Package ‘officer’

June 13, 2022

Type Package
Title Manipulation of Microsoft Word and PowerPoint Documents
Version 0.4.3
Description Access and manipulate 'Microsoft Word' and 'Microsoft PowerPoint' documents from R. The package focuses on tabular and graphical reporting from R; it also provides two functions that let users get document content into data objects. A set of functions lets add and remove images, tables and paragraphs of text in new or existing documents. The package does not require any installation of Microsoft products to be able to write Microsoft files.
License GPL-3
Imports R6, grDevices, stats, graphics, utils, zip (>= 2.1.0), uuid (>= 0.1-4), xml2 (>= 1.1.0)
Encoding UTF-8
BugReports https://github.com/davidgohel/officer/discussions
RoxygenNote 7.2.0
Suggests testthat, devEMF, ggplot2, rmarkdown, base64enc, knitr, rsvg
NeedsCompilation no
Author David Gohel [aut, cre], Frank Hangler [ctb] (function body_replace_all_text), Liz Sander [ctb] (several documentation fixes), Anton Victorson [ctb] (fixes xml structures), Jon Calder [ctb] (update vignettes), John Harrold [ctb] (function annotate_base), John Muschelli [ctb] (google doc compatibility), Bill Denney [ctb] (<https://orcid.org/0000-0002-5759-428X>, function as.matrix.rpptx), Nikolai Beck [ctb] (set speaker notes for .pptx documents), Stefan Moog [ctb] (added functionality to set shape geometry and outline), Greg Leleu [ctb] (fields functionality in ppt)
Maintainer  David Gohel <david.gohel@ardata.fr>
Repository  CRAN
Date/Publication  2022-06-12 22:20:02 UTC

R topics documented:

- add_sheet ........................................... 4
- add_slide ........................................... 5
- annotate_base ...................................... 5
- as.matrix.rpptx .................................... 6
- block_caption ...................................... 7
- block_list ......................................... 8
- block_pour_docx ................................... 9
- block_section ..................................... 10
- block_table ......................................... 11
- block_toc .......................................... 12
- body_add_blocks ................................... 12
- body_add_break .................................... 13
- body_add_caption .................................. 14
- body_add_docx ..................................... 15
- body_add_fpar ..................................... 16
- body_add_gg ....................................... 17
- body_add_img ...................................... 18
- body_add_par ...................................... 19
- body_add_plot ..................................... 20
- body_add_table .................................... 21
- body_add_toc ...................................... 22
- body_bookmark ..................................... 23
- body_end_block_section ............................ 23
- body_end_section_columns ....................... 24
- body_end_section_columns_landscape ............ 25
- body_end_section_continuous .................... 26
- body_end_section_landscape ..................... 27
- body_end_section_portrait ...................... 28
- body_remove ........................................ 28
- body_replace_all_text ............................. 29
- body_replace_text_at_bkm ........................ 31
- body_set_default_section ....................... 32
- change_styles ..................................... 33
- color_scheme ....................................... 34
- cursor_begin ...................................... 35
- docx_bookmarks .................................... 36
- docx_dim ........................................... 37
- docx_show_chunk .................................. 38
- docx_summary ...................................... 39
- doc_properties .................................... 39
- empty_content ...................................... 40
R topics documented:

external_img .................................................. 41
fpar ............................................................... 42
fp_border ........................................................ 43
fp_cell ............................................................ 44
fp_par ............................................................. 46
fp_text ............................................................ 47
ftext ............................................................... 50
hyperlink_ftext .................................................. 51
layout_properties ............................................... 52
layout_summary ................................................ 52
length.rdocx ..................................................... 53
length.rpptx ..................................................... 54
media_extract ................................................... 54
move_slide ......................................................... 55
notes_location_label ......................................... 56
notes_location_type ........................................... 56
officer ............................................................. 57
officer-defunct .................................................. 58
on_slide ........................................................... 58
page_mar ........................................................... 59
page_size .......................................................... 60
ph_hyperlink ....................................................... 61
ph_location ........................................................ 62
ph_location_fullsize .......................................... 63
ph_location_label .............................................. 64
ph_location_left ............................................... 65
ph_location_right ............................................. 66
ph_location_template ......................................... 66
ph_location_type ............................................... 68
ph_remove ........................................................ 70
ph_slidelink ....................................................... 71
ph_with ............................................................ 72
plot_instr ........................................................ 76
plot_layout_properties ....................................... 77
pptx_summary .................................................... 78
print.rpptx ........................................................ 79
prop_section ...................................................... 79
prop_table ........................................................ 81
read_docx ........................................................ 82
read_pptx ........................................................ 83
read_xlsx ........................................................ 84
remove_slide ..................................................... 85
run_autonum ...................................................... 85
run_bookmark ..................................................... 87
run_columnbreak ............................................... 88
run_footnote ...................................................... 88
run_footnoteref .................................................. 89
run_linebreak ..................................................... 90
add_sheet

Description

Add a sheet into an xlsx worksheet.

Usage

    add_sheet(x, label)

Arguments

    x             rxlsx object
    label         sheet label

Examples

    my_ws <- read_xlsx()
    my_pres <- add_sheet(my_ws, label = "new sheet")
add_slide

Add a slide

Description

Add a slide into a pptx presentation.

