"

Value

An indents specification object

Examples

```
## Not run:
spec <- create_text() |>
  add_style("my_style",
    s_paragraph(
      indents = s_indents(left = "10mm", first_line = "-5mm")
    )
  )

## End(Not run)
```

s_paragraph *Define paragraph properties for a style*

Description

This function can only be used inside add_style.

Usage

```
s_paragraph(
  alignment = NULL,
  spacing = NULL,
  indents = NULL,
  word_style = NULL,
  borders = NULL
)
```

Arguments

alignment	Text alignment: "left", "right", "center", "justify", "distributed"
spacing	Spacing object created with s_spacing or a list with keys: before, after, line_spacing
indents	Indents object created with s_indents or a list with keys: left, right, first_line
word_style	Base Word style to inherit from
borders	Borders object created with s_borders. Applied as paragraph-level borders (<w:pBdr> in OOXML), distinct from cell-level borders set via s_table_style.

Value

A paragraph specification object

See Also

add_style() for applying styles, s_spacing(), s_indents(), s_borders() for nested components, s_font(), s_table_style() for other style components

Examples

```
## Not run:
spec <- create_text() |>
  add_style("my_style",
    s_paragraph(
      alignment = "center",
      spacing = s_spacing(before = "12pt", after = "6pt"),
      word_style = "Normal"
    )
  )
```

```
# Paragraph with a bottom border (applied to the text, not the cell)
spec <- create_table(mtcars) |>
  add_style("para_border",
    s_paragraph(
      alignment = "center",
      borders = s_borders(
        bottom = s_border(color = "#000000", width = "0.5pt", line_style = "single")
      )
    )
  )

## End(Not run)
```

`s_spacing` *Define spacing properties for paragraphs*

Description

This function can only be used inside `s_paragraph`.

Usage

```
s_spacing(before = NULL, after = NULL, line_spacing = NULL)
```

Arguments

`before`	Space before paragraph, e.g. "6pt"
`after`	Space after paragraph, e.g. "6pt"
`line_spacing`	Line spacing multiplier, minimum 1

Value

A spacing specification object

Examples

```
## Not run:
spec <- create_text() |>
  add_style("my_style",
    s_paragraph(
      alignment = "center",
      spacing = s_spacing(before = "12pt", after = "6pt")
    )
  )

## End(Not run)
```

`s_table_style` *Define table-specific styling*

Description

This function can only be used inside `add_style`.

Usage

```
s_table_style(
  background_color = NULL,
  row_height = NULL,
  topEmptyLine = NULL,
  bottomEmptyLine = NULL,
  vertical_alignment = NULL,
  text_orientation = NULL,
  borders = NULL
)
```

Arguments

`background_color`	Cell background color as hex code or color name
`row_height`	Row height, e.g. "15mm" or "auto"
`topEmptyLine`	Empty spacer row height after header, e.g. "6pt"; NULL disables it
`bottomEmptyLine`	Empty spacer row height before bottom border, e.g. "6pt"; NULL disables it
`vertical_alignment`	Vertical alignment: "top", "center", "bottom"
`text_orientation`	Text orientation: "horizontal", "vertical_90", "vertical_270"
`borders`	Borders object created with `s_borders`

Value

A table style specification object

See Also

`add_style()` for applying styles, `s_borders()`, `s_border()` for border components, `s_font()`, `s_paragraph()` for other style components

Examples

```
## Not run:
spec <- create_text() |>
  add_style("header_style",
    s_table_style(
      background_color = "#D9D9D9",
      vertical_alignment = "center",
      borders = s_borders(
        bottom = s_border(color = "#000000", width = "2pt")
      )
    )
  )

## End(Not run)
```

---

save_report *Save TFL Report to JSON with Data Files*

Description

Serializes a TFL_report object to JSON format along with associated data files (for tables) or figure files (for figures). Validates the report structure, processes each specification by its docType, and writes all outputs to disk.

