Package 'reporter'

September 8, 2021

Type Package

Title Creates Statistical Reports

Version 1.1.6

Author David Bosak

Maintainer David Bosak <dbosak01@gmail.com>

Description Contains functions to create regulatory-style statistical reports. Originally designed to create tables, listings, and figures for the pharmaceutical, biotechnology, and medical device industries, these reports are generalized enough that they could be used in any industry. Generates text, rich-text, and PDF file formats. The package specializes in printing wide and long tables with automatic page wrapping and splitting. Reports can be produced with a minimum of function calls, and without relying on other table packages. The package supports titles, footnotes, page header, page footers, spanning headers, page by variables, and automatic page numbering.

License CC BY-NC 4.0

Encoding UTF-8

URL https://reporter.r-sassy.org

BugReports https://github.com/dbosak01/reporter/issues

Depends R (>= 3.6.0)

Suggests testthat, magrittr, dplyr, tidyr, readr, knitr, rmarkdown, ggplot2, gridExtra, survminer, utils, logr, covr, graphics

Imports fmtr, stringi, crayon, jpeg

RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2021-09-08 16:10:01 UTC

R topics documented:

`add_content` *Add content to a report*

Description

This function adds an object to the report content list. A report will accept multiple pieces of content. The `add_content` function also controls overall alignment of the content on the page, and whether there is a page break before or after.

Usage

```
add_content(
  x,
  object,
  page_break = TRUE,
  align = "center",
  blank_row = "below"
)
```

Arguments

x	The report_spec to append content to.
object	The object to append.
page_break	Whether to add a page break after the object. Valid values are TRUE or FALSE. You can manipulate the page_break parameter to add multiple objects to the same page.
align	How to align the content. Valid values are 'left', 'right', 'center', and 'centre'.
blank_row	Whether to put a blank row above or below the content. Valid values are 'above', 'below', 'both', or 'none'.

Details

The add_content function adds a piece of content to a report. For a text report, valid objects are a table or text object. For an RTF or PDF report, valid objects are a table, text, or plot object. See create_table, create_text, or create_plot for further information on how to create content objects.

Content will be appended to the report in the order it is added. By default, a page break is added after the content. You can stack two pieces of content together closely by setting the page_break parameter to FALSE, and the blank_row parameter to "none".

Value

The modified report_spec.

See Also

create_table, create_text, and create_plot to create content for a report.

Other report: create_report(), footnotes(), options_fixed(), page_by(), page_footer(), page_header(), print.report_spec(), set_margins(), title_header(), titles(), write_report()

Examples

```
library(reporter)
library(magrittr)

# Create temp file path
tmp <- file.path(tempdir(), "mtcars.txt")

# Create first table
tbl1 <- create_table(mtcars[1:5, 1:6]) %>%
  column_defaults(width = .5)

# Create second table
tbl2 <- create_table(mtcars[6:10, 1:6], headerless=TRUE) %>%
  column_defaults(width = .5)

# Create the report object
rpt <- create_report(tmp) %>%
```

```
  titles("MTCARS Sample Data", align = "left") %>%
  add_content(tbl1, page_break = FALSE, align = "left", blank_row = "none") %>%
  add_content(tbl2, page_break = FALSE, align = "left") %>%
  add_content(create_text("* NOTE: Above table is actually two tables stacked."))

# Write the report to the file system
res <- write_report(rpt)

# Write report to console
writeLines(readLines(tmp, encoding = "UTF-8"))

# MTCARS Sample Data
#
#    mpg    cyl   disp     hp   drat     wt
# ----------------------------------------
#     21      6    160    110    3.9   2.62
#     21      6    160    110    3.9  2.875
#   22.8      4    108     93   3.85   2.32
#   21.4      6    258    110   3.08  3.215
#   18.7      8    360    175   3.15   3.44
#   18.1      6    225    105   2.76   3.46
#   14.3      8    360    245   3.21   3.57
#   24.4      4  146.7     62   3.69   3.19
#   22.8      4  140.8     95   3.92   3.15
#   19.2      6  167.6    123   3.92   3.44
#
# * NOTE: Above table is actually two tables stacked.
```

`column_defaults` *Set default attributes for one or more columns*

Description

A function to set default attributes for columns on a table. The `column_defaults` function contains a subset of the parameters on the `define` function that can be shared across variables. Any attributes set by `column_defaults` can be overridden by the `define` function. The overall purpose of the function is to minimize redundancy in column definitions.

Usage

```
column_defaults(
  x,
  vars = NULL,
  from = NULL,
  to = NULL,
  label = NULL,
  format = NULL,
  align = NULL,
  label_align = NULL,
```

```
  width = NULL,
  n = NULL,
  standard_eval = FALSE
)
```

Arguments

x	A table spec.
vars	The variable name or names to define defaults for. Variable names may be quoted or unquoted. The parameter will also accept integer column positions instead of names. For multiple variables, pass the names or positions as a vector. If you want to pass an R variable of names, set the standard_eval parameter to TRUE. The standard_eval parameter is useful when writing functions that construct reports dynamically.
from	The variable name or position that starts a column range. If passed as a variable name, it may be quoted or unquoted.
to	The variable name or position that ends a column range. If passed as a variable name, it may be quoted or unquoted.
label	The label to use for a column header. This label will be applied to all variables assigned to the column_defaults function.
format	The format to use for the column data. The format can be a string format, a formatting function, a lookup list, a user-defined format, or a formatting list. All formatting is performed by the fmtr package. For additional information, see the help for that package.
align	The column alignment. Valid values are "left", "right", "center", and "centre".
label_align	How to align the header labels for this column. Valid values are "left", "right", "center", and "centre".
width	The width of the column in the specified units of measure. The units of measure are specified on the units parameter of the create_report function. If no width is supplied, the write_report function will assign a default width based on the width of the column data and the label. write_report will not set a column width less than the width of the largest word in the data or label. In other words, write_report will not break words.
n	The n value to place in the "N=" header label. Formatting for the n value will be performed by the formatting function assigned to the n_format parameter on create_table.
standard_eval	A TRUE or FALSE value that indicates whether to use standard or non-standard evaluation of the vars, from, and to parameters. Set standard_eval to TRUE if you want to pass the column names as variables. Default is FALSE, meaning it will use non-standard (unquoted) evaluation.

Details

Column defaults can be specified for multiple variables. By default, the function will apply to all variables. Alternately, you can specify a vector of columns on the vars parameter, or a range

of columns using the `from` and `to` parameters. Both the `vars` parameters and the `from` and `to` parameters will accept column positions, quoted variable names, or unquoted variable names.

The parameters that can be set with the `column_defaults` include the formatting attributes 'width', 'justify', 'label', and 'format'. Any parameters set with `column_defaults` will override any attributes set on the data frame.

Note that you may call the `column_defaults` function multiple times on the same table specification. Typically, multiple `column_defaults` calls would be made with a different set or range of variables.

Value

The modified table spec.

See Also

Other table: `create_table()`, `define()`, `print.table_spec()`, `spanning_header()`, `stub()`

Examples

```
library(reporter)
library(magrittr)

# Create temp file name
tmp <- file.path(tempdir(), "mtcars.txt")

# Prepare data
dat <- mtcars[1:10, ]
dat <- data.frame(vehicle = rownames(dat), dat, stringsAsFactors = FALSE)

# Define table
tbl <- create_table(dat, show_cols = 1:8) %>%
  column_defaults(from = mpg, to = qsec, width = .5, format = "%.1f") %>%
  define(vehicle, label = "Vehicle", width = 1.5, align = "left") %>%
  define(c(cyl, hp), format = "%.0f")

# Create the report
rpt <- create_report(tmp, orientation = "portrait") %>%
  titles("Table 2.5", "MTCARS Sample Report") %>%
  add_content(tbl)

# Write the report
write_report(rpt)

# Send report to console for viewing
writeLines(readLines(tmp, encoding = "UTF-8"))

#                                Table 2.5
#                           MTCARS Sample Report
#
#    Vehicle               mpg    cyl   disp     hp   drat     wt   qsec
#    ------------------------------------------------------------------
```

```
#    Mazda RX4           21.0     6  160.0    110    3.9    2.6   16.5
#    Mazda RX4 Wag       21.0     6  160.0    110    3.9    2.9   17.0
#    Datsun 710          22.8     4  108.0     93    3.8    2.3   18.6
#    Hornet 4 Drive      21.4     6  258.0    110    3.1    3.2   19.4
#    Hornet Sportabout   18.7     8  360.0    175    3.1    3.4   17.0
#    Valiant             18.1     6  225.0    105    2.8    3.5   20.2
#    Duster 360          14.3     8  360.0    245    3.2    3.6   15.8
#    Merc 240D           24.4     4  146.7     62    3.7    3.2   20.0
#    Merc 230            22.8     4  140.8     95    3.9    3.1   22.9
#    Merc 280            19.2     6  167.6    123    3.9    3.4   18.3
#
```

`create_plot` *Create plot content*

Description

Function to create a plot specification that can be added as content to a report. The `create_plot` function can be used to include charts, graphs, and figures on a statistical report. The function only supports plot objects returned by `ggplot` or `ggsurvplot`. It does not support the Base R `plot` function.

Usage

```
create_plot(x, height, width)
```

Arguments

`x`	The plot to create. Specifically, this parameter should be set to an object returned from a call to `ggplot` or `ggsurvplot`.
`height`	The height of the plot in the specified units of measure.
`width`	The width of the plot in the specified units of measure.

Details

To add a plot to a report, use the `create_plot` function. The function allows you to set a width and height for the plot. The function will preserve any other geometries you apply to the plot. See the `add_content` function to control page breaking and blanks spaces above or below the plot.

A plot specification accepts a `page_by` function. If a page by is applied to the plot, the plot data will be subset by the page by variable, and re-run for each subset.

The plot specification also accepts titles and footnotes. See the `titles` and `footnotes` functions for further details.

Value

The plot specification.

See Also

titles to add a title block to the plot, footnotes to add footnotes, and add_content to add the plot object to a report.

Other plot: print.plot_spec()

Examples

```
library(reporter)
library(ggplot2)
library(magrittr)

# Create temp file path
tmp <- file.path(tempdir(), "mtcars.rtf")

# Create ggplot
p <- ggplot(mtcars, aes(x=cyl, y=mpg)) + geom_point()

# Create plot object
plt <- create_plot(p, height = 4, width = 8)

rpt <- create_report(tmp, output_type = "RTF") %>%
  page_header("Client", "Study: XYZ") %>%
  titles("Figure 1.0", "MTCARS Miles per Cylinder Plot") %>%
  set_margins(top = 1, bottom = 1) %>%
  add_content(plt) %>%
  footnotes("* Motor Trend, 1974") %>%
  page_footer("Time", "Confidential", "Page [pg] of [tpg]")

# Write out report
write_report(rpt)

# Uncomment to view RTF file
# shell.exec(tmp)
```

---

create_report *Create a report*

Description

Creates a report shell to which you may add titles, footnotes, content, etc.