Usage

add_slide(x, layout = "Title and Content", master = "Office Theme")

Arguments

x # an rpptx object
layout # slide layout name to use
master # master layout name where layout is located

See Also

print.rpptx(), read_pptx(), plot_layout_properties(), ph_with(), layout_summary()

Other functions slide manipulation: move_slide(), on_slide(), remove_slide(), set_notes()

Examples

my_pres <- read_pptx()
layout_summary(my_pres)
my_pres <- add_slide(my_pres,
layout = "Two Content", master = "Office Theme")

annotate_base

Placeholder parameters annotation

Description

generates a slide from each layout in the base document to identify the placeholder indexes, types, names, master names and layout names.

This is to be used when need to know what parameters should be used with ph_location* calls. The parameters are printed in their corresponding shapes.

Note that if there are duplicated ph_label, you should not use ph_location_label.

Usage

annotate_base(path = NULL, output_file = "annotated_layout.pptx"
Arguments

path       path to the pptx file to use as base document or NULL to use the officer default
output_file filename to store the annotated powerpoint file or NULL to suppress generation

Value

rpptx object of the annotated PowerPoint file

See Also

Other functions for reading presentation informations: color_scheme(), doc_properties(), layout_properties(), layout_summary(), length.rpptx(), plot_layout_properties(), slide_size(), slide_summary()

Examples

# To generate an annotation of the default base document with officer:
annotate_base(output_file = tempfile(fileext = ".pptx"))

# To generate an annotation of the base document 'mydoc.pptx' and place the
# annotated output in 'mydoc_annotate.pptx'
# annotate_base(path = 'mydoc.pptx', output_file='mydoc_annotate.pptx')
id

The table ID to load from (ignored if `is.null(slide_id)`), NA indicates to load the first table from the `slide_id`.

span

How should `col_span`/`row_span` values be handled? NA means to leave the value as NA, and "fill" means to fill matrix cells with the value.

Value

A matrix with the data, or if `slide_id=NULL`, a list of matrices.

Examples

```r
library(officer)
pptx_file <- system.file(package="officer", "doc_examples", "example.pptx")
z <- read_pptx(pptx_file)
as.matrix(z, slide_id = NULL)
```

block_caption

Create a representation of a caption that can be used for cross reference.

Description

Create a representation of a caption that can be used for cross reference.

Usage

```r
block_caption(label, style, autonum = NULL)
```

Arguments

- `label` a scalar character representing label to display
- `style` paragraph style name
- `autonum` an object generated with function `run_autonum`

See Also

Other block functions for reporting: `block_list()`, `block_pour_docx()`, `block_section()`, `block_table()`, `block_toc()`, `fpar()`, `plot_instr()`, `unordered_list()`

Examples

```r
library(officer)
run_num <- run_autonum(seq_id = "tab", pre_label = "tab.", bkm = "mtcars_table")
caption <- block_caption("mtcars table", style = "Normal", autonum = run_num)
```
block_list

List of blocks

Description

A list of blocks can be used to gather several blocks (paragraphs, tables, ...) into a single object. The result can be added into a Word document or a PowerPoint presentation.

Usage

block_list(...)

Arguments

...  
a list of blocks. When output is only for Word, objects of class `external_img()` can also be used in fpar construction to mix text and images in a single paragraph. Supported objects are: `block_caption()`, `block_pour_docx()`, `block_section()`, `block_table()`, `block_toc()`, `fpar()`, `plot_instr()`.

See Also

`ph_with()`, `body_add_blocks()`, `fpar()`

Other block functions for reporting: `block_caption()`, `block_pour_docx()`, `block_section()`, `block_table()`, `block_toc()`, `fpar()`, `plot_instr()`, `unordered_list()`

Examples

# block list ------

```
img.file <- file.path( R.home("doc"), "html", "logo.jpg" )
fpt_blue_bold <- fp_text(color = "#006699", bold = TRUE)
fpt_red_italic <- fp_text(color = "#C32900", italic = TRUE)

## This can be only be used in a MS word output as pptx does not support paragraphs made of text and images.
## (actually it can be used but image will not appear in the pptx output)
value <- block_list(
  fpar(ftext("hello world", fpt_blue_bold)),
```
Pour the content of a docx file in the resulting docx from an 'R Markdown' document.

### Usage

```r
block_pour_docx(file)
```

### Arguments

- **file**
  - external docx file path

### See Also

Other block functions for reporting: `block_caption()`, `block_list()`, `block_section()`, `block_table()`, `block_toc()`, `fpar()`, `plot_instr()`, `unordered_list()`
Examples

```r
library(officer)
docx <- tempfile(fileext = ".docx")
doc <- read_docx()
doc <- body_add(doc, iris[1:20,], style = "table_template")
print(doc, target = docx)

target <- tempfile(fileext = ".docx")
doc_1 <- read_docx()
doc_1 <- body_add(doc_1, block_pour_docx(docx))
print(doc_1, target = target)
```

Description

Create a representation of a section.

A section affects preceding paragraphs or tables; i.e. a section starts at the end of the previous section (or the beginning of the document if no preceding section exists), and stops where the section is declared.