Usage

```
save_report(
  report,
  docFileName,
  outDir = NULL,
  metaPath = NULL,
  prettify = FALSE,
  insertTOC = NULL,
  tocTitle = NULL
)
```

Arguments

report	A TFL_report object (output from create_report())
docFileName	Character string. Name of the rendered document file that will be created by the C++ renderer (e.g., "report.docx"). This value is stored in the _metadata/docFileName property of the exported JSON spec.
outDir	Character string. Output directory where the C++ renderer will save the rendered document. This path is stored in the _metadata/outDir property. If not provided, defaults to tfl_get_option("output_directory").

metaPath	Character string. Directory where this function will save the JSON spec and associated data files. If not provided, a temporary directory is created. Default: tempdir()
prettify	Logical. If TRUE, format JSON output with indentation and line breaks for readability. Default: FALSE (compact format)
insertTOC	Logical. When TRUE the renderer prepends a Table of Contents page (using a { TOC \f \h \z } field) before the first spec. Requires at least one add_title() or add_subtitle() call with toclevel set. Defaults to the insertTOC package option (see tfl_set_options()). Default FALSE.
tocTitle	Character. Heading text placed above the TOC field on the TOC page. Defaults to the tocTitle package option (default "Table of Contents"). Set to "" to omit the heading.

Details

The function performs the following steps:

1. Validates input is a TFL_report object
2. Serializes the report using serialize_spec() to validate against schema
3. Processes each specification by docType:
4. Text: Drops .metadata (no additional files needed)
5. Table: Extracts data, filters to included columns, saves as JSON
6. Figure: Copies image file with original extension preserved
7. Creates _metadata section with outDir and docFileName
8. Serializes the complete fixed object to JSON
9. Returns information about saved files

Error Conditions:

- Report class is not TFL_report
- Table/Figure spec has hasData=TRUE but no actual data available
- dataRef has multiple entries (currently only single file references supported)
- File I/O errors when writing JSON or copying files

Value

Invisibly returns a list with:

- spec_file: Name of the saved spec JSON file (hash-based filename)
- datetime: Timestamp when the report was saved (ISO 8601 format)
- metaPath: Directory where files were saved

Examples

```
## Not run:
spec1 <- create_table(mtcars)
spec2 <- create_text()
report <- create_report(spec1, spec2)

# Save with explicit parameters
result <- save_report(
  report,
  docFileName = "report.docx",
  outDir = "/output/path",
  metaPath = "/meta/path"
)

# Save with defaults (uses tempdir and tfl_options)
result <- save_report(report, docFileName = "report.docx")

cat("Spec saved as:", result$spec_file, "\n")
cat("Saved to:", result$metaPath, "\n")

# Generate a report with an auto-populated TOC page
t1 <- create_table(adsl) |>
  add_title("Table 1: Demographics", toclevel = 1) |>
  set_document(hasData = TRUE)

t2 <- create_table(advs) |>
  add_title("Table 2: Vital Signs by Visit", toclevel = 1) |>
  add_subtitle("#ByGroup1", toclevel = 2) |>
  set_document(hasData = TRUE)

report <- create_report(t1, t2)
save_report(
  report,
  docFileName  = "tables.docx",
  outDir       = "output/",
  insertTOC    = TRUE,
  tocTitle     = "List of Tables"
)
# Open tables.docx in Word, click the TOC placeholder, press F9 to populate it.

## End(Not run)
```

`set_document` *Set document properties*

Description

Define document-level properties. Multiple calls merge with last-win strategy. Document type (`docType`) is set automatically by `create_table()`, `create_figure()`, or `create_text()` and cannot be changed here. Global document order (`docOrder`) is assigned by `create_report()`.