Usage

```
create_report(
  file_path = "",
  output_type = "TXT",
  orientation = "landscape",
  units = "inches",
```

```
  paper_size = "letter",
  missing = ""
)
```

Arguments

file_path	The output path of the desired report. Either a full path or a relative path is acceptable. This parameter is not required to create the report_spec object, but will be required to write the report. In addition, the file extension is not required. If the file extension is not supplied, the write_report function will add a file extension based on the output_type specified.
output_type	The report output type. Default is "TXT". Valid values are "TXT", "RTF", and "PDF".
orientation	The page orientation of the desired report. Valid values are "landscape" or "portrait". The default page orientation is "landscape".
units	Specifies the units of measurement. This setting will indicate the units for columns widths, margins, paper size, and other measurements. Valid values are "inches" or "cm" (centimeters). Default value is "inches".
paper_size	The expected paper size on which the report may be printed. The paper_size will determine how much text can fit on one page. Valid values are "letter", "legal", "A4", and "RD4". Default is "letter".
missing	How to display missing values in the report. Default is to replace them with an empty string, which removes them from the report. To display missing values as is, set the missing parameter to NULL. To replace missing values with a character string (such as ".", "-", or "<NA>") pass the desired character string to the missing parameter.

Details

This function is the constructor for the report object. The report object contains information needed to create a report. The object is defined as an S3 object, and has a class of 'report_spec'.

The report object holds information concerning report page size, orientation, titles, footnotes, page header, page footer, margins, and other options. Use the add_content function to add content to the report. The report may be written to a file using the write_report function.

Value

A new report_spec object.

Report family of functions

The report is the primary container for report specifications. The following functions add additional specifications to the report object initialized with create_report.

- titles to add titles to the report.
- footnotes to add footnotes to the report.
- title_header to add a title header to the report.

- page_header to add a page header to the report.
- page_footer to add a page_footer to the report.
- add_content to add content to the report.
- options_fixed to set options for fixed-width output.
- add_content to add content to the report.
- write_report to write the report to the file system.

The report family of functions are pipe-friendly. After creating the report, you may pipe the object to any of the above functions to append additional options.

Note that PDF output has some limitations not found in TXT and RTF output. See NotesOnPDF for additional information.

See Also

create_table, create_text, and create_plot functions to create content for the report.

Other report: add_content(), footnotes(), options_fixed(), page_by(), page_footer(), page_header(), print.report_spec(), set_margins(), title_header(), titles(), write_report()

Examples

```
library(reporter)
library(magrittr)

# Create temp file path
tmp <- file.path(tempdir(), "airquality.txt")

# Prepare Data
dat <- airquality[sample(1:153, 15), ]
dat$Month <-  as.Date(paste0("1973-", dat$Month, "-01"))

# Define table
tbl <- create_table(dat, show_cols = c("Month", "Day", "Wind", "Temp", "Ozone")) %>%
  titles("Table 9.6", "Air Quality Sample Report") %>%
  column_defaults(width = .5) %>%
  define(Month, format = "%B", align = "left", width = 1) %>%
  define(Temp, format = "%.0f") %>%
  footnotes("* New York, May to September 1973")

# Define report
rpt <- create_report(tmp, orientation = "portrait",  missing = "-") %>%
  add_content(tbl)

# Write the report to the file system
write_report(rpt)

# Write the report to the console
writeLines(readLines(tmp, encoding = "UTF-8"))

#                      Table 9.6
#              Air Quality Sample Report
```

```
#
#      Month           Day   Wind   Temp  Ozone
#      ---------------------------------------
#      July              8    6.3     92     97
#      July              9    5.7     92     97
#      August            1    6.9     81     39
#      July             23   11.5     82      -
#      June              9   13.8     90     71
#      July             12   14.3     73     10
#      July              4   10.9     84      -
#      May              31    7.4     76     37
#      September        30   11.5     68     20
#      June             25      8     75      -
#      June             28   11.5     80      -
#      August           18    7.4     76     23
#      June             20   10.3     76     13
#      July              1    4.1     84    135
#      May              23    9.7     61      4
#
#      * New York, May to September 1973
```

`create_table` *Create a table*

Description

The `create_table` function creates a table object to which further specifications can be added. The object can be added to a report using the `add_content` function. The object is implemented as an S3 object of class 'table_spec'.

Usage

```
create_table(
  x,
  show_cols = "all",
  use_attributes = "all",
  width = NULL,
  first_row_blank = FALSE,
  n_format = upcase_parens,
  headerless = FALSE
)
```

Arguments

`x`	The data frame or tibble from which to create the table object.
`show_cols`	This parameter gives control over which columns in the input data to display on the report by default. Valid values are 'all', 'none', a vector of quoted column names, or a vector of column positions. 'all' means show all columns, unless

overridden by the column definitions. 'none' means don't show any columns unless specified in the column definitions. If a vector of column names or positions is supplied, those columns will be shown in the report in the order specified, whether or not a definition is supplied. See the `define` function for additional information on how to show/hide report columns.

`use_attributes` Whether or not to use any formatting attributes assigned to the columns on the input data frame. Valid values are 'all', 'none', or a vector of attribute names to use. Possible attributes that may be used are 'label', 'format', 'width', and 'justify'. By default, any of these attribute values will be applied to the table. For example, if you assign a label to the 'label' attribute of a data frame column, pass that data frame into `create_table`, and don't override the label value on a `define` function, the label will appear as a column header on the table. The `use_attributes` parameter allows you to control this default behavior, and use or ignore data frame attributes as desired.

`width` The expected width of the table in the report units of measure. By default, the width setting is NULL, and columns will be sized according to the width of the data and labels. If the width parameter is set, the function will attempt to size the table to the specified width. If the sum of the column widths is less than the specified width, the function will adjust the columns widths proportionally to fit the specified width. If the sum of the column widths is wider than the table width parameter value, the table width parameter will be ignored.

`first_row_blank`

Whether to place a blank row under the table header. Valid values are TRUE or FALSE. Default is FALSE.

`n_format` The formatting function to apply to the header "N=" label. The default formatting function is `upcase_parens`.

`headerless` Whether to create a headerless table. A headerless table displays the table data only. Default is FALSE, meaning the table will have a header.

Details

A table object is a container to hold information about a table. The only required information for a table is the table data. All other parameters and functions are optional.

By default, the table will display all columns in the data frame. To change this default, use the `show_cols` parameter. Setting this parameter to 'none' will display none of the columns in the data, unless they are explicitly defined with a `define` function.

The `show_cols` parameter also accepts a vector of column positions or column names. When a vector is supplied, `create_table` will display only those columns on the report, in the order encountered in the vector. The `show_cols` parameter is the only mechanism in `create_table` to modify the column order. Otherwise, modify the order prior to sending the data to `create_table` using the many options available in Base R or supplemental packages.

Setting Formatting Attributes

Formatting attributes can be controlled in three ways. By default, formatting attributes assigned to the data frame will be passed through to the reporting functions. The reporting functions will recognize the 'label', 'format', 'width', and 'justify' attributes. In other words, you can control

the column label, width, format, and alignment of your report columns simply by assigning those attributes to your data frame. The advantage of using attributes assigned to data frame columns is that you can store those attributes permanently with the data frame, and those attributes will not have to be re-specified for each report. To ignore attributes assigned to the data frame, set the `use_attributes` parameter to 'none'.

Secondly, attributes can be specified using the `column_defaults` function. This function allows the user to apply a default set of parameters to one or more columns. If no columns are specified in the `var` or `from` and `to` parameter of this function, the defaults will apply to all columns. Any default parameter value can be overridden by the `define` function.

Lastly, the `define` function provides the most control over column parameters. This function provides a significant amount of functionality that cannot be specified elsewhere. See the `define` function for additional information. The `define` function will also override any formatting attributes assigned to the data frame, or anything set by the `column_defaults` function.

Additional Functionality

The `create_table` function also provides the capabilities to create a "headerless" table. A headerless table is useful when combining two tables into one report. The example below illustrates use of a headerless table.

Since the purpose of the **reporter** package is to create statistical reports, the `create_table` function makes it easy to add population counts to the table header. These population counts are added to column labels and spanning header labels using the `n` parameter on the `define` or `spanning_header` functions. The population count is formatted according to the `n_format` parameter on `create_table`. The **reporter** package provides four population count formatting functions. You may create your own formatting function if one of these functions does not meet your needs. See `upcase_parens` for further details.

See Also

`create_report` to create a report, `create_plot` to create a plot, `create_text` to create text content, and `add_content` to append content to a report. Also see the `titles`, `footnotes`, and `page_by` functions to add those items to the table if desired.

Other table: `column_defaults()`, `define()`, `print.table_spec()`, `spanning_header()`, `stub()`

Examples

```
library(reporter)
library(magrittr)

# Create temp file path
tmp <- file.path(tempdir(), "mtcars.txt")

#Subset cars data
dat <- mtcars[1:10, 1:7]

# Calculate means for all columns
dat_sum <- data.frame(all_cars = "All cars average", as.list(sapply(dat, mean)),
                      stringsAsFactors = FALSE)
```

```
# Get vehicle names into first column
dat_mod <- data.frame(vehicle = rownames(dat), dat, stringsAsFactors = FALSE)

# Create table for averages
tbl1 <- create_table(dat_sum) %>%
        titles("Table 1.0", "MTCARS Sample Data") %>%
        column_defaults(width = .5) %>%
        define(all_cars, label = "", width = 2) %>%
        define(mpg, format = "%.1f") %>%
        define(disp, format = "%.1f") %>%
        define(hp, format = "%.0f") %>%
        define(qsec, format = "%.2f")

# Create table for modified data
tbl2 <- create_table(dat_mod, headerless = TRUE) %>%
        column_defaults(width = .5) %>%
        define(vehicle, width = 2)

# Create the report object
rpt <- create_report(tmp) %>%
  add_content(tbl1, align = "left", page_break = FALSE) %>%
  add_content(tbl2, align = "left")

# Write the report to the file system
write_report(rpt)

# Write report to console
writeLines(readLines(tmp, encoding = "UTF-8"))

#                               Table 1.0
#                            MTCARS Sample Data
#
#                            mpg    cyl   disp     hp   drat     wt   qsec
# ---------------------------------------------------------------------------
# All cars average          20.4    5.8  208.6    123  3.538  3.128  18.58
#
# Mazda RX4                   21      6    160    110    3.9   2.62  16.46
# Mazda RX4 Wag               21      6    160    110    3.9  2.875  17.02
# Datsun 710                22.8      4    108     93   3.85   2.32  18.61
# Hornet 4 Drive            21.4      6    258    110   3.08  3.215  19.44
# Hornet Sportabout         18.7      8    360    175   3.15   3.44  17.02
# Valiant                   18.1      6    225    105   2.76   3.46  20.22
# Duster 360                14.3      8    360    245   3.21   3.57  15.84
# Merc 240D                 24.4      4  146.7     62   3.69   3.19     20
# Merc 230                  22.8      4  140.8     95   3.92   3.15   22.9
# Merc 280                  19.2      6  167.6    123   3.92   3.44   18.3
#
```

`create_text` *Create text content*

Description

Function to create a text specification that can be added as content to a report. The text content can be used to include analysis on a statistical report. A text specification is an S3 object of class 'text_spec'.

