spec_table_latex.R

vincent

2024-01-05

```r
spec_table_latex <- function(x,
                             hlines = "booktabs",
                             vlines = FALSE,
                             inner = NULL,
                             outer = NULL) {

  checkmate::assert_string(inner, null.ok = TRUE)
  checkmate::assert_string(outer, null.ok = TRUE)
  checkmate::assert(
    checkmate::check_choice(hlines, choices = "booktabs"),
    checkmate::check_integerish(hlines, lower = 1, upper = nrow(x)),
    checkmate::check_flag(hlines)
  )

  template <- readLines(here::here("inst/template_tblr.tex"))

  if (!is.null(colnames(x))) {
    header <- paste(colnames(x), collapse = " & ")
    header <- paste(header, "\\\\")
  } else {
    header <- NULL
  }
  body <- apply(x, 1, paste, collapse = " & ")
  body <- paste(body, "\\\\")

  if (isTRUE(hlines == "booktabs")) {
    header <- c("\\toprule", header, "\\midrule")
    body <- c(body, "\\bottomrule")
  }

  idx <- grep("\\$TINYTABLE_BODY", template)
  out <- c(
    template[1:(idx - 1)],
    header,
    body,
    template[(idx + 1):length(template)]
  )

  out <- trimws(out)
  out <- paste(out, collapse = "\n")

  tabularray_cols <- rep("Q[]", ncol(x))
  tabularray_rows <- rep("Q[]", nrow(x))
  if (!is.null(header)) {
    tabularray_rows <- c("Q[]", tabularray_rows)
  }

  # colspec & rowspec
  new <- c(
    sprintf("colspec={%s},", paste(tabularray_cols, collapse = "")),
    sprintf("rowspec={%s},", paste(tabularray_rows, collapse = ""))
  )
  out <- tabularray_setting(out, new, inner = TRUE)

  # vlines
  if (isTRUE(vlines)) {
    out <- tabularray_setting(out, "vlines,", inner = TRUE)
  } else if (isTRUE(checkmate::check_integerish(vlines))) {
    out <- tabularray_setting(
      out,
      sprintf("vlines={%s}{solid},", paste(vlines, collapse = ",")),
      inner = TRUE)

  }

  # hlines
  if (isTRUE(hlines)) {
    out <- tabularray_setting(out, "hlines,", inner = TRUE)
  }

  if (!is.null(inner)) {
    if (!grepl(",$", trimws(inner))) inner <- paste0(inner, ",")
    out <- tabularray_setting(out, inner, inner = TRUE)
  }
  if (!is.null(outer)) {
    if (!grepl(",$", trimws(outer))) outer <- paste0(outer, ",")
    out <- tabularray_setting(out, outer, inner = FALSE)
  }

  attr(out, "ncol") <- ncol(x)
  attr(out, "nrow") <- nrow(x)
  attr(out, "tabularray_cols") <- tabularray_cols
  attr(out, "tabularray_rows") <- tabularray_rows
  class(out) <- c("tinytable_latex", class(out))
  return(out)
}



tabularray_setting <- function(x, new, inner = TRUE) {
  att <- attributes(x)
  out <- strsplit(x, "\n")[[1]]
  if (isTRUE(inner)) {
    idx <- grep("% tabularray inner close", out)
  } else {
    idx <- grep("% tabularray outer close", out)
  }
  out <- c(
    out[1:(idx - 1)],
    new,
    out[idx:length(out)]
  )
  out <- paste(out, collapse = "\n")
  attributes(out) <- att
  return(out)
}



tabularray_spec <- function(bold,
                            italic,
                            monospace,
                            smallcaps,
                            fg,
                            bg,
                            wd,
                            halign) {
  # Initialize spec
  spec <- ""

  # Flag styles
  args <- list(
    "bold" = list(bold, "\\\\bfseries"),
    "italic" = list(italic, "\\\\textit"),
    "monospace" = list(monospace, "\\\\texttt"),
    "smallcaps" = list(smallcaps, "\\\\scshape")
  )

  font <- ""
  for (n in names(args)) {
    flag <- checkmate::check_flag(args[[n]][[1]])
    if (!isTRUE(flag)) {
      msg <- sprintf("`%s` is not a logical flag.", n)
      stop(msg, call. = FALSE)
    }
    if (isTRUE(args[[n]][[1]])) {
      font <- paste0(font, args[[n]][[2]])
    }
  }
  if (font != "") {
    spec <- paste0(spec, "cmd=", font, ",")
  }

  # String settings: fragile input checks
  args <- list(
    "fg" = fg,
    "bg" = bg,
    "wd" = wd
  )
  for (n in names(args)) {
    flag <- checkmate::check_string(args[[n]], null.ok = TRUE)
    if (!isTRUE(flag)) {
      msg <- sprintf("`%s` is not a string.", n)
      stop(msg, call. = FALSE)
    }
    spec <- paste0(spec, sprintf("%s=%s,", n, args[[n]]))
  }

  # Horizontal alignment
  checkmate::assert_choice(halign, choices = c("c", "l", "r"), null.ok = TRUE)
  if (!is.null(halign)) {
    tmp <- sprintf("halign=%s,", halign)
    spec <- paste0(spec, tmp)
  }

  # Overwrite Q[]/X[] brackets
  spec <- sprintf("[%s]", spec)

  return(spec)
}
```