When a new landscape section is needed, it is recommended to add a `block_section` with `type = "continuous"`, to add the content to be appended in the new section and finally to add a `block_section` with `page_size = page_size(orient = "landscape")`.

Usage

`block_section(property)`

Arguments

- `property` section properties defined with function `prop_section`

See Also

Other block functions for reporting: `block_caption()`, `block_list()`, `block_pour_docx()`, `block_table()`, `block_toc()`, `fpar()`, `plot_instr()`, `unordered_list()`

Examples

```r
ps <- prop_section(
  page_size = page_size(orient = "landscape"),
  page_margins = page_mar(top = 2),
  type = "continuous"
)
block_section(ps)
```
Description

Create a representation of a table

Usage

block_table(x, header = TRUE, properties = prop_table(), alignment = NULL)

Arguments

x
  a data.frame to add as a table
header
  display header if TRUE
properties
  table properties, see prop_table(). Table properties are not handled identically between Word and PowerPoint output format. They are fully supported with Word but for PowerPoint (which does not handle as many things as Word for tables), only conditional formatting properties are supported.
alignment
  alignment for each columns, 'l' for left, 'r' for right and 'c' for center. Default to NULL.

See Also

prop_table()

Other block functions for reporting: block_caption(), block_list(), block_pour_docx(), block_section(), block_toc(), fpar(), plot_instr(), unordered_list()

Examples

block_table(x = head(iris))

block_table(x = mtcars, header = TRUE, properties = prop_table(
  tcf = table_conditional_formatting(
    first_row = TRUE, first_column = TRUE)
))
block_toc    Table of content for 'Word'

Description
Create a representation of a table of content for Word documents.

Usage
block_toc(level = 3, style = NULL, seq_id = NULL, separator = ";")

Arguments
level: max title level of the table
style: optional. If not NULL, its value is used as style in the document that will be used to build entries of the TOC.
seq_id: optional. If not NULL, its value is used as sequence identifier in the document that will be used to build entries of the TOC. See also run_autonum() to specify a sequence identifier.
separator: optional. Some configurations need ";" (i.e. from Canada) separator instead of ",".

See Also
Other block functions for reporting: block_caption(), block_list(), block_pour_docx(), block_section(), block_table(), fpar(), plot_instr(), unordered_list()

Examples
block_toc(level = 2)
block_toc(style = "Table Caption")

body_add_blocks    Add a list of blocks into a 'Word' document

Description
add a list of blocks produced by block_list into an rdocx object.

Usage
body_add_blocks(x, blocks, pos = "after")
Arguments

x an rdocx object
blocks set of blocks to be used as footnote content returned by function block_list().
pos where to add the new element relative to the cursor, one of "after", "before", "on".

See Also

Other functions for adding content: body_add_break(), body_add_caption(), body_add_docx(),
body_add_fpar(), body_add_gg(), body_add_img(), body_add_par(), body_add_plot(), body_add_table(),
body_add_toc()

Examples

library(officer)

img.file <- file.path( R.home("doc"), "html", "logo.jpg" )

bl <- block_list(
  fpar(ftext("hello", shortcuts$fp_bold(color="red"))),
  fpar(
    ftext("hello world", shortcuts$fp_bold()),
    external_img(src = img.file, height = 1.06, width = 1.39),
    fp_p = fp_par(text.align = "center")
  )
)

doc_1 <- read_docx()
doc_1 <- body_add_blocks(doc_1, blocks = bl)
print(doc_1, target = tempfile(fileext = ".docx"))
```r

See Also

Other functions for adding content: `body_add_blocks()`, `body_add_caption()`, `body_add_docx()`, `body_add_fpar()`, `body_add_gg()`, `body_add_img()`, `body_add_par()`, `body_add_plot()`, `body_add_table()`, `body_add_toc()`

Examples

```r
doc <- read_docx()
doc <- body_add_break(doc)
print(doc, target = tempfile(fileext = ".docx"))
```

```
body_add_caption

Add Word caption in a 'Word' document

Description

Add a Word caption into an rdocx object.

Usage

`body_add_caption(x, value, pos = "after")`

Arguments

- `x`: an rdocx object
- `value`: an object returned by `block_caption()`
- `pos`: where to add the new element relative to the cursor, one of "after", "before", "on".

See Also

Other functions for adding content: `body_add_blocks()`, `body_add_break()`, `body_add_docx()`, `body_add_fpar()`, `body_add_gg()`, `body_add_img()`, `body_add_par()`, `body_add_plot()`, `body_add_table()`, `body_add_toc()`

Examples

```r
doc <- read_docx()

if( capabilities(what = "png") )
    doc <- body_add_plot(doc,
        value = plot_instr(
            code = {barplot(1:5, col = 2:6)},
            style = "centered" )
    run_num <- run_autonum(seq_id = "fig", pre_label = "Figure ",
        bkm = "barplot")
caption <- block_caption("a barplot", style = "Normal",
        autonum = run_num )
```
**body_add_docx**

```r
doc <- body_add_caption(doc, caption)
print(doc, target = tempfile(fileext = "docx"))
```

---

**Add an external docx in a 'Word' document**

**Description**

Add content of a docx into an rdocx object.