Usage

```
create_text(txt, width = NULL, align = "left")
```

Arguments

txt	The text to create.
width	The width of the text in the specified units of measure. If no width is specified, the full page width will be used.
align	How to align the text within the content area. Valid values are 'left', 'right', 'center', or 'centre'. Default is 'left'.

Details

To add plain text to a report, use the `create_text` function. The function allows you to set a width and alignment for the text. The function will preserve any other formatting you apply to the text. See the `add_content` function to control page breaking and blanks spaces above or below the text.

The text specification also accepts titles and footnotes. See the `titles` and `footnotes` functions for further details.

Value

The text specification.

See Also

`titles` to add a title block to the text, `footnotes` to add footnotes, and `add_content` to add the text object to a report.

Other text: `print.text_spec()`

Examples

```
library(reporter)
library(magrittr)

# Create temp file path
tmp <- file.path(tempdir(), "mtcars.txt")

# Create dummy text
dt <- paste0("Lorem ipsum dolor sit amet, consectetur adipiscing elit, ",
  "sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. ",
  "Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris ",
  "nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in ",
  "reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla ",
```

```
  "pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa ",
  "qui officia deserunt mollit anim id est laborum.")

# Create the text object
txt <- create_text(dt) %>%
  titles("Text Content 1.0", "Sample Text Report") %>%
  footnotes("* Cicero, 1st century BCE")

# Create the report object
rpt <- create_report(tmp, orientation = "portrait") %>%
  add_content(txt)

# Write the report to the file system
write_report(rpt)

# Write the report to console
writeLines(readLines(tmp, encoding = "UTF-8"))

#                                Text Content 1.0
#                               Sample Text Report
#
# Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
# incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis
# nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
# Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore
# eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt
# in culpa qui officia deserunt mollit anim id est laborum.
#
# * Cicero, 1st century BCE
#
```

`define` *Defines a column*

Description

A function to define a table column. The `define` function contains a variety of a parameters to control the appearance of the report. Using the `define` function, you can control simple options like column alignment and width, but also control more sophisticated options like page wrapping and page breaking.

Usage

```
define(
  x,
  vars,
  label = NULL,
  format = NULL,
  align = NULL,
```

```
  label_align = NULL,
  width = NULL,
  visible = TRUE,
  n = NULL,
  blank_after = FALSE,
  dedupe = FALSE,
  id_var = FALSE,
  page_wrap = FALSE,
  page_break = FALSE,
  indent = NULL,
  label_row = FALSE,
  standard_eval = FALSE
)
```

Arguments

x	The table spec.
vars	The variable name or names to define a column for. Names may be quoted or unquoted. If defining for multiple variables, pass them as a vector of names. If you want to pass an R variable of names, set the standard_eval parameter to TRUE . The standard_eval parameter is useful when writing functions that construct reports dynamically.
label	The label to use for the column header. If a label is assigned to the label column attribute, it will be used as a default. Otherwise, the column name will be used.
format	The format to use for the column data. The format can be a string format, a formatting function, a lookup list, a user-defined format, or a formatting list. All formatting is performed by the fapply function from the fmtr package. For a list of common formatting codes, see FormattingStrings.
align	The column alignment. Valid values are "left", "right", "center", and "centre". By default, text columns will be left aligned and numeric columns will be right aligned.
label_align	How to align the header labels for this column. Valid values are "left", "right", "center", and "centre". By default, the label alignment will follow any alignment set on the column align parameter.
width	The width of the column in the specified units of measure. The units of measure are specified on the units parameter of the create_report function. If no width is supplied, the write_report function will assign a default width based on the width of the column data and the label. write_report will not set a column width less than the width of the largest word in the data or label. In other words, write_report will not break words.
visible	Whether or not the column should be visible on the report. This parameter can be used as a simple way to drop columns from the report.
n	The n value to place in the "N=" header label. Formatting for the n value will be performed by the formatting function assigned to the n_format parameter on create_table.
blank_after	Whether to place a blank row after unique values of this variable. Valid values are TRUE or FALSE. Default is FALSE.

dedupe	Whether to dedupe the values for this variable. Variables that are deduped only show the value on the first row of each group. This option is commonly used for grouping variables.
id_var	Whether this variable should be considered an ID variable. ID variables are retained on each page when the page is wrapped. ID variables are also moved to the far left of the page.
page_wrap	Force a page wrap on this variable. A page wrap is a vertical page break necessary when the table is too wide to fit on a single page. The excess variables will be wrapped to the next page. Page wraps will continue until all columns are displayed. Use the id_var parameter to identify rows across wrapped pages.
page_break	You may control when page breaks occur by defining a page break variable yourself, and setting this parameter to TRUE for that variable. Only one page break variable can be defined per table. If two or more variables are defined as a page break, an error will be generated.
indent	How much to indent the column values. The parameter takes a numeric value that will be interpreted according to the units (Unit Of Measure) setting on the report. This parameter can be used to help create a stub column. The default value is NULL, meaning the column should not be indented. See the stub function for additional information on creating a stub column.
label_row	Whether the values of the variable should be used to create a label row. Valid values are TRUE or FALSE. Default is FALSE. If label_row is set to TRUE, the dedupe parameter will also be set to TRUE. This parameter is often used in conjunction with the stub function and indent parameter to create a stub column.
standard_eval	A TRUE or FALSE value indicating whether to use standard evaluation on the vars parameter value. Default is FALSE. Set this parameter to TRUE if you want to pass the vars value(s) using a variable.

Details

Column definitions are optional. By default, all columns in the data are displayed in the order assigned to the data frame.

The report will use attributes assigned to the data frame such as 'width', 'justify', 'label', and 'format'. In other words, some control over the column formatting is available by manipulating the data frame attributes prior to assigning the data frame to create_table. See create_table for more details.

The define function is used to provide additional control over column appearance. For example, you may use the define function to assign an "N=" population count, eliminate duplicates from the column, or place a blank row after each unique value of the variable. See the parameter documentation for additional options.

Some of the parameters on the define function are used in the creation of a table stub. Specifically, the label_row and indent parameters participate in the creation of the stub column. See the stub function for further information.

A single column definition may be defined for multiple variables. To create a definition for multiple variables, pass the variables as a quoted or unquoted vector. When creating a single definition for

multiple variables, the parameters will be unified across those variables. Note that some parameters (such as page_break) may only be set once per report, and cannot be shared across multiple variables.

Value

The modified table spec.

See Also

Other table: column_defaults(), create_table(), print.table_spec(), spanning_header(), stub()

Examples

```
library(reporter)
library(magrittr)

# Create temp file name
tmp <- file.path(tempdir(), "mtcars.txt")

# Prepare data
dat <- mtcars[1:10, ]
dat <- data.frame(vehicle = rownames(dat), dat, stringsAsFactors = FALSE)

# Define table
tbl <- create_table(dat, show_cols = 1:8) %>%
  define(vehicle, label = "Vehicle", width = 3, id_var = TRUE, align = "left") %>%
  define(mpg, label = "Miles per Gallon", width = 1) %>%
  define(cyl, label = "Cylinders", format = "%.1f") %>%
  define(disp, label = "Displacement") %>%
  define(hp, label = "Horsepower", page_wrap = TRUE) %>%
  define(drat, visible = FALSE) %>%
  define(wt, label = "Weight") %>%
  define(qsec, label = "Quarter Mile Time", width = 1.5)


# Create the report
rpt <- create_report(tmp, orientation = "portrait") %>%
  titles("Listing 2.0", "MTCARS Data Listing with Page Wrap") %>%
  add_content(tbl, align = "left") %>%
  page_footer(right = "Page [pg] of [tpg]")

# Write the report
write_report(rpt)

# Send report to console for viewing
writeLines(readLines(tmp, encoding = "UTF-8"))

#                                 Listing 2.0
#                       MTCARS Data Listing with Page Wrap
#
```

```
#                                        Miles per
# Vehicle                                   Gallon Cylinders Displacement
# ------------------------------------------------------------------------
# Mazda RX4                                   21       6.0          160
# Mazda RX4 Wag                               21       6.0          160
# Datsun 710                                22.8       4.0          108
# Hornet 4 Drive                            21.4       6.0          258
# Hornet Sportabout                         18.7       8.0          360
# Valiant                                   18.1       6.0          225
# Duster 360                                14.3       8.0          360
# Merc 240D                                 24.4       4.0        146.7
# Merc 230                                  22.8       4.0        140.8
# Merc 280                                  19.2       6.0        167.6
#
# ...
#
#                                                                Page 1 of 2
#                                  Listing 2.0
#                         MTCARS Data Listing with Page Wrap
#
# Vehicle                                 Horsepower Weight  Quarter Mile Time
# ------------------------------------------------------------------------
# Mazda RX4                                    110   2.62              16.46
# Mazda RX4 Wag                                110  2.875              17.02
# Datsun 710                                    93   2.32              18.61
# Hornet 4 Drive                               110  3.215              19.44
# Hornet Sportabout                            175   3.44              17.02
# Valiant                                      105   3.46              20.22
# Duster 360                                   245   3.57              15.84
# Merc 240D                                     62   3.19                 20
# Merc 230                                      95   3.15               22.9
# Merc 280                                     123   3.44               18.3
#
# ...
#
#                                                                Page 2 of 2
```

`footnotes` *Adds a footnote block*

Description

The `footnotes` function adds one or more footnotes to the report. If added to the report specification, the footnotes will be added to the page template, and thus appear on each page of the report. Footnotes may also be added directly to table, text, or plot content.

Usage

```
footnotes(
  x,
```

```
  ...,
  align = "left",
  blank_row = "above",
  borders = "none",
  valign = NULL
)
```

Arguments

x	The object to assign footnotes to.
...	A set of footnote strings.
align	The position to align the footnotes. Valid values are: 'left', 'right', 'center', or 'centre'.
blank_row	Whether to print a blank row above or below the footnote. Valid values are 'above', 'below', 'both', or 'none'. Default is 'above'.
borders	Whether to print a border above or below the footnote. Valid values are 'top', 'bottom', 'all', or 'none'. Default is 'none'. For fixed width reports, the border character will be taken from the value of the uchar parameter on the options_fixed function.
valign	The vertical position to align the footnotes. Valid values are: 'top' and 'bottom'. For footnotes attached to a report, default is 'bottom'. For footnotes attached to content, default is 'top'.

Details

The footnotes function accepts a set of strings of the desired footnote text. The footnotes may be aligned center, left or right using the align parameter. The user is responsible for adding desired footnote symbols. Footnote symbols will not be generated automatically.

If footnotes are assigned to the report, alignment will be oriented to the page body. If footnotes are assigned to a table or text, alignment will be oriented to the edge of the content.

One footnote function accepts up to 25 footnotes. However, multiple footnote blocks may be added to the same object.

Blank rows above or below the footnote block may be controlled using the blank_row parameter.

Each footnote string must fit within the available space. The **reporter** package will not wrap footnotes. If a footnote does not fit within the available space, a warning will be generated and the footnote will be truncated. In these situations, either shorten the footnote or split it into multiple footnotes that each fit within the available space.