Usage

```
set_document(
  spec,
  isContinues = NULL,
  continuousSection = NULL,
  contentWidth = NULL,
  footnotePlace = NULL,
  hasData = NULL,
  topEmptyLine = NULL,
  bottomEmptyLine = NULL,
  docTemplate = NULL,
  figureWidth = NULL,
  figureHeight = NULL,
  figureDevice = NULL,
  figureScaleMode = NULL
)
```

Arguments

spec	TFL spec object
isContinues	Logical. When TRUE, titles and subtitles are shown only on the first page of a multi-page spec instead of repeating on every page. Default FALSE (titles repeat on each page).
continuousSection	Logical. When TRUE on a spec that is not the first in a report, the page break before this spec is suppressed and it continues on the same page as the previous spec. Requires matching page size and margins with the previous spec; if they differ, Word may still force a page break. The paginator is not adjusted — Word handles natural overflow when content exceeds the remaining page space. Works best for short content (figures, text, small tables). Default FALSE.
contentWidth	Width of content, e.g. "100%", "25cm", "10in".
footnotePlace	Character; controls where footnotes are rendered. One of "doc_footer" (place inside the Word footer, below footer rows), "repeated" (place under the table on every page), or "last_page" (place under the table on the last page only). Default "repeated".
hasData	Logical. Whether this spec has data to render. Set to TRUE for tables with data rows. When FALSE, the body text (if any) is rendered instead.
topEmptyLine	Empty spacer row height after table header (table-level), e.g. "6pt". Use NULL to disable. "0pt" is treated as no spacer row.
bottomEmptyLine	Empty spacer row height before table bottom border (table-level), e.g. "6pt". Use NULL to disable.
docTemplate	Character. Template to use for rendering. Accepts either: • Name of a bundled template (see tfl_list_templates()). • Full path to a custom styles JSON file.

figureWidth	Figure width with units, e.g. "6in", "70%", "16.51cm". Only relevant for docType = "Figure".
figureHeight	Figure height with units. Same syntax as figureWidth.
figureDevice	Character. Image format for ggplot2 rendering. One of "svg", "png", or "jpeg".
figureScaleMode	Character. How the figure is scaled in the DOCX. One of "fixed" (exact dimensions) or "fitWidth" (scale to page width, preserving aspect ratio).

Value

Updated spec object

Examples

```
## Not run:
# Table spec with data
spec <- create_table(mtcars) |>
  set_document(hasData = TRUE)

# Text spec for narrative-only output
spec <- create_text() |>
  set_document(hasData = FALSE) |>
  add_body_text("No adverse events were reported.")

# Figure spec with custom sizing
spec <- create_figure("plot.png") |>
  set_document(
    figureWidth  = "7in",
    figureHeight = "5in",
    figureScaleMode = "fitWidth"
  )

# Footnotes placed on last page only
spec <- create_table(mtcars) |>
  set_document(hasData = TRUE, footnotePlace = "last_page") |>
  add_footnote("Source: Motor Trend, 1974.")

# Use a bundled template (see tfl_list_templates() for available names)
spec <- create_table(mtcars) |>
  set_document(hasData = TRUE, docTemplate = "Navy_Pro")

## End(Not run)
```

---

set_page_style *Set document style properties*

Description

Set document-level style configuration. Multiple calls merge with last-win strategy.

Usage

```
set_page_style(spec, docTemplate = NULL, page = NULL)

## S3 method for class 'TFL_spec'
set_page_style(spec, docTemplate = NULL, page = NULL)

## S3 method for class 'TFL_options'
set_page_style(spec, docTemplate = NULL, page = NULL)
```

Arguments

spec	TFL spec object
docTemplate	Character. Template to use for rendering. Accepts either: • A predefined bundled template name (e.g. "CRO Example_default", "Navy_Pro"). Use tfl_list_templates() to see all available names. • A file path (absolute or relative) to an external template JSON file. The path must point to an existing file conforming to styles_schema_v2.json. When NULL (default) the current session template is used.
page	Page settings object created with p_page or a list with keys: size, orientation, margins

Value

Updated spec object

Examples

```
## Not run:
# Use a predefined bundled template
spec <- create_text() |>
  set_page_style(
    docTemplate = "Navy_Pro",
    page = p_page(size = "A4", orientation = "landscape")
  )

# Use an external template file
spec <- create_text() |>
  set_page_style(docTemplate = "/path/to/my_template.json")

## End(Not run)
```

tfl_font_status *Show current font status*

Description

Prints the font resolution report from the most recent scan without re-scanning. Use `tfl_rescan_fonts()` to perform a fresh scan.