**Usage**

```r
body_add_docx(x, src, pos = "after")
```

**Arguments**

- `x`: an rdocx object
- `src`: docx filename
- `pos`: where to add the new element relative to the cursor, one of "after", "before", "on".

**Note**

The function is using a 'Microsoft Word' feature: when the document will be edited, the content of the file will be inserted in the main document.

This feature is unlikely to work as expected if the resulting document is edited by another software.

**See Also**

Other functions for adding content: `body_add_blocks()`, `body_add_break()`, `body_add_caption()`, `body_add_fpar()`, `body_add_gg()`, `body_add_img()`, `body_add_par()`, `body_add_plot()`, `body_add_table()`, `body_add_toc()`

**Examples**

```r
file1 <- tempfile(fileext = "docx")
file2 <- tempfile(fileext = "docx")
file3 <- tempfile(fileext = "docx")

x <- read_docx()
x <- body_add_par(x, "hello world 1", style = "Normal")
print(x, target = file1)

x <- read_docx()
x <- body_add_par(x, "hello world 2", style = "Normal")
print(x, target = file2)

x <- read_docx(path = file1)
x <- body_add_break(x)
x <- body_add_docx(x, src = file2)
print(x, target = file3)
```
**Description**

Add an fpar (a formatted paragraph) into an rdocx object.

**Usage**

```r
body_add_fpar(x, value, style = NULL, pos = "after")
```

**Arguments**

- `x`: a docx device
- `value`: a character
- `style`: paragraph style. If NULL, paragraph settings from fpar will be used. If not NULL, it must be a paragraph style name (located in the template provided as `read_docx(path = ...)`); in that case, paragraph settings from fpar will be ignored.
- `pos`: where to add the new element relative to the cursor, one of "after", "before", "on".

**See Also**

fpar

Other functions for adding content: `body_add_blocks()`, `body_add_break()`, `body_add_caption()`, `body_add_docx()`, `body_add_gg()`, `body_add_img()`, `body_add_par()`, `body_add_plot()`, `body_add_table()`, `body_add_toc()`

**Examples**

```r
bold_face <- shortcuts$fpar_bold(font.size = 30)
bolt_redface <- update(bold_face, color = "red")
fpar_ <- fpar(ftext("Hello ", prop = bold_face),
              ftext("World", prop = bold_redface ),
              ftext(" , how are you?", prop = bold_face ) )
doc <- read_docx()
doc <- body_add_fpar(doc, fpar_)
print(docx)
```

# a way of using fpar to center an image in a Word doc ----
```r
rlogo <- file.path( R.home("doc"), "html", "logo.jpg" )
img_in_par <- fpar(
                       external_img(src = rlogo, height = 1.06/2, width = 1.39/2),
                       hyperlink_ftext(
                           href = "https://cran.r-project.org/index.html",
```
body_add_gg

```r
text = "cran", prop = bold_redface),
fp_p = fp_par(text.align = "center")

doc <- read_docx()
doc <- body_add_fpar(doc, img_in_par)
print(doc, target = tempfile(fileext = ".docx") )
```

---

**body_add_gg**  
*Add a 'ggplot' in a 'Word' document*

**Description**

add a ggplot as a png image into an rdocx object.

**Usage**

```r
body_add_gg(
  x,
  value,
  width = 6,
  height = 5,
  res = 300,
  style = "Normal",
  scale = 1,
  ...
)
```

**Arguments**

- `x`  
an rdocx object
- `value`  
ggplot object
- `width`  
height in inches
- `height`  
height in inches
- `res`  
resolution of the png image in ppi
- `style`  
paragraph style
- `scale`  
Multiplicative scaling factor, same as in ggsave
- `...`  
Arguments to be passed to png function.

**See Also**

Other functions for adding content: body_add_blocks(), body_add_break(), body_add_caption(), body_add_docx(), body_add_fpar(), body_add_img(), body_add_par(), body_add_plot(), body_add_table(), body_add_toc()
Examples

```r
if( require("ggplot2") ){  
  doc <- read_docx()

  gg_plot <- ggplot(data = iris ) +  
  geom_point(mapping = aes(Sepal.Length, Petal.Length))

  if( capabilities(what = "png") )  
    doc <- body_add_gg(doc, value = gg_plot, style = "centered")

  print(doc, target = tempfile(fileext = ".docx") )
}
```

---

**body_add_img**

*Add an image in a 'Word' document*

**Description**

add an image into an rdocx object.

**Usage**

```r
body_add_img(x, src, style = NULL, width, height, pos = "after")
```

**Arguments**

- `x`  
  an rdocx object

- `src`  
  image filename, the basename of the file must not contain any blank.

- `style`  
  paragraph style

- `width`  
  height in inches

- `height`  
  height in inches

- `pos`  
  where to add the new element relative to the cursor, one of "after", "before", "on".