Value

The modified report.

See Also

Other report: add_content(), create_report(), options_fixed(), page_by(), page_footer(), page_header(), print.report_spec(), set_margins(), title_header(), titles(), write_report()

Examples

```
library(reporter)
library(magrittr)

# Create a temporary file
tmp <- file.path(tempdir(), "expenses.txt")

# Prepare data
dat <- data.frame(category = rownames(USPersonalExpenditure),
                  USPersonalExpenditure, stringsAsFactors = FALSE)

# Define table
tbl <- create_table(dat) %>%
  titles("Table 1.0", "US Personal Expenditures from 1940 - 1960") %>%
  column_defaults(from = X1940, to = X1960, width = .6, format = "$%.2f") %>%
  define(category, label = "Category") %>%
  define(X1940, label = "1940") %>%
  define(X1945, label = "1945") %>%
  define(X1950, label = "1950") %>%
  define(X1955, label = "1955") %>%
  define(X1960, label = "1960") %>%
  footnotes("* In billions of dollars")

# Define report
rpt <- create_report(tmp, orientation="portrait") %>%
  add_content(tbl)

# Write the report
write_report(rpt)

# Display in console
writeLines(readLines(tmp, encoding = "UTF-8"))

#                               Table 1.0
#               US Personal Expenditures from 1940 - 1960
#
#     Category                1940    1945    1950    1955    1960
#     ------------------------------------------------------------
#     Food and Tobacco      $22.20  $44.50  $59.60  $73.20  $86.80
#     Household Operation   $10.50  $15.50  $29.00  $36.50  $46.20
#     Medical and Health     $3.53   $5.76   $9.71  $14.00  $21.10
#     Personal Care          $1.04   $1.98   $2.45   $3.40   $5.40
#     Private Education      $0.34   $0.97   $1.80   $2.60   $3.64
#
#     * In billions of dollars
```

---

lowcase_parens *Functions to format the population label*

Description

These functions are used to format the "N=" population label on column headers.

Usage

```
lowcase_parens(x)

upcase_parens(x)

lowcase_n(x)

upcase_n(x)
```

Arguments

x	Population count

Details

Which function to use to format the population label is specified on the `n_format` parameter on the `create_table` function. These formatting functions provide several options for formatting the "N=", including whether the "N" should be upper case or lower case, and whether or not to put the value in parentheses. If one of these options does not meet the specifications for your report, you may also write your own formatting function and pass it to the `n_format` function. When an N value is supplied, the output of this function will be concatenated to the header label.

See Also

`create_table` function to create a table.

Examples

```
# Create test data
l <- "Label"
n <- 47

cat(paste0(l, lowcase_parens(n)))
# Label
# (n=47)

cat(paste0(l, upcase_parens(n)))
# Label
# (N=47)

cat(paste0(l, lowcase_n(n)))
# Label
# n=47

cat(paste0(l, upcase_n(n)))
# Label
# N=47
```

```
customN <- function(n) {
  return(paste0(": N=", n))
}
cat(paste0(l, customN(n)))
# Label: N=47
```

NotesOnPDF *Notes on PDF output type*

Description

As of **reporter** v1.1.3, the package will generate PDF files directly, without using **rmarkdown** as an intermediary. This technique eliminates dependencies, makes the package easier to install, and greatly improves performance. It is recommended to use the most recent version of the package if possible.

For versions of **reporter** prior to v1.1.3, the package has additional dependencies and installation requirements. See details below.

Details

For **reporter** v1.1.2 and below, PDF output type requires the **rmarkdown** package and a LaTeX renderer like **MiKTeX**. **MiKTeX** must be installed separately, and will not be installed as part of the **reporter** install. This program must furthermore be added to the `path` environment variable on the system where **reporter** will run, so that **reporter** can find it.

PDF output also supports a limited character set. Some Unicode characters can cause the PDF rendering to fail. Common characters like mathematical operators and Greek letters are supported. However, other special characters and characters from Asian languages may not be supported, depending on the operating system and installed operating system languages.

---

options_fixed *Set options for a fixed-width report*

Description

This function sets the options for a report with a fixed width font.

Usage

```
options_fixed(
  x,
  editor = NULL,
  cpuom = NULL,
  lpuom = NULL,
  min_margin = NULL,
  blank_margins = FALSE,
  font_size = NULL,
  line_size = NULL,
  line_count = NULL,
  uchar = ""
)
```

Arguments

x	The report spec.
editor	The expected text editor to use for printing text reports. Assigning this parameter will set the cpuom and lpuom parameters appropriately for the text editor. Valid values are 'notepad', 'word', 'wordpad', 'notepad++', and 'editplus'. If the editor parameter is used, any settings for cpuom and lpuom will be ignored. It is not necessary to set this parameter for RTF and PDF reports.
cpuom	Characters per unit of measure of printed text. If units is inches, the default is 12. If units is centimeters (cm), the default is 4.687. This value will be used to determine how many characters can fit on a line.
lpuom	Lines per unit of measure of the printed text. Default for inches is 6. The default for centimeters (cm) is 2.55. This value will be used to determine the number of lines that can fit on a page.
min_margin	The editor minimum margin. This parameter normally defaults to 0, but may be set for some types of editors.
blank_margins	When this option is TRUE, **reporter** will use blank spaces and blank rows to create left and top margins, rather than rely on the editor to set margins. When used, editor margins should be set to zero. Valid values are TRUE and FALSE. Default is FALSE. This option is only valid for output_type = 'TXT'.
font_size	The size of the font in points. Default is 10pt. This option is only valid for output types RTF and PDF. Valid values are 8, 10, and 12.
line_size	The number of characters that will fit on a line. Normally, the line_size is calculated based on the page size, font size, and cpuom. You can override the calculated value by setting the line_size directly.
line_count	The number of lines that will fit on page. Normally, the line_count is calculated based on the page size, font size, and lpuom. You can override the calculated value by setting the line_count directly.
uchar	The character to use for underlines on the table header and spanning headers. Default is a Unicode macron character #U00AF. You may use a dash or underscore if your editor does not support Unicode. The uchar is forced to a dash for PDF output, as the LaTeX converter does not support the macron character.

Details

The options_fixed function sets options for reports with a fixed-width, monospace font. These reports are based off a text report, but may be output as type "RTF" or "PDF".

Value

The updated report spec.

Text Reports

The options_fixed function sets the characters per unit of measure (cpuom) and lines per unit of measure (lpuom) settings for the report. These settings determine how many characters and lines will fit within one unit of measure (uom), as specified on the create_report function. These settings are important to ensure the report content stays within the available page size and margins. Because every editor allows a different number of characters and lines on a page, these settings must be adjusted depending on the editor.

The options_fixed function provides a shortcut editor parameter to directly specify a popular editor. If this parameter is specified, the function will set the characters per unit of measure and lines per unit of measure for you. If the editor is not available in the editor parameter selections, for best results, you should set the cpuom and lpuom parameters manually. To determine your cpuom and lpuom, see the help for write_registration_file.

Alternatively, using the options_fixed function, you may set the line_size and line_count directly. Note that the line_size and line_count may be different for different output types and editors.

The min_margin parameter is used to set the minimum margin allowed by the editor. This value will be subtracted from the margin settings when the blank_margins option is used. It is useful for editors that do not calculate margins from the edge of the page.

As some editors do not support Unicode characters, it may be necessary to change the character used for the header and spanning header underlines. The default character is a Unicode #U00AF macron. The macron is sometimes referred to as an "overline", since it is located at the top of the character area. If your editor does not support Unicode, the macron will not be displayed properly. In this case, change the underline character to a dash ("-") or an underscore ("_") using the uchar parameter.

RTF and PDF Reports

For RTF and PDF reports, most of the parameters on the options_fixed function do not apply. For RTF and PDF reports, these parameters will be set automatically, and cannot be changed.

Some of the options_fixed function apply only to RTF and PDF. In particular, the font_size parameter applies only to RTF and PDF reports. Valid font size options are 8, 10, and 12.

See Also

create_report to create a report and set the unit of measure, write_registration_file to determine the characters and lines per unit of measure manually.

Other report: add_content(), create_report(), footnotes(), page_by(), page_footer(), page_header(), print.report_spec(), set_margins(), title_header(), titles(), write_report()

Examples

```
library(reporter)
library(magrittr)

# Create a temporary file
tmp <- file.path(tempdir(), "bod.txt")

# Define table
tbl <- create_table(BOD, width = 2.5) %>%
  titles("Table 3.6", "BOD* Sample Report") %>%
  define(Time, format = "Day %s", align = "left") %>%
  define(demand, format = "%2.1f mg/l", label = "Demand") %>%
  footnotes("* Biochemical Oxygen Demand")

# Define report #1 - No blank margins
rpt <- create_report(tmp, orientation="portrait") %>%
  add_content(tbl, align = "left")

# Write the report
write_report(rpt)

# Write report to console
writeLines(readLines(tmp, encoding = "UTF-8"))

#           Table 3.6
#       BOD* Sample Report
#
# Time                   Demand
# ---------------------------
# Day 1                8.3 mg/l
# Day 2               10.3 mg/l
# Day 3               19.0 mg/l
# Day 4               16.0 mg/l
# Day 5               15.6 mg/l
# Day 7               19.8 mg/l
#
# * Biochemical Oxygen Demand


# Define report #2 - blank margins
rpt <- create_report(tmp, orientation="portrait") %>%
  options_fixed(blank_margins = TRUE) %>%
  set_margins(top = .5, left = 1) %>%
  add_content(tbl, align = "left")

# Write the report
write_report(rpt)

# Write report to console
writeLines(readLines(tmp, encoding = "UTF-8"))

#
```

```
#
#
#                      Table 3.6
#                   BOD* Sample Report
#
#             Time                   Demand
#             ----------------------------
#             Day 1                8.3 mg/l
#             Day 2               10.3 mg/l
#             Day 3               19.0 mg/l
#             Day 4               16.0 mg/l
#             Day 5               15.6 mg/l
#             Day 7               19.8 mg/l
#
#             * Biochemical Oxygen Demand
```

---

page_by *Adds a page by variable*

Description

The page_by function adds a page by variable to a report, table, or plot. The page by will generate a page break for each value of the page by variable. A label will appear above the content showing the page by variable value. You must be sort the data by the page by variable prior to reporting.

Usage

```
page_by(x, var, label = NULL, align = "left", blank_row = "below")
```

Arguments

x	The report specification to assign the page by to.
var	The page by variable. There can be only one page by per report, and one page by variable. The page by can be passed either quoted or unquoted.
label	A label to be used as a prefix to the page by variable value. By default, the label will be assigned to the variable name. Alternatively, you may specify a string value to use for the label.
align	How to align the page by. Default value is 'left'. Valid values are 'left', 'right', 'center', or 'centre'.
blank_row	Indicates whether a blank row is desired above or below the page by. Default value is 'none'. Valid values are 'above', 'below', 'both', or 'none'.

Details

Only one page by is allowed per report, table, or plot. The page by label will appear on all pages of the object. The page by label may be aligned on the left, right, or center. Use the align parameter to specify the alignment.