Usage

```
tfl_font_status()
```

Value

Invisibly returns the cached font scan report.

See Also

`tfl_rescan_fonts()` to re-run the scan.

Examples

```
## Not run:
# Check which fonts are resolved and which use fallbacks
tfl_font_status()

## End(Not run)
```

tfl_get_option *Retrieve a single package option*

Description

Fetch a single named option from the active ksTFL settings. This is a convenience wrapper around `tfl_get_options()` that returns one element or throws a friendly error if the option does not exist.

Usage

```
tfl_get_option(name)
```

Arguments

name	Character(1). Name of the option to fetch. Common options: `"page"`, `"styles"`, `"footnotePlace"`, `"isContinues"`, `"contentWidth"`, `"missings"`, `"autoColWidth"`, `"minColWidth"`, `"insertTOC"`, `"tocTitle"`, `"output_directory"`, `"meta_directory"`.

Value

The value associated with `name` (type depends on the option).

Examples

```
## Not run:
# Get the current page settings
tfl_get_option("page")

# Check whether TOC generation is enabled
tfl_get_option("insertTOC")
tfl_get_option("tocTitle")

## End(Not run)
```

---

tfl_get_options *Return the active package options*

Description

Returns the current session settings for ksTFL as a named list. These are the effective values used when building specs and rendering documents.

Usage

```
tfl_get_options()
```

Details

The returned object is the internal settings list stored in the package environment. Modifying the returned object will not change package state; use `tfl_set_options()` to update settings for the current session.

Value

A named list representing the current ksTFL options.

Examples

```
## Not run:
# Inspect all current settings
tfl_get_options()

## End(Not run)
```

tfl_list_templates *List available bundled templates*

Description

Returns the names of all template JSON files bundled with the package in `inst/templates/`. These names can be passed directly to `set_page_style(docTemplate = ...)` or `tfl_set_options(docTemplate = ...)`.

Usage

```
tfl_list_templates()
```

Value

A character vector of template names (without the `.json` extension), sorted alphabetically.

Examples

```
## Not run:
tfl_list_templates()

## End(Not run)
```

tfl_print_style_atoms *Print all built-in style atoms to the console*

Description

Iterates over every atom in the internal `.const_options_styles` registry and prints a coloured, grouped summary using **cli**. `tfl_style_atoms_catalog()` is a convenience alias for `tfl_print_style_atoms()`.

Usage

```
tfl_print_style_atoms()

tfl_style_atoms_catalog()
```

Value

Invisible `NULL`.

Examples

```
## Not run:
# Print the full catalog of built-in style atoms
tfl_print_style_atoms()

# Same output via the alias
tfl_style_atoms_catalog()

## End(Not run)
```

`tfl_rescan_fonts` *Rescan system fonts*

Description

Re-runs the font discovery process using the current value of `getOption("ksTFL.font_dirs")`. This is useful after installing new fonts or after changing the `ksTFL.font_dirs` option.

Usage

```
tfl_rescan_fonts()
```

Details

To point ksTFL at a custom TTF folder, set the `ksTFL.font_dirs` option before or during your session, then call `tfl_rescan_fonts()`:

```
options(ksTFL.font_dirs = "/path/to/your/fonts")
tfl_rescan_fonts()
```

Multiple directories are supported:

```
options(ksTFL.font_dirs = c("/path/to/fonts1", "/path/to/fonts2"))
tfl_rescan_fonts()
```

The package's own bundled fonts directory is always scanned automatically; `ksTFL.font_dirs` only adds extra directories on top of that. To make the setting persistent across sessions, add the `options()` call to your '`~/.Rprofile`'.