**See Also**

Other functions for adding content: `body_add_blocks()`, `body_add_break()`, `body_add_caption()`, `body_add_docx()`, `body_add_fpar()`, `body_add_gg()`, `body_add_par()`, `body_add_plot()`, `body_add_table()`, `body_add_toc()`
Examples

doc <- read_docx()

img.file <- file.path(R.home("doc"), "html", "logo.jpg")
if( file.exists(img.file) ){
  doc <- body_add_img(x = doc, src = img.file, height = 1.06, width = 1.39)
}

print(doc, target = tempfile(fileext = "docx"))

---

body_add_par  Add paragraphs of text in a 'Word' document

Description

add a paragraph of text into an rdocx object

Usage

body_add_par(x, value, style = NULL, pos = "after")

Arguments

x  a docx device
value  a character
style  paragraph style name
pos  where to add the new element relative to the cursor, one of "after", "before", "on".

See Also

Other functions for adding content: body_add_blocks(), body_add_break(), body_add_caption(),
body_add_docx(), body_add_fpar(), body_add_gg(), body_add_img(), body_add_plot(), body_add_table(), body_add_toc()

Examples

doc <- read_docx()
doc <- body_add_par(doc, "A title", style = "heading 1")
doc <- body_add_par(doc, "Hello world!", style = "Normal")
doc <- body_add_par(doc, "centered text", style = "centered")

print(doc, target = tempfile(fileext = "docx") )
Description

Add a plot as a png image into an rdocx object.

Usage

```r
body_add_plot(
  x,
  value,
  width = 6,
  height = 5,
  res = 300,
  style = "Normal",
  ...
)
```

Arguments

- `x`: an rdocx object
- `value`: plot instructions, see `plot_instr()`.
- `width`: height in inches
- `height`: height in inches
- `res`: resolution of the png image in ppi
- `style`: paragraph style
- `...`: Arguments to be passed to png function.

See Also

Other functions for adding content: `body_add_blocks()`, `body_add_break()`, `body_add_caption()`, `body_add_docx()`, `body_add_fpar()`, `body_add_gg()`, `body_add_img()`, `body_add_par()`, `body_add_table()`, `body_add_toc()`

Examples

```r
doc <- read_docx()

if( capabilities(what = "png") )
  doc <- body_add_plot(doc,
    value = plot_instr(
      code = {barplot(1:5, col = 2:6)},
      style = "centered" )
  )

print(doc, target = tempfile(fileext = ".docx") )
```
Add table in a ‘Word’ document

Description

Add a table into an rdocx object.

Usage

```r
doctools::body_add_table(
  x,  
  value, 
  style = NULL, 
  pos = "after", 
  header = TRUE, 
  alignment = NULL, 
  align_table = "center", 
  stylenames = table_stylenames(), 
  first_row = TRUE, 
  first_column = FALSE, 
  last_row = FALSE, 
  last_column = FALSE, 
  no_hband = FALSE, 
  no_vband = TRUE)
```

Arguments

- `x`: a docx device
- `value`: a data.frame to add as a table
- `style`: table style
- `pos`: where to add the new element relative to the cursor, one of after", "before", "on".
- `header`: display header if TRUE
- `alignment`: columns alignment, argument length must match with columns length, values must be "l" (left), "r" (right) or "c" (center).
- `align_table`: table alignment within document, value must be "left", "center" or "right"
- `stylenames`: columns styles defined by `table_stylenames()`
- `first_row`: Specifies that the first column conditional formatting should be applied. Details for this and other conditional formatting options can be found at http://officeopenxml.com/WPtblLook.php
- `first_column`: Specifies that the first column conditional formatting should be applied.
- `last_row`: Specifies that the first column conditional formatting should be applied.
- `last_column`: Specifies that the first column conditional formatting should be applied.
- `no_hband`: Specifies that the first column conditional formatting should be applied.
- `no_vband`: Specifies that the first column conditional formatting should be applied.
See Also

Other functions for adding content: `body_add_blocks()`, `body_add_break()`, `body_add_caption()`, `body_add_docx()`, `body_add_fpar()`, `body_add_gg()`, `body_add_img()`, `body_add_par()`, `body_add_plot()`, `body_add_table()`, `body_add_toc()`

Examples

```r
doc <- read_docx()
doc <- body_add_table(doc, iris, style = "table_template")

print(doc, target = tempfile(fileext = ".docx")
```

---

### body_add_toc

Add table of content in a 'Word' document

#### Description

Add a table of content into an rdocx object. The TOC will be generated by Word, if the document is not edited with Word (i.e. Libre Office) the TOC will not be generated.

#### Usage

```r
body_add_toc(x, level = 3, pos = "after", style = NULL, separator = ";")
```

#### Arguments

- `x` an rdocx object
- `level` max title level of the table
- `pos` where to add the new element relative to the cursor, one of "after", "before", "on".
- `style` optional. style in the document that will be used to build entries of the TOC.
- `separator` optional. Some configurations need "," (i.e. from Canada) separator instead of ";".

#### See Also

Other functions for adding content: `body_add_blocks()`, `body_add_break()`, `body_add_caption()`, `body_add_docx()`, `body_add_fpar()`, `body_add_gg()`, `body_add_img()`, `body_add_par()`, `body_add_plot()`, `body_add_table()`

#### Examples

```r
doc <- read_docx()
doc <- body_add_toc(doc)

print(doc, target = tempfile(fileext = ".docx")
```
**body_bookmark**

Add a bookmark in a 'Word' document

**Description**

Add a bookmark at the cursor location. The bookmark is added on the first run of text in the current paragraph.