You must be sort the data by the page by variable prior to reporting. The page by labels will appear in the sorted order. Failure to sort the page by variable prior to reporting may produce unexpected results.

See Also

create_table to create a table, and create_plot to create a plot.

Other report: add_content(), create_report(), footnotes(), options_fixed(), page_footer(), page_header(), print.report_spec(), set_margins(), title_header(), titles(), write_report()

Examples

```
library(reporter)
library(magrittr)

# Create temp file path
tmp <- file.path(tempdir(), "iris.txt")

# Sample and sort iris data frame
dat <- iris[sample(1:150, 50), ]
dat <- dat[order(dat$Species), ]

# Create table
tbl <- create_table(dat) %>%
  page_by(Species, "Species: ") %>%
  define(Species, visible = FALSE)

# Create the report object
rpt <- create_report(tmp, orientation = "portrait") %>%
  page_header("Sponsor: Iris Society", "Study: flowers") %>%
  titles("Table 2.0", "IRIS Sample Report with Page By") %>%
  add_content(tbl) %>%
  page_footer(Sys.time(), right = "Page [pg] of [tpg]")

# Write the report to the file system
write_report(rpt)

# Write report to console
writeLines(readLines(tmp, encoding = "UTF-8"))

# Sponsor: Iris Society                                          Study: flowers
#                                  Table 2.0
#                       IRIS Sample Report with Page By
#
#              Species: setosa
#
#              Sepal.Length Sepal.Width Petal.Length Petal.Width
#              ------------------------------------------------
#                       5.4         3.9          1.7         0.4
#                       4.9         3.1          1.5         0.1
#                       4.8         3.1          1.6         0.2
#                       5.1         3.5          1.4         0.3
```

```
#                         5         3.5          1.6         0.6
#                         5         3.3          1.4         0.2
#                       4.4           3          1.3         0.2
#                       5.1         3.5          1.4         0.2
#                       5.4         3.4          1.5         0.4
#                       4.9         3.6          1.4         0.1
#                       4.6         3.1          1.5         0.2
#                       4.6         3.2          1.4         0.2
#                       5.1         3.3          1.7         0.5
# ...
# 2020-10-25 19:33:35                                        Page 1 of 3
#
# Sponsor: Iris Society                                    Study: flowers
#                                 Table 2.0
#                       IRIS Sample Report with Page By
#
#              Species: versicolor
#
#              Sepal.Length Sepal.Width Petal.Length Petal.Width
#              ------------------------------------------------
#                       4.9         2.4          3.3           1
#                       6.3         3.3          4.7         1.6
#                       6.1         2.8          4.7         1.2
#                         6         2.9          4.5         1.5
#                       6.7           3            5         1.7
#                       5.6           3          4.5         1.5
#                       5.8         2.7          4.1           1
#                       6.7         3.1          4.7         1.5
#                       6.1         2.9          4.7         1.4
#                         5           2          3.5           1
#                       5.9         3.2          4.8         1.8
#                       5.5         2.5            4         1.3
#                         7         3.2          4.7         1.4
#                       6.3         2.3          4.4         1.3
#                       6.1         2.8            4         1.3
#                         6         2.2            4           1
#                       5.5         2.6          4.4         1.2
#                         6         3.4          4.5         1.6
#                         5         2.3          3.3           1
#                       5.5         2.4          3.7           1
# ...
# 2020-10-25 19:33:35                                        Page 2 of 3
#
# Sponsor: Iris Society                                    Study: flowers
#                                 Table 2.0
#                       IRIS Sample Report with Page By
#
#              Species: versicolor
#
#              Sepal.Length Sepal.Width Petal.Length Petal.Width
#              ------------------------------------------------
#                       6.3         3.4          5.6         2.4
#                       7.9         3.8          6.4           2
```

```
#                     6.7         3.1         5.6         2.4
#                     6.2         2.8         4.8         1.8
#                     6.7         3.3         5.7         2.5
#                     6.2         3.4         5.4         2.3
#                     5.6         2.8         4.9           2
#                     7.7         3.8         6.7         2.2
#                     7.7         2.6         6.9         2.3
#                     6.9         3.1         5.4         2.1
#                     6.5         3.2         5.1           2
#                     6.1         2.6         5.6         1.4
#                     5.7         2.5           5           2
#                     6.5           3         5.8         2.2
#                     6.3         2.8         5.1         1.5
#                     7.6           3         6.6         2.1
#                     6.3         2.5           5         1.9
# ...
# 2020-10-25 19:33:35                                    Page 3 of 3
```

`page_footer` *Adds a page footer*

Description

This function adds a page footer to the report. The page footer will appear on each page of the report, at the bottom of the page. The page footer contains three sections: left, center, and right. Content for each section may be specified with the appropriate parameter.

Usage

```
page_footer(x, left = "", center = "", right = "", blank_row = "above")
```

Arguments

`x`	The report spec object.
`left`	The left page footer text. May be a single string or a vector of strings.
`center`	The center page footer text. May be a single string or a vector of strings.
`right`	The right page footer text. May be a single string or a vector of strings.
`blank_row`	Whether to create a blank row above the page footer. Valid values are 'above' and 'none'. Default is 'above'.

Details

Only one page footer is allowed per report. The page footer will appear on all pages of the report. The page footer may contain text on the left, right, or center. Use the appropriate parameters to specify the desired text for each section. Multiple strings may be passed to each section as a vector of strings.

If the width of the page header string exceeds the available space, an error will be generated. In addition, there is a limit of 5 strings for each page footer section.

There are two special tokens to generate page numbers: [pg] and [tpg]. Use [pg] to indicate the current page number. Use [tpg] to indicate the total number of pages in the report. These tokens may be placed anywhere in the page header or page footer.

Use the `blank_row` parameter to control the blank space above the page footer.

Each footer string must fit within the available space. The **reporter** package will not wrap footer. If a footer string does not fit within the available space, an error will be generated. In these situations, either shorten the footer string or split it into multiple footers that each fit within the available space.

Value

The modified report.

See Also

Other report: `add_content()`, `create_report()`, `footnotes()`, `options_fixed()`, `page_by()`, `page_header()`, `print.report_spec()`, `set_margins()`, `title_header()`, `titles()`, `write_report()`

Examples

```
library(reporter)
library(magrittr)

# Create temp file path
tmp <- file.path(tempdir(), "mtcars.txt")

dat <- data.frame(name = rownames(mtcars[1:10, ]), mtcars[1:10, 1:5],
                  stringsAsFactors = FALSE)

# Create the report object
rpt <- create_report(tmp, orientation = "portrait") %>%
  page_header("Client: Motor Trend", "Study: Cars") %>%
  titles("MTCARS Sample Report") %>%
  add_content(create_table(dat)) %>%
  page_footer(Sys.time(), right = "Page [pg] of [tpg]")

# Write the report to the file system
write_report(rpt)

# Write report to console
writeLines(readLines(tmp, encoding = "UTF-8"))

# Client: Motor Trend                                              Study: Cars
#                               MTCARS Sample Report
#
#                name                 mpg cyl   disp   hp  drat
#                ------------------------------------------------
#                Mazda RX4             21   6    160  110   3.9
#                Mazda RX4 Wag         21   6    160  110   3.9
#                Datsun 710          22.8   4    108   93  3.85
#                Hornet 4 Drive      21.4   6    258  110  3.08
#                Hornet Sportabout   18.7   8    360  175  3.15
```

```
#                Valiant            18.1   6    225  105  2.76
#                Duster 360         14.3   8    360  245  3.21
#                Merc 240D          24.4   4  146.7   62  3.69
#                Merc 230           22.8   4  140.8   95  3.92
#                Merc 280           19.2   6  167.6  123  3.92
#
# ...
#
# 2020-10-17 11:53:51                                    Page 1 of 1
```

---

page_header *Add a page header*

Description

This function adds a page header to the report. The page header will appear at the top of each page of the report.

Usage

```
page_header(x, left = "", right = "", blank_row = "none")
```

Arguments

x	The report object.
left	The left page header text. May be a single string or a vector of strings.
right	The right page header text. May be a single string or a vector of strings.
blank_row	Whether to create a blank row below the page header. Valid values are 'below' and 'none'. Default is 'none'.

Details

The page header may contain text on the left or right. Use the appropriate parameters to specify the desired text. Only one page header is allowed on a report. The page header will be repeated on every page of the report. Multiple text values for each side may be specified as a vector of strings.

If the width of the page header string exceeds the available space, an error will be generated. There is also a limit of 5 page header strings per each side.

There are two special tokens to generate page numbers: [pg] and [tpg]. Use [pg] to indicate the current page number. Use [tpg] to indicate the total number of pages in the report. These tokens may be placed anywhere in the page header or page footer.

Each header string must fit within the available space. The **reporter** package will not wrap headers. If a header string does not fit within the available space, an error will be generated. In these situations, either shorten the header string or split it into multiple headers that each fit within the available space.

Value

The modified report specification.

See Also

Other report: add_content(), create_report(), footnotes(), options_fixed(), page_by(), page_footer(), print.report_spec(), set_margins(), title_header(), titles(), write_report()

Examples

```
library(magrittr)

# Create temp file path
tmp <- file.path(tempdir(), "mtcars.txt")

dat <- data.frame(name = rownames(mtcars[1:10, ]), mtcars[1:10, 1:5],
                  stringsAsFactors = FALSE)

# Create the report object
rpt <- create_report(tmp, orientation = "portrait") %>%
  page_header("Client: Motor Trend", "Study: Cars") %>%
  titles("MTCARS Sample Report") %>%
  add_content(create_table(dat)) %>%
  page_footer(Sys.time(), right = "Page [pg] of [tpg]")

# Write the report to the file system
write_report(rpt)

# Write report to console
writeLines(readLines(tmp, encoding = "UTF-8"))

# Client: Motor Trend                                          Study: Cars
#                               MTCARS Sample Report
#
#                name                  mpg cyl   disp   hp  drat
#                ---------------------------------------------
#                Mazda RX4              21   6    160  110   3.9
#                Mazda RX4 Wag          21   6    160  110   3.9
#                Datsun 710           22.8   4    108   93  3.85
#                Hornet 4 Drive       21.4   6    258  110  3.08
#                Hornet Sportabout    18.7   8    360  175  3.15
#                Valiant              18.1   6    225  105  2.76
#                Duster 360           14.3   8    360  245  3.21
#                Merc 240D            24.4   4  146.7   62  3.69
#                Merc 230             22.8   4  140.8   95  3.92
#                Merc 280             19.2   6  167.6  123  3.92
#
# ...
#
# 2020-10-17 11:53:51                                          Page 1 of 1
```

print.plot_spec *Prints the plot spec*

Description

A function to print the plot spec. The **print** function will print the plot spec in summary form. To view all parameters, set the verbose parameter to TRUE.

Usage

```
## S3 method for class 'plot_spec'
print(x, ..., verbose = FALSE)
```

Arguments

x	The plot spec.
...	Additional parameters to pass to the underlying print function.
verbose	Whether to print in verbose form. Default is FALSE.

Value

The plot spec, invisibly.

See Also

create_plot function to create a plot specification.