Value

Invisibly returns the font scan report (a list with `resolutions` and `dirs_scanned`).

See Also

`tfl_font_status()` to print the cached report without rescanning.

Examples

```
## Not run:
# Rescan after installing new fonts
tfl_rescan_fonts()

# Point to a custom font directory, then rescan
options(ksTFL.font_dirs = c("/usr/share/fonts/custom"))
tfl_rescan_fonts()

## End(Not run)
```

`tfl_reset_options` *Reset all session options to package defaults*

Description

Restores all ksTFL session options (headers, footers, body text, styles, page settings, column width behavior, etc.) to their original package defaults. Useful at the start of a new reporting session or after experimenting with `tfl_set_options()`.

Usage

```
tfl_reset_options()
```

Value

The default options list, returned invisibly.

See Also

`tfl_set_options()`, `tfl_get_options()`, `tfl_get_option()`

Examples

```
## Not run:
# Set some session defaults
tfl_set_options(
  add_header("Study ABC", "Phase II", "CONFIDENTIAL"),
  add_footer("Company", "Page {PAGE}", "2025")
)

# ... build tables ...

# Reset everything back to package defaults
tfl_reset_options()
tfl_get_options()  # Confirm reset

## End(Not run)
```

`tfl_set_options` *Update the session package settings*

Description

Update ksTFL session options. This function accepts:

- Named scalar options (e.g. `contentWidth = "95%"`, `missings = "."`).
- Settings objects produced by helper constructors such as `add_header()`, `add_footer()`, `add_body_text()`, or page objects from `p_page()`.
- A mixture of both named values and settings objects.

Usage

```
tfl_set_options(
  ...,
  docTemplate = NULL,
  footnotePlace = NULL,
  isContinues = NULL,
  contentWidth = NULL,
  missings = NULL,
  figureWidth = NULL,
  figureHeight = NULL,
  figureDevice = NULL,
  figureScaleMode = NULL,
  autoColWidth = NULL,
  minColWidth = NULL,
  insertTOC = NULL,
  tocTitle = NULL,
  output_directory = NULL,
  meta_directory = NULL
)
```

Arguments

`...`	Named arguments OR settings objects returned from helper constructors. • Named scalar options (e.g. `contentWidth = "95%"`, `missings = "."`). • Settings objects produced by helper constructors such as `add_header()`, `add_footer()`, `add_style()`, `add_body_text()`, and `set_page_style(p_page(p_margins()))`. • A mixture of both named values and settings objects is accepted; the function routes each into the appropriate internal slot.
`docTemplate`	Character; name of a predefined bundled template (e.g. `"CRO Example_default"`, `"Navy_Pro"`) or a file path to an external template JSON file. When `NULL` (default) the current session template is left unchanged. Use `tfl_reset_options()` to restore the built-in default (`"CRO Example_default"`).

`footnotePlace` Character; controls where footnotes are rendered. One of `"doc_footer"` (place inside the Word footer, below footer rows), `"repeated"` (place under the table on every page), or `"last_page"` (place under the table on the last page only). Default `"repeated"`.

`isContinues` Logical; override continuation behavior.

`contentWidth` Character; width for content area (e.g. "100%", "95%").

`missings` Character; default representation for missing values (e.g. "", ".", "NA", "—"). Default is an empty string ("").

`figureWidth` Character; default width for figure output (e.g. `"6in"`, `"16cm"`). Applied when `create_figure()` specs do not specify their own width.

`figureHeight` Character; default height for figure output (e.g. `"4in"`, `"10cm"`). Applied when `create_figure()` specs do not specify their own height.

`figureDevice` Character; graphics device used for figure rendering (e.g. `"png"`, `"pdf"`, `"svg"`). Default depends on system capabilities.