**Usage**

```r
doc <- read_docx()
doc <- body_add_par(doc, "centered text", style = "centered")
doc <- body_bookmark(doc, "text_to_replace")
```

**Arguments**

- **x**: an rdocx object
- **id**: bookmark name

**Examples**

```r
# cursor_bookmark ----
```

**body_end_block_section**

Add any section

**Description**

Add a section to the document. You can define any section with a block_section object. All other `body_end_section_*` are specialized, this one is highly flexible but it’s up to the user to define the section properties.

**Usage**

```r
doc <- read_docx()
doc <- body_add_par(doc, "centered text", style = "centered")
doc <- body_bookmark(doc, "text_to_replace")
```

**Arguments**

- **x**: an rdocx object
- **value**: a block_section object
Illustrations

See Also

Other functions for Word sections: `body_end_section_columns_landscape()`, `body_end_section_columns()`, `body_end_section_continuous()`, `body_end_section_landscape()`, `body_end_section_portrait()`, `body_set_default_section()`

Examples

```r
library(officer)
str1 <- "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
str1 <- rep(str1, 20)
str1 <- paste(str1, collapse = " ")

ps <- prop_section(
  page_size = page_size(orient = "landscape"),
  page_margins = page_mar(top = 2),
  type = "continuous"
)

doc_1 <- read_docx()
doc_1 <- body_add_par(doc_1, value = str1, style = "Normal")
doc_1 <- body_add_par(doc_1, value = str1, style = "Normal")

doc_1 <- body_end_block_section(doc_1, block_section(ps))
doc_1 <- body_add_par(doc_1, value = str1, style = "centered")

print(doc_1, target = tempfile(fileext = ".docx"))
```

---

**body_end_section_columns**

*Add multi columns section*

**Description**

A section with multiple columns is added to the document.

You may prefer to use `body_end_block_section()` that is more flexible.

**Usage**

```r
body_end_section_columns(x, widths = c(2.5, 2.5), space = 0.25, sep = FALSE)
```
body_end_section_columns_landscape

Arguments

- **x**: an rdocx object
- **widths**: columns widths in inches. If 3 values, 3 columns will be produced.
- **space**: space in inches between columns.
- **sep**: if TRUE a line is separating columns.

See Also

Other functions for Word sections: `body_end_block_section()`, `body_end_section_columns_landscape()`, `body_end_section_continuous()`, `body_end_section_landscape()`, `body_end_section_portrait()`, `body_set_default_section()`

Examples

```r
str1 <- "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
str1 <- rep(str1, 5)
str1 <- paste(str1, collapse = " ")

doc_1 <- read_docx()
doc_1 <- body_add_par(doc_1, value = str1, style = "Normal")
doc_1 <- body_add_par(doc_1, value = str1, style = "Normal")
doc_1 <- body_end_section_columns(doc_1)
doc_1 <- body_add_par(doc_1, value = str1, style = "Normal")
print(doc_1, target = tempfile(fileext = ".docx"))
```

---

**body_end_section_columns_landscape**

*Add a landscape multi columns section*

Description

A landscape section with multiple columns is added to the document.

Usage

```r
body_end_section_columns_landscape(
  x,
  widths = c(2.5, 2.5),
  space = 0.25,
  sep = FALSE,
  w = 21/2.54,
  h = 29.7/2.54
)
```
body_end_section_continuous

Arguments

x          an rdocx object
widths     columns widths in inches. If 3 values, 3 columns will be produced.
space      space in inches between columns.
sep         if TRUE a line is separating columns.
w, h        page width, page height (in inches)

See Also

Other functions for Word sections: body_end_block_section(), body_end_section_columns(), body_end_section_continuous(), body_end_section_landscape(), body_end_section_portrait(), body_set_default_section()

Examples

str1 <- "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
str1 <- rep(str1, 5)
str1 <- paste(str1, collapse = " ")

doc_1 <- read_docx()
doc_1 <- body_add_par(doc_1, value = str1, style = "Normal")
doc_1 <- body_add_par(doc_1, value = str1, style = "Normal")
doc_1 <- body_end_section_columns_landscape(doc_1, widths = c(6, 2))
doc_1 <- body_add_par(doc_1, value = str1, style = "Normal")
print(doc_1, target = tempfile(fileext = ".docx"))

body_end_section_continuous

Add continuous section

Description

Section break starts the new section on the same page. This type of section break is often used to change the number of columns without starting a new page.

Usage

body_end_section_continuous(x)

Arguments

x          an rdocx object

See Also

Other functions for Word sections: body_end_block_section(), body_end_section_columns_landscape(), body_end_section_columns(), body_end_section_landscape(), body_end_section_portrait(), body_set_default_section()
Examples

```r
str1 <- "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
str1 <- rep(str1, 5)
str1 <- paste(str1, collapse = " ")
str2 <- "Aenean venenatis varius elit et fermentum vivamus vehicula."
str2 <- rep(str2, 5)
str2 <- paste(str2, collapse = " ")

doc_1 <- read_docx()
doc_1 <- body_add_par(doc_1, value = "Default section", style = "heading 1")
doc_1 <- body_add_par(doc_1, value = str1, style = "Normal")
doc_1 <- body_add_par(doc_1, value = str2, style = "Normal")
doc_1 <- body_end_section_continuous(doc_1)

print(doc_1, target = tempfile(fileext = ".docx"))
```

---

**body_end_section_landscape**

*Add landscape section*

**Description**

A section with landscape orientation is added to the document.