Other plot: create_plot()

Examples

```
txt <- create_text("Lorem ipsum dolor sit amet, consectetur...")
print(txt)

# A text specification:
# - text: data.frame 'mtcars' 32 rows 11 cols
```

`print.report_spec` *Prints the report specification*

Description

A function to print the report specification. The **print** function will print the report spec in summary form by default. To print in list form, set the `verbose` parameter to TRUE.

Usage

```
## S3 method for class 'report_spec'
print(x, ..., verbose = FALSE)
```

Arguments

`x`	The report spec.
`...`	Additional parameters to pass to the underlying print function.
`verbose`	Whether to print the report object in verbose (list) form or summary form. The default is FALSE.

Value

The report spec, invisibly.

See Also

`create_report` function to create a report specification.

Other report: `add_content()`, `create_report()`, `footnotes()`, `options_fixed()`, `page_by()`, `page_footer()`, `page_header()`, `set_margins()`, `title_header()`, `titles()`, `write_report()`

Examples

```
library(reporter)
library(magrittr)

# Create temp file path
tmp <- file.path(tempdir(), "mtcars.txt")

# Create the table
tbl <- create_table(mtcars) %>%
  titles("Table 1.0", "MTCARS Sample Report") %>%
  footnotes("* NOTE: Data from 1974")

# Create the report object
rpt <- create_report(tmp) %>%
  add_content(tbl, align = "left")

# Write the report to the file system
```

```
res <- write_report(rpt)

# Write the modified report object to the console
print(res)

# # A report specification: 1 pages
# - file_path: 'C:/Users/User/AppData/Local/Temp/RtmpWQybXs/mtcars.txt'
# - output_type: TXT
# - units: inches
# - orientation: landscape
# - line size/count: 108/45
# - content:
# # A table specification:
# - data: data.frame 'mtcars' 32 rows 11 cols
# - show_cols: all
# - use_attributes: all
# - title 1: 'Table 1.0'
# - title 2: 'MTCARS Sample Report'
# - footnote 1: '* NOTE: Data from 1974'
```

`print.table_spec` *Prints the table spec*

Description

A function to print the table spec. The **print** function will print the table spec in summary form. To view all parameters, set the `verbose` parameter to TRUE.

Usage

```
## S3 method for class 'table_spec'
print(x, ..., verbose = FALSE)
```

Arguments

`x`	The table spec.
`...`	Additional parameters to pass to the underlying print function.
`verbose`	Whether to print in verbose form, which is similar to a list. Default is FALSE, which prints in summary form.

Value

The table spec, invisibly.

See Also

`create_table` function to create a table specification.

Other table: `column_defaults()`, `create_table()`, `define()`, `spanning_header()`, `stub()`

Examples

```
library(magrittr)

# Create Table
tbl <- create_table(mtcars) %>%
  define(mpg, label = "Miles Per Gallon", width = .5) %>%
  define(cyl, label = "Cylinders") %>%
  titles("Table 6.4", "MTCARS Sample Table") %>%
  footnotes("* Motor Trend, 1974")

tbl

# A table specification:
# - data: data.frame 'mtcars' 32 rows 11 cols
# - show_cols: all
# - use_attributes: all
# - title 1: 'Table 6.4'
# - title 2: 'MTCARS Sample Table'
# - footnote 1: '* Motor Trend, 1974'
# - define: mpg 'Miles Per Gallon' width=0.5
# - define: cyl 'Cylinders'
```

---

print.text_spec *Prints the text spec*

Description

A function to print the text spec. The **print** function will print the text spec in summary form. To view all parameters, set the `verbose` parameter to TRUE.

Usage

```
## S3 method for class 'text_spec'
print(x, ..., verbose = FALSE)
```

Arguments

`x`	The text spec.
`...`	Additional parameters to pass to the underlying print function.
`verbose`	Whether to print in verbose form. Default is FALSE.

Value

The text spec, invisibly.

See Also

`create_text` function to create a text specification.

Other text: `create_text()`

Examples

```
txt <- create_text("Lorem ipsum dolor sit amet, consectetur...",
                   align = "left", width = 3)
txt

# A text specification: 6 words
# - text: Lorem ipsum dolor sit amet, consectetur...
# - width: 3
# - align: left
```

`reporter` *reporter: A package for creating statistical reports*

Description

The **reporter** package creates statistical reports in TXT, RTF, and PDF file formats. Features include automatic page wrapping and breaking for wide and long tables, spanning headers, titles, footnotes, page headers, page footers, and page numbering. The package allows mixing of multiple tables, text, and plots in a single report, or even on a single page.

Details

The **reporter** package creates regulatory-style, statistical reports. It was designed to create tables, listings, and figures (TLFs) for use in the pharmaceutical, biotechnology, and medical-device industries. However, the functions are generalized enough to provide statistical reporting for any industry. The package is written primarily in Base R, and has few dependencies on other packages.

The package is intended to give R programmers flexible report layout capabilities, and a choice of output formats. The package will initially focus on printable, file-based output formats, as there are already numerous R packages that provide tabular reporting in HTML.

PDF output may have limitations not associated with TXT and RTF. See NotesOnPDF for more information.

Key Features

The **reporter** package contains the following key features:

- Titles, footnotes, page header, and page footer are repeated on each page
- Supports header labels and spanning headers
- Calculates default columns widths automatically
- Includes automatic wrapping and splitting of wide and long tables
- Integrates with the **fmtr** package to format numeric, date, and character data
- Plots from the popular **ggplot2** package can be added to RTF and PDF reports
- Allows appending multiple tables to a report, multiple tables to a page, and intermingling of text, tables, and plots
- Supports in-report date/time stamps and "Page X of Y" page numbering

Key Functions

- `create_report` to define a report
- `create_table` to define a table
- `create_text` to define text content
- `create_plot` to define a plot
- `write_report` to write out the report

Package Assumptions and Limitations

Note that the **reporter** package is built on several assumptions, and has some limitations. Those assumptions and limitations are as follows:

- The package assumes that displaying the data in the proper layout is more important than aesthetic considerations like colors, borders, fonts, striping, etc.
- The current version supports a monospace, fixed-width font only. Variable width fonts will be supported in future versions.
- RTF and PDF output are actually text reports with an RTF or PDF wrapper. Future versions will support native RTF and PDF file formats.
- The package assumes that, except for formatting and layout, the incoming data is ready for printing. The **reporter** package has no capabilities to perform calculations, summaries, grouping, or derivations. Use the many capabilities in R to perform these operations prior to sending data to the **reporter** package.
- It is assumed that the incoming data is sorted as desired. The **reporter** package has no capabilities for sorting. Use the sorting functionality in Base R or supplemental packages to sort the data prior to sending to **reporter**.
- Titles, footnotes, page headers, and page footers must fit in the available space. If they don't, the **reporter** package will generate a warning. In these situations, the recommended course of action is to split the offending string into two or more strings that fit within the available width.
- The **reporter** package will never break a word. That means you cannot set a column width that is less than the length of the longest word. If you wish to break words, add the breaks with `strwrap` or an equivalent function before reporting.
- The package will make a best guess on column widths based on the width of the data and column headers. To override the best guess, set the column width directly by placing a width attribute on the variable or by using the `column_defaults` or `define` functions.
- The max automatic column width is 5 inches. Longer data values will be wrapped.
- The package support plots from **ggplot2**. These plots can be added to RTF and PDF output types. The package does not support Base R plots.

set_margins *Set page margins*

Description

Sets the page margins for the report. The units for this parameter can be inches or centimeters, depending on the units of measure specified on the create_report function.

Usage

```
set_margins(x, top = NULL, bottom = NULL, left = NULL, right = NULL)
```

Arguments

x	The report spec object.
top	The top margin.
bottom	The bottom margin.
left	The left margin.
right	The right margin.

Details

The margins set with set_margins will be used for the entire report. Units for the margins are specified by the units parameter on the create_report function. Available units are 'inches' and 'cm'. When the unit of measure is inches, default margins are 1 inch on the left and right, and .5 inches on top and bottom. When the unit of measure is centimeters, default margins are 2.54 cm on left and right, and 1.27 cm on top and bottom.

Note that when using output type of TXT, and not using the blank_margins option, setting the margins only reduces the area available for content on a page. You must still set the actual margins on the available editor to match those specified in set_margins. Any mismatch may result in content not fitting properly on the page. For best results, set the right and bottom margins to zero to allow for slight overflow without causing a page break or wrapping lines.

Value

The report_spec with margins set as desired.

See Also

Other report: add_content(), create_report(), footnotes(), options_fixed(), page_by(), page_footer(), page_header(), print.report_spec(), title_header(), titles(), write_report()

Examples

```
library(reporter)
library(magrittr)

# Create a temporary file
tmp <- file.path(tempdir(), "bod.txt")

# Define table
tbl <- create_table(BOD, width = 2.5) %>%
  titles("Table 3.6", "BODÂ¹ Sample Report") %>%
  define(Time, format = "Day %s", align = "left") %>%
  define(demand, format = "%2.1f mg/l", label = "Demand") %>%
  footnotes("Â¹ Biochemical Oxygen Demand")

# Define report #1 - No blank margins
rpt <- create_report(tmp, orientation="portrait") %>%
  add_content(tbl, align = "left")

# Write the report
write_report(rpt)

# Write report to console
writeLines(readLines(tmp, encoding = "UTF-8"))

#           Table 3.6
#       BOD* Sample Report
#
# Time                   Demand
# ---------------------------
# Day 1                8.3 mg/l
# Day 2               10.3 mg/l
# Day 3               19.0 mg/l
# Day 4               16.0 mg/l
# Day 5               15.6 mg/l
# Day 7               19.8 mg/l
#
# * Biochemical Oxygen Demand


# Define report #2 - blank margins
rpt <- create_report(tmp, orientation="portrait") %>%
  options_fixed(blank_margins = TRUE) %>%
  set_margins(top = .5, left = 1) %>%
  add_content(tbl, align = "left")

# Write the report
write_report(rpt)

# Write report to console
writeLines(readLines(tmp, encoding = "UTF-8"))

#
```

```
#
#
#                      Table 3.6
#                   BOD* Sample Report
#
#             Time                   Demand
#             ----------------------------
#             Day 1                8.3 mg/l
#             Day 2               10.3 mg/l
#             Day 3               19.0 mg/l
#             Day 4               16.0 mg/l
#             Day 5               15.6 mg/l
#             Day 7               19.8 mg/l
#
#             * Biochemical Oxygen Demand
```

`spanning_header` *Defines a spanning header*

Description

Create a header that spans multiple columns. Spanning headers are used to group related columns. Such groupings are a common feature of statistical reports.