`figureScaleMode` Character; how figures are scaled into the page content area. Typically `"fit"` (scale to fit) or `"exact"` (use exact dimensions).

`autoColWidth` Logical; enable automatic column width recalculation when user sets `colWidth` via `define_cols()`. Default TRUE. When TRUE, locked columns maintain exact width while unlocked columns normalize to fill remaining space. Set FALSE to disable auto-recalculation and manage widths manually.

`minColWidth` Numeric; minimum relative column width (%) for unlocked columns during recalculation. Default 0.5. Used to validate that relative widths don't squeeze columns below acceptable minimum.

`insertTOC` Logical; when `TRUE` the renderer prepends a Table of Contents page (using a `{ TOC \f \h \z }` field) before the first spec. Requires at least one `add_title()` or `add_subtitle()` call with `toclevel` set. Default `FALSE`. Can be overridden per-render via `save_report(insertTOC = )`.

`tocTitle` Character; heading text placed above the TOC field on the TOC page. Default `"Table of Contents"`. Set to `""` to omit the heading. Can be overridden per-render via `save_report(tocTitle = )`.

`output_directory` Character; path to default output directory of rendered document.

`meta_directory` Character; path to default directory for intermediate metadata (JSON specs, data, and asset files) created during rendering.

Details

The function tries to intelligently route each supplied object into the appropriate internal settings slot (headers, footers, styles, bodyText, page).

Value

The updated settings list, returned invisibly. Use `tfl_get_options()` to inspect.

Examples

```
## Not run:
# Set a named option
tfl_set_options(contentWidth = "95%", missings = ".")

# Set a predefined bundled template
tfl_set_options(docTemplate = "Navy_Pro")

# Set an external template file
tfl_set_options(docTemplate = "/path/to/my_template.json")

# Update page style via helper
 tfl_set_options(
   set_page_style(page= p_page(
   size = "Letter",
   orientation = "portrait",
   margins = p_margins(top = "1in", bottom = "1in", left = "0.75in", right = "0.75in")
 )))

# Add default header and footer via helpers
tfl_set_options(
  add_header(c("Left Header", "Center Header", "Right Header")),
  add_footer(c("Left Footer", "Center Footer", "Right Footer"))
)

# Add default body text via helper
tfl_set_options(
  add_body_text("This is the default body text for all text specs.")
)

# Control automatic column width recalculation
# Enable auto-recalculation (default):
tfl_set_options(autoColWidth = TRUE)
spec <- create_table(data) |>
  define_cols("id", colWidth = "20%")  # Locks id, others auto-adjust

# Disable auto-recalculation for manual width management:
tfl_set_options(autoColWidth = FALSE)
spec <- create_table(data) |>
  define_cols(c("id", "age"), colWidth = c("20%", "30%"))  # Exact widths, no auto-adjust

# Enable TOC page for all reports in the session
tfl_set_options(insertTOC = TRUE, tocTitle = "List of Tables")
# Then mark individual titles/subtitles with toclevel:
spec <- create_table(adsl) |>
  add_title("Table 1: Demographics", toclevel = 1)
# save_report() will now prepend a TOC page automatically.
# In Word: click the TOC placeholder and press F9 to populate it.

## End(Not run)
```

tfl_spec_addins *RStudio Addins for TFL Specification Preview*

Description

Two addins that open the HTML TFL Specification Preview in the RStudio Viewer pane.

Usage

```
tfl_spec_preview_selection()

tfl_spec_preview_prompt()
```

Details

`tfl_spec_preview_selection()` Evaluates the currently selected text in the source editor and, if the result is a `TFL_spec` object, opens the HTML preview.

`tfl_spec_preview_prompt()` Prompts for the name of an object in `.GlobalEnv` and, if it is a `TFL_spec`, opens the HTML preview.

Both functions require RStudio (`rstudioapi::isAvailable()`).