**Usage**

```r
body_end_section_landscape(x, w = 21/2.54, h = 29.7/2.54)
```

**Arguments**

- `x` an `rdocx` object
- `w, h` page width, page height (in inches)

**See Also**

Other functions for Word sections: `body_end_block_section()`, `body_end_section_columns_landscape()`, `body_end_section_columns()`, `body_end_section_continuous()`, `body_end_section_portrait()`, `body_set_default_section()`

**Examples**

```r
str1 <- "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
str1 <- rep(str1, 5)
str1 <- paste(str1, collapse = " ")

doc_1 <- read_docx()
doc_1 <- body_add_par(doc_1, value = str1, style = "Normal")
doc_1 <- body_end_section_landscape(doc_1)

print(doc_1, target = tempfile(fileext = ".docx"))
```
**body_end_section_portrait**  

*Add portrait section*

**Description**

A section with portrait orientation is added to the document.

**Usage**

```
body_end_section_portrait(x, w = 21/2.54, h = 29.7/2.54)
```

**Arguments**

- `x` an rdocx object
- `w`, `h` page width, page height (in inches)

**See Also**

Other functions for Word sections: `body_end_block_section()`, `body_end_section_columns_landscape()`, `body_end_section_columns()`, `body_end_section_continuous()`, `body_end_section_landscape()`, `body_set_default_section()`

**Examples**

```r
str1 <- "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
str1 <- rep(str1, 5)
str1 <- paste(str1, collapse = " ")

doc_1 <- read_docx()
doc_1 <- body_add_par(doc_1, value = str1, style = "Normal")
doc_1 <- body_end_section_portrait(doc_1)
doc_1 <- body_add_par(doc_1, value = str1, style = "Normal")
print(doc_1, target = tempfile(fileext = ".docx"))
```

---

**body_remove**  

*Remove an element in a 'Word' document*

**Description**

Remove element pointed by cursor from a 'Word' document.

**Usage**

```
body_remove(x)
```
Arguments

x  
an rdocx object

Examples

library(officer)

str1 <- rep("Lorem ipsum dolor sit amet, consectetur adipiscing elit. ", 20)
str1 <- paste(str1, collapse = "")

str2 <- "Drop that text"

str3 <- rep("Aenean venenatis varius elit et fermentum vivamus vehicula. ", 20)
str3 <- paste(str3, collapse = "")

my_doc <- read_docx()
my_doc <- body_add_par(my_doc, value = str1, style = "Normal")
my_doc <- body_add_par(my_doc, value = str2, style = "centered")
my_doc <- body_add_par(my_doc, value = str3, style = "Normal")

new_doc_file <- print(my_doc,
  target = tempfile(fileext = ".docx"))

my_doc <- read_docx(path = new_doc_file)
my_doc <- cursor_reach(my_doc, keyword = "that text")
my_doc <- body_remove(my_doc)

print(my_doc, target = tempfile(fileext = ".docx"))

---

body_replace_all_text  Replace text anywhere in the document

Description

Replace text anywhere in the document, or at a cursor.

Replace all occurrences of old_value with new_value. This method uses grepl/gsub for pattern matching; you may supply arguments as required (and therefore use regex features) using the optional ... argument.

Note that by default, grepl/gsub will use fixed=FALSE, which means that old_value and new_value will be interpreted as regular expressions.

Chunking of text

Note that the behind-the-scenes representation of text in a Word document is frequently not what you might expect! Sometimes a paragraph of text is broken up (or "chunked") into several "runs," as a result of style changes, pauses in text entry, later revisions and edits, etc. If you have not styled the text, and have entered it in an "all-at-once" fashion, e.g. by pasting it or by outputting it programmatically into your Word document, then this will likely not be a problem. If you are
working with a manually-edited document, however, this can lead to unexpected failures to find text.

You can use the officer function `docx_show_chunk` to show how the paragraph of text at the current cursor has been chunked into runs, and what text is in each chunk. This can help troubleshoot unexpected failures to find text.