Usage

```
spanning_header(
  x,
  from,
  to,
  label = "",
  label_align = "center",
  level = 1,
  n = NULL,
  underline = TRUE,
  standard_eval = FALSE
)
```

Arguments

`x`	The table object to add spanning headers to.
`from`	The starting column to span. Spanning columns are defined as range of columns 'from' and 'to'. The columns may be identified by position, or by quoted or unquoted variable names. If you want to pass the `from` value using an R variable, set the `standard_eval` parameter to TRUE. The `from` parameter is required.
`to`	The ending column to span. Spanning columns are defined as range of columns 'from' and 'to'. The columns may be identified by position, or by quoted or unquoted variable names. If you want to pass the `to` value using an R variable, set the `standard_eval` parameter to TRUE. The `to` parameter is required.

label	The label to apply to the spanning header.
label_align	The alignment to use for the label. Valid values are "left", "right", "center", and "centre". The default for spanning columns is "center".
level	The level to use for the spanning header. The lowest spanning level is level 1, the next level above is level 2, and so on. By default, the level is set to 1.
n	The population count to use for the "N=" label on the spanning header. The "N=" label will be formatted according to the n_format parameter on the create_table function.
underline	A TRUE or FALSE value indicating whether the spanning header should be underlined. Default is TRUE.
standard_eval	A TRUE or FALSE value that indicates whether to use standard or non-standard evaluation of the from, and to parameters. Set standard_eval to TRUE if you want to pass the column names as variables. Default is FALSE, meaning it will use non-standard (unquoted) evaluation.

Details

A spanning header is a label and underline that spans one or more columns. A spanning header is defined minimally by identifying the column range to be spanned, and a label. A label alignment and "N=" value may also be supplied.

The spanning column range is defined by the from and to parameters. The range identifies a contiguous set of variables on the data. Variables can be identified by position, a quoted variable name, or an unquoted variable name.

Value

The modified table spec.

See Also

Other table: column_defaults(), create_table(), define(), print.table_spec(), stub()

Examples

```
library(reporter)
library(magrittr)

# Create a temporary file
tmp <- file.path(tempdir(), "iris.txt")

# Prepare data
dat <- iris[sample(1:150, 15), c(5, 1, 2, 3, 4)]
dat <- dat[order(dat$Species), ]

# Define table
tbl <- create_table(dat) %>%
  titles("Table 3.2", "IRIS Sample Report") %>%
  spanning_header(2, 3, label = "Sepal") %>%
  spanning_header(4, 5, label = "Petal") %>%
```

```
  column_defaults(2:5, format = "%.1f") %>%
  define(Species, align = "left", dedupe = TRUE, blank_after = TRUE) %>%
  define(Sepal.Length, label = "Length") %>%
  define(Sepal.Width, label = "Width") %>%
  define(Petal.Length, label = "Length") %>%
  define(Petal.Width, label = "Width") %>%
  footnotes("* From Fisher's Iris Dataset")

# Define report
rpt <- create_report(tmp, orientation="portrait") %>%
  options_fixed(blank_margins = TRUE) %>%
  set_margins(top = 1, bottom =1) %>%
  add_content(tbl, align = "left")

# Write the report
write_report(rpt)

writeLines(readLines(tmp, encoding = "UTF-8"))

#
#
#
#
#                     Table 3.2
#                  IRIS Sample Report
#
#                       Sepal        Petal
#                    ------------ ------------
#       Species     Length Width Length Width
#       -------------------------------------
#       setosa         5.0   3.0    1.6   0.2
#                      4.6   3.4    1.4   0.3
#                      5.0   3.4    1.6   0.4
#                      5.7   3.8    1.7   0.3
#
#       versicolor     5.7   2.8    4.1   1.3
#                      6.2   2.9    4.3   1.3
#                      7.0   3.2    4.7   1.4
#                      6.6   2.9    4.6   1.3
#
#       virginica      6.2   3.4    5.4   2.3
#                      7.2   3.0    5.8   1.6
#                      6.9   3.1    5.1   2.3
#                      5.6   2.8    4.9   2.0
#                      7.7   2.6    6.9   2.3
#                      6.3   2.8    5.1   1.5
#                      7.7   2.8    6.7   2.0
#
#
#       * From Fisher's Iris Dataset
```

stub *Defines a report stub*

Description

Combine columns into a nested report stub. The report stub is a common feature of statistical reports. The stub is created with the `stub` function, and frequently appears in combination with the `label_row` and `indent` parameters from the `define` function. These elements work together to define the appearance of the stub.

Usage

```
stub(
  x,
  vars,
  label = "",
  label_align = NULL,
  align = "left",
  width = NULL,
  standard_eval = FALSE
)
```

Arguments

`x`	The table spec.
`vars`	A vector of quoted or unquoted variable names from which to create the stub. If you want to pass an R variable of names, escape the values with double curly braces, i.e. `vars = {{myvar}}`. The curly brace escape is useful when writing functions that construct reports dynamically.
`label`	The label for the report stub. The default label is an empty string.
`label_align`	The alignment for the stub column label. Valid values are 'left', 'right', 'center', and 'centre'. Default follows the `align` parameter.
`align`	How to align the stub column. Valid values are 'left', 'right', 'center', and 'centre'. Default is 'left'.
`width`	The width of the stub, in report units of measure.
`standard_eval`	A TRUE or FALSE value that indicates whether to use standard or non-standard evaluation of the `vars`, `from`, and `to` parameters. Set `standard_eval` to TRUE if you want to pass the column names as variables. Default is FALSE, meaning it will use non-standard (unquoted) evaluation.

Details

The table stub is a nested set of labels that identify rows on the table. The stub is created by combining two or more columns into a single stub column. The relationship between the columns is typically visualized as a hierarchy, with lower level concepts indented under higher level concepts.

A typical stub is created with the following steps:

- Prepare the data.
- Create the table object.
- Define the stub on the table using the `stub` function, and identify the variables to be combined.
- Identify higher level concepts with the `label_row` parameter on the `define` function.
- Identify lower level concepts using the `indent` parameter on the `define` function.

The stub will be automatically added as an identity variable on the report, and will always appear as the leftmost column. There can only be one stub defined on a report.

If you wish to create multiple levels of nested labels, use an NA value to prevent lower level labels from overwriting higher level labels.

For example, the following data:

```
continent          country   state_province
"North America"    NA        NA
"North America"    "Canada"  NA
"North America"    "Canada"  "Ontario"
"North America"    "USA"     NA
"North America"    "USA"     "New York"
"South America"    NA        NA
"South America"    "Brazil"  NA
"South America"    "Brazil"  "Amazonas"
"South America"    "Brazil"  "Bahia"
```

Will produce the following stub:

```
North America
  Canada
    Ontario
  USA
    New York
South America
  Brazil
    Amazonas
    Bahia
```

With the following code:

```
tbl <- create_table(dat) %>%
  stub(c(continent, country, state_province)) %>%
  define(country, indent = .25) %>%
  define(state_province, indent = .5)
```

Value

The modified table spec.

See Also

Other table: `column_defaults()`, `create_table()`, `define()`, `print.table_spec()`, `spanning_header()`

Examples

```
library(reporter)
library(magrittr)

# Create temporary path
tmp <- file.path(tempdir(), "stub.txt")

# Read in prepared data
df <- read.table(header = TRUE, text = '
      var      label        A             B
      "ampg"   "N"          "19"          "13"
      "ampg"   "Mean"       "18.8 (6.5)"  "22.0 (4.9)"
      "ampg"   "Median"     "16.4"        "21.4"
      "ampg"   "Q1 - Q3"    "15.1 - 21.2" "19.2 - 22.8"
      "ampg"   "Range"      "10.4 - 33.9" "14.7 - 32.4"
      "cyl"    "8 Cylinder" "10 ( 52.6%)" "4 ( 30.8%)"
      "cyl"    "6 Cylinder" "4 ( 21.1%)"  "3 ( 23.1%)"
      "cyl"    "4 Cylinder" "5 ( 26.3%)"  "6 ( 46.2%)"')

# Create table
tbl <- create_table(df, first_row_blank = TRUE) %>%
  stub(c(var, label)) %>%
  define(var, blank_after = TRUE, label_row = TRUE,
         format = c(ampg = "Miles Per Gallon", cyl = "Cylinders")) %>%
  define(label, indent = .25) %>%
  define(A, label = "Group A", align = "center", n = 19) %>%
  define(B, label = "Group B", align = "center", n = 13)


# Create report and add content
rpt <- create_report(tmp, orientation = "portrait") %>%
  page_header(left = "Client: Motor Trend", right = "Study: Cars") %>%
  titles("Table 1.0", "MTCARS Summary Table") %>%
  add_content(tbl) %>%
  footnotes("* Motor Trend, 1974") %>%
  page_footer(left = Sys.time(),
              center = "Confidential",
              right = "Page [pg] of [tpg]")

# Write out report
write_report(rpt)

# View report in console
writeLines(readLines(tmp, encoding = "UTF-8"))

# Client: Motor Trend                                                  Study: Cars
#                                  Table 1.0
#                              MTCARS Summary Table
#
#                                     Group A      Group B
#                                      (N=19)       (N=13)
#                 ------------------------------------------
```

```
#
#                 Miles Per Gallon
#                    N                    19           13
#                    Mean              18.8 (6.5)   22.0 (4.9)
#                    Median               16.4         21.4
#                    Q1 - Q3           15.1 - 21.2  19.2 - 22.8
#                    Range             10.4 - 33.9  14.7 - 32.4
#
#                 Cylinders
#                    8 Cylinder       10 ( 52.6%)   4 ( 30.8%)
#                    6 Cylinder        4 ( 21.1%)   3 ( 23.1%)
#                    4 Cylinder        5 ( 26.3%)   6 ( 46.2%)
#
# ...
#
#
# * Motor Trend, 1974
#
# 2020-08-30 03:50:02              Confidential                    Page 1 of 1
#
```

`titles` *Adds a title block*

Description

This function adds one or more titles to an object as a title block. If added to a report, the titles will be added to the page template, and thus appear on each page of the report. Titles may also be added to a table, text, or plot object.

Usage

```
titles(x, ..., align = "center", blank_row = "below", borders = "none")
```

Arguments

`x`	The object to assign titles to. Valid objects are a report, or a table, text, or plot specification.
`...`	A set of title strings.
`align`	The position to align the titles. Valid values are 'left', 'right', 'center' or 'centre'. For titles, the default is 'center'.
`blank_row`	Where to place a blank row. Valid values are 'above', 'below', 'both', or 'none'. Default is "below".
`borders`	Whether and where to place a border. Valid values are 'top', 'bottom', 'all', or 'none'. Default is "none".

Details

The titles function accepts a set of strings of the desired title text. To specify multiple titles for the block, pass them to the function as separate strings.

The titles may be aligned center, left or right using the align parameter. The alignment will be applied to all titles contained in the block. To control alignment of titles separately for each title, use multiple titles functions.

Titles may be assigned to a report, a table, a text specification, or a plot. If assigned to the report, the title will appear at the top of the page, and be repeated for every page of the report. If the titles are assigned to content, the titles will appear above the content, and be repeated if the content breaks to the next page.

If titles are assigned to the report, alignment will be oriented to the page body. If titles are assigned to content, alignment will be oriented to the edge of the content.

One title function accepts up to 10 titles. However, multiple title blocks may be added to the same object if needed.

Blank rows above or below the title block may be controlled using the `blank_row` parameter.

Each title string must fit within the available space. The **reporter** package will not wrap titles. If a title does not fit within the available space, a warning will be generated and the title will be truncated. In these situations, either shorten the title or split it into multiple titles that each fit within the available space.