See Also

`view_tfl_spec()`

view_tfl_spec *Open the HTML TFL Specification Preview in the RStudio Viewer*

Description

Renders a comprehensive HTML overview of a TFL specification and displays it in the RStudio Viewer pane. Requires RStudio and the **htmltools** package.

Usage

```
view_tfl_spec(spec)
```

Arguments

`spec`	A `TFL_spec` object.

Value

`TRUE` invisibly on success, `FALSE` invisibly if the viewer could not be opened (e.g. not running inside RStudio or **htmltools** is missing).

write_doc *Save and Render a TFL Report to DOCX*

Description

Convenience wrapper around save_report() and the internal DOCX renderer that saves a TFL_report object to JSON (plus any required data/figure files) and immediately renders it to a DOCX file in a single call.

Usage

```
write_doc(
  report,
  name,
  outDir = tfl_get_option("output_directory"),
  metaPath = tfl_get_option("meta_directory") %||% tempdir(),
  prettify = FALSE,
  toc = tfl_get_option("insertTOC"),
  tocTitle = tfl_get_option("tocTitle"),
  overrideTemplate = NULL,
  font_dirs = NULL,
  fallback_font = NULL,
  verbose = FALSE
)
```

Arguments

report	A TFL_report object created by create_report().
name	Character(1). Base file name (without extension) for the output DOCX document. The .docx extension is appended automatically.
outDir	Character(1). Directory where the final DOCX file will be written. Defaults to tfl_get_option("output_directory").
metaPath	Character(1). Directory where the intermediate specification JSON and associated data/figure files will be stored. Defaults to tfl_get_option("meta_directory").
prettify	Logical. When TRUE, pretty-prints the JSON written by save_report() for easier inspection. Default FALSE (compact JSON).
toc	Logical. When TRUE, enables automatic insertion of a Table of Contents page via save_report(). Defaults to tfl_get_option("insertTOC").
tocTitle	Character(1). Heading placed above the TOC field on the TOC page. Defaults to tfl_get_option("tocTitle").
overrideTemplate	Optional character string. Global template override used by the internal renderer for all specs. Accepts either: • A predefined bundled template name (e.g. "Navy_Pro").

- A file path (absolute or relative) to an external template JSON file.

If NULL (default), templates are resolved per-spec from each spec's docTemplate value (allowing mixed templates in multi-spec reports). If a provided name/path cannot be resolved, a warning is emitted and CRO Example_default is used.

font_dirs — Optional character vector of additional directories to search for fonts during rendering.

fallback_font — Optional character string. Path to a fallback font file used by the renderer. If NULL, the package default is used.

verbose — Logical. If TRUE, prints renderer progress messages.

Details

This mirrors the helper used in the example inst/examples/init.R script (previously called save_and_render()), but is provided as a public, documented API function named write_doc().

Value

Invisibly returns the full path to the generated .docx file.

See Also

create_report(), save_report(), replay_report(), tfl_set_options(), tfl_get_option()

Examples

```
## Not run:
# Basic end-to-end workflow ------------------------------------------------
library(ksTFL)

# Create a simple table spec
tbl <- create_table(mtcars) |>
  add_title("Table 1: Motor Trend Car Road Tests") |>
  set_document(hasData = TRUE)

# Combine into a report
rpt <- create_report(tbl)

# Write DOCX to the default output directory (getwd() by default)
doc_path <- write_doc(
  report = rpt,
  name   = "mtcars_demo"
)

cat("DOCX written to:", doc_path, "\n")

# Custom output and meta directories, with TOC ----------------------------
tfl_set_options(
  output_directory = "output",
  insertTOC = TRUE,
  tocTitle  = "List of Tables"
```

```
)

rpt2 <- create_report(tbl)
write_doc(
  report   = rpt2,
  name     = "mtcars_with_toc",
  outDir   = "output",
  metaPath = file.path(tempdir(), "ksTFL_meta"),
  prettify = TRUE,
  toc      = TRUE
)

## End(Not run)
```

Index