Value

The modified report.

See Also

Other report: `add_content()`, `create_report()`, `footnotes()`, `options_fixed()`, `page_by()`, `page_footer()`, `page_header()`, `print.report_spec()`, `set_margins()`, `title_header()`, `write_report()`

Examples

```
library(reporter)
library(magrittr)

# Create a temporary file
tmp <- file.path(tempdir(), "expenses.txt")

# Prepare data
dat <- data.frame(category = rownames(USPersonalExpenditure),
                  USPersonalExpenditure, stringsAsFactors = FALSE)

# Define table
tbl <- create_table(dat) %>%
  titles("Table 1.0", "US Personal Expenditures from 1940 - 1960") %>%
  column_defaults(from = X1940, to = X1960, width = .6, format = "$%.2f") %>%
  define(category, label = "Category") %>%
  define(X1940, label = "1940") %>%
  define(X1945, label = "1945") %>%
```

```
  define(X1950, label = "1950") %>%
  define(X1955, label = "1955") %>%
  define(X1960, label = "1960") %>%
  footnotes("* In billions of dollars")

# Define report
rpt <- create_report(tmp, orientation="portrait") %>%
  add_content(tbl)

# Write the report
write_report(rpt)

# Display in console
writeLines(readLines(tmp, encoding = "UTF-8"))
#                               Table 1.0
#              US Personal Expenditures from 1940 - 1960
#
#     Category                1940    1945    1950    1955    1960
#     ------------------------------------------------------------
#     Food and Tobacco      $22.20  $44.50  $59.60  $73.20  $86.80
#     Household Operation   $10.50  $15.50  $29.00  $36.50  $46.20
#     Medical and Health     $3.53   $5.76   $9.71  $14.00  $21.10
#     Personal Care          $1.04   $1.98   $2.45   $3.40   $5.40
#     Private Education      $0.34   $0.97   $1.80   $2.60   $3.64
#
#     * In billions of dollars
```

`title_header` *Adds a title header block*

Description

This function adds a title header to an object. A title header is a special type of title layout that has titles on the left and header information on the right.

Usage

```
title_header(x, ..., right = "", blank_row = "below", borders = "none")
```

Arguments

`x`	The object to assign titles to. Valid objects are a report, table, text, or plot specification.
`...`	A set of title strings.
`right`	A set of header strings to be shown on the right side of the title header. Pass the header strings as a vector of strings.
`blank_row`	Where to place a blank row. Valid values are 'above', 'below', 'both', or 'none'. Default is 'below'.
`borders`	Whether and where to place a border. Valid values are 'top', 'bottom', 'all', or 'none'. Default is 'none'.

Details

The `title_header` function accepts a set of strings of the desired title text, and a vector of header strings. The titles will appear on the left of the title header, and the header strings on the right. To specify multiple titles for the block, pass them to the function as separate strings.

Title headers may be assigned to a report, a table, a text specification, or a plot. If assigned to the report, the title header will appear at the top of the page, and be repeated for every page of the report. If the title header is assigned to content, the titles will appear above the content, and be repeated if the content breaks to the next page.

One title header function accepts up to 10 titles. Blank rows above or below the title block may be controlled using the `blank_row` parameter.

Each title string must fit within the available space. The **reporter** package will not wrap titles. If a title does not fit within the available space, an error will be generated. In these situations, either shorten the title or split it into multiple titles that each fit within the available space.

Value

The modified report.

See Also

Other report: `add_content()`, `create_report()`, `footnotes()`, `options_fixed()`, `page_by()`, `page_footer()`, `page_header()`, `print.report_spec()`, `set_margins()`, `titles()`, `write_report()`

Examples

```
library(reporter)
library(magrittr)

# Create a temporary file
tmp <- file.path(tempdir(), "expenses.txt")

# Prepare data
dat <- data.frame(category = rownames(USPersonalExpenditure),
                  USPersonalExpenditure, stringsAsFactors = FALSE)

# Define table
tbl <- create_table(dat) %>%
  title_header("Table 1.0", "US Personal Expenditures from 1940 - 1960",
               right = c("Page [pg] of [tpg]", "World Almanac")) %>%
  column_defaults(from = X1940, to = X1960, width = .6, format = "$%.2f") %>%
  define(category, label = "Category") %>%
  define(X1940, label = "1940") %>%
  define(X1945, label = "1945") %>%
  define(X1950, label = "1950") %>%
  define(X1955, label = "1955") %>%
  define(X1960, label = "1960") %>%
  footnotes("* In billions of dollars")

# Define report
rpt <- create_report(tmp, orientation="portrait") %>%
```

```
  add_content(tbl)

# Write the report
write_report(rpt)

# Display in console
writeLines(readLines(tmp, encoding = "UTF-8"))

#     Table 1.0                                          Page 1 of 1
#     US Personal Expenditures from 1940 - 1960      World Almanac
#
#     Category                1940    1945    1950    1955    1960
#     ------------------------------------------------------------
#     Food and Tobacco      $22.20  $44.50  $59.60  $73.20  $86.80
#     Household Operation   $10.50  $15.50  $29.00  $36.50  $46.20
#     Medical and Health     $3.53   $5.76   $9.71  $14.00  $21.10
#     Personal Care          $1.04   $1.98   $2.45   $3.40   $5.40
#     Private Education      $0.34   $0.97   $1.80   $2.60   $3.64
#
#     * In billions of dollars
```

`write_registration_file` *Create a registration file*

Description

This function will create a registration file to help determine the correct `cpuom` and `lpuom` for your editor/printer.

Usage

```
write_registration_file(file_path)
```

Arguments

`file_path`	The full or relative file name and path to create the registration file.

Details

The `cpi` and `lpi` are used in `output_type = "TXT"` to determine available space on the page. The registration file can help determine the correct settings for the target text editor and printer. Failure to set the correct characters per unit of measure (cpuom) and lines per unit of measure (lpuom) may result in misalignment of content on the page when printing text output.

How to Use the Registration File

To use the registration file, first decide the units of measure you wish to use, inches or centimeters. Next, create the registration file by calling the `write_registration_file` function. Then print the registration file.

Once the registration file is printed, take a ruler and measure both the horizontal and vertical registration lines from zero to 60 in the desired units of measure. For example, if your units of measure is 'inches', measure the registration lines in inches.

Record the distance measured in each direction. For each direction, divide 60 by the distance measured, and round to three decimal places. The horizontal result is the characters per unit of measure (cpuom). The vertical result is the lines per unit of measure (lpuom). To get an accurate printing of text reports, assign these values to the `cpuom` and `lpuom` parameters on the `options_fixed` function.

For best results, test the calculated values by printing some reports and checking for undesired page breaks or wrapped lines. If necessary, adjust the calculated cpuom and lpuom values until all content stays within the available space without wrapping or breaking.

Examples

```
library(reporter)

# Create temp file path
tmp <- file.path(tempdir(), "reg.txt")

# Create the registration file
write_registration_file(tmp)

# Write registration file to the console
writeLines(readLines(tmp))

# 0--------+---------+---------+---------+---------+---------+
# -       10        20        30        40        50        60
# -
# -
# -
# -
# -
# -
# -
# + 10
# -
# -
# -
# -
# -
# -
# -
# -
# -
# + 20
# -
```

```
# -
# -
# -
# -
# -
# -
# -
# -
# + 30
# -
# -
# -
# -
# -
# -
# -
# -
# -
# + 40
# -
# -
# -
# -
# -
# -
# -
# -
# -
# + 50
# -
# -
# -
# -
# -
# -
# -
# -
# -
# + 60
```

`write_report` *Write a report to the file system*

Description

This function writes a report_spec object to the file system, using the specifications provided in the object.

Usage

```
write_report(x, file_path = NULL, output_type = NULL, preview = NULL)
```

Arguments

x	The report object to write.
file_path	The file name and path to write the report to. If supplied, this parameter overrides the file_path parameter on the create_report function. Default is NULL.
output_type	The output file type. This parameter will override the output_type on the create_report function. This parameter can be used to output the same report object to multiple output types. Default value is NULL, meaning it will not override the create_report value. Valid values are 'TXT', 'RTF', and 'PDF'.
preview	Whether to write the entire report, or a report preview. A report preview is a subset of pages of the report. The default value is NULL, meaning the entire report will be written. You may also pass a number of pages to write. For example, passing the number 1 will print the first page, while passing a 5 will print the first five pages.

Details

The function renders the report in the requested format, and writes it to the location specified in the report file_path parameter. Attempts to write an object that is not of class "report_spec" will generate an error.

The write_report function is a driver for very complex set of rendering functions. The rendering functions perform most of the advanced functionality of the **reporter** package: generating spanning headers, page wrapping and breaking, creating stub columns, etc. When things go wrong, they will usually go wrong during this function call. For that reason, although this function can be part of the pipeline that creates the report object, it is best to call write_report independently, to help isolate any issues from the report definition procedure.

Value

The report spec, with settings modified during rendering. These modified settings can sometimes be useful for documentation, and for debugging issues with the procedure.

See Also

Other report: add_content(), create_report(), footnotes(), options_fixed(), page_by(), page_footer(), page_header(), print.report_spec(), set_margins(), title_header(), titles()

Examples

```
library(reporter)
library(fmtr)
library(magrittr)

# Create temp file path
tmp <- file.path(tempdir(), "beaver2.txt")

# Take Sample of Data
dat <- beaver2[sample(1:100, 15), ]
```

```
# Create format for active variable
fmt <- value(condition(x == 0, "No"),
             condition(x == 1, "Yes"))

# Create the table
tbl <- create_table(dat) %>%
  titles("Table 1.0", "BEAVERS Sample Report") %>%
  column_defaults(width = .75) %>%
  define(day, label = "Day", format = "Day %s") %>%
  define(time, label = "Time") %>%
  define(temp, label = "Temperature", width = 1, format = "%.1f") %>%
  define(activ,label = "Active", format = fmt) %>%
  footnotes("* NOTE: Data on beaver habits")

# Create the report object
rpt <- create_report(tmp) %>%
  add_content(tbl, align = "left")

# Write the report to the file system
res <- write_report(rpt)

# Write the modified report object to the console
print(res)

# Write the report to console
writeLines(readLines(tmp, encoding = "UTF-8"))

#                 Table 1.0
#           BEAVERS Sample Report
#
#      Day      Time  Temperature    Active
# ----------------------------------------
#  Day 307      1020         37.2        No
#  Day 307      1030         37.2        No
#  Day 307       940         36.7        No
#  Day 307      1340         37.1        No
#  Day 307      1410         37.2        No
#  Day 307      1400         37.1        No
#  Day 307      1130         36.9        No
#  Day 307      1140         37.0        No
#  Day 307      1120         37.0        No
#  Day 307      1000         37.1        No
#  Day 307      1250         37.0        No
#  Day 307      2100         37.9       Yes
#  Day 307      1210         37.0        No
#  Day 307      1740         38.0       Yes
#  Day 308       130         37.8       Yes
#
# * NOTE: Data on beaver habits
```

Index