**Usage**

```r
body_replace_all_text(
  x,
  old_value,
  new_value,
  only_at_cursor = FALSE,
  warn = TRUE,
  ...
)
```

```r
headers_replace_all_text(
  x,
  old_value,
  new_value,
  only_at_cursor = FALSE,
  warn = TRUE,
  ...
)
```

```r
footers_replace_all_text(
  x,
  old_value,
  new_value,
  only_at_cursor = FALSE,
  warn = TRUE,
  ...
)
```

**Arguments**

- `x` a docx device
- `old_value` the value to replace
- `new_value` the value to replace it with
- `only_at_cursor` if TRUE, only search-and-replace at the current cursor; if FALSE (default), search-and-replace in the entire document (this can be slow on large documents!)
- `warn` warn if `old_value` could not be found.
- `...` optional arguments to grepl/gsub (e.g. `fixed=TRUE`)

**header_replace_all_text**

Replacements will be performed in each header of all sections.
Replace text content enclosed in a bookmark with different text. A bookmark will be considered as valid if enclosing words within a paragraph; i.e., a bookmark along two or more paragraphs is invalid, a bookmark set on a whole paragraph is also invalid, but bookmarking few words inside a paragraph is valid.
body_set_default_section

Define Default Section

Usage

body_replace_text_at_bkm(x, bookmark, value)
body_replace_img_at_bkm(x, bookmark, value)
headers_replace_text_at_bkm(x, bookmark, value)
headers_replace_img_at_bkm(x, bookmark, value)
footers_replace_text_at_bkm(x, bookmark, value)
footers_replace_img_at_bkm(x, bookmark, value)

Arguments

x a docx device
bookmark bookmark id
value the replacement string, of type character

Examples

doc <- read_docx()
doc <- body_add_par(doc, "a paragraph to replace", style = "centered")
doc <- body_bookmark(doc, "text_to_replace")
doc <- body_replace_text_at_bkm(doc, "text_to_replace", "new text")

# demo usage of bookmark and images ----
template <- system.file(package = "officer", "doc_examples/example.docx")

img.file <- file.path( R.home("doc"), "html", "logo.jpg" )

doc <- read_docx(path = template)
doc <- headers_replace_img_at_bkm(x = doc, bookmark = "bmk_header",
    value = external_img(src = img.file, width = .53, height = .7))
doc <- footers_replace_img_at_bkm(x = doc, bookmark = "bmk_footer",
    value = external_img(src = img.file, width = .53, height = .7))
print(doc, target = tempfile(fileext = ".docx"))

body_set_default_section

Define Default Section

Description

Define default section of the document. You can define section properties (page size, orientation, ...
with a prop_section object.
Usage

body_set_default_section(x, value)

Arguments

x
  an rdocx object
value
  a prop_section object

Illustrations

See Also

Other functions for Word sections: body_end_block_section(), body_end_section_columns_landscape(), body_end_section_columns(), body_end_section_continuous(), body_end_section_landscape(), body_end_section_portrait()

Examples

default_sect_properties <- prop_section(
  page_size = page_size(orient = "landscape"), type = "continuous",
  page_margins = page_mar(bottom = .75, top = 1.5, right = 2, left = 2)
)

doc_1 <- read_docx()
doc_1 <- body_add_table(doc_1, value = mtcars[1:10,], style = "table_template")
doc_1 <- body_add_par(doc_1, value = paste(rep(letters, 40), collapse = " "))
doc_1 <- body_set_default_section(doc_1, default_sect_properties)

print(doc_1, target = tempfile(fileext = ".docx"))

change_styles

Replace styles in a 'Word' Document

Description

Replace styles with others in a 'Word' document. This function can be used for paragraph, run/character and table styles.

Usage

change_styles(x, mapstyles)

Arguments

x
  an rdocx object
mapstyles
  a named list, names are the replacement style, content (as a character vector) are the styles to be replaced. Use styles_info() to display available styles.
Examples

# creating a sample docx so that we can illustrate how
# to change styles
doc_1 <- read_docx()

doc_1 <- body_add_par(doc_1, "A title", style = "heading 1")
doc_1 <- body_add_par(doc_1, "Another title", style = "heading 2")
doc_1 <- body_add_par(doc_1, "Hello world!", style = "Normal")
file <- print(doc_1, target = tempfile(fileext = ".docx"))

# now we can illustrate how
# to change styles with `change_styles`
doc_2 <- read_docx(path = file)
mapstyles <- list(
  "centered" = c("Normal", "heading 2"),
  "strong" = "Default Paragraph Font"
)
doc_2 <- change_styles(doc_2, mapstyles = mapstyles)
print(doc_2, target = tempfile(fileext = ".docx"))

color_scheme

Color scheme of a PowerPoint file

Description

Get the color scheme of a `PowerPoint` master layout into a data.frame.

Usage

color_scheme(x)

Arguments

x an rpptx object

See Also

Other functions for reading presentation informations: annotate_base(), doc_properties(), layout_properties(), layout_summary(), length.rpptx(), plot_layout_properties(), slide_size(), slide_summary()

Examples

x <- read_pptx()
color_scheme ( x = x )
cursor_begin

Set cursor in a 'Word' document

Description

A set of functions is available to manipulate the position of a virtual cursor. This cursor will be used when inserting, deleting or updating elements in the document.

Usage

```plaintext
cursor_begin(x)
cursor_bookmark(x, id)
cursor_end(x)
cursor_reach(x, keyword)
cursor_forward(x)
cursor_backward(x)
```

Arguments

- **x**: a docx device
- **id**: bookmark id
- **keyword**: keyword to look for as a regular expression

**cursor_begin**

Set the cursor at the beginning of the document, on the first element of the document (usually a paragraph or a table).

**cursor_bookmark**

Set the cursor at a bookmark that has previously been set.

**cursor_end**

Set the cursor at the end of the document, on the last element of the document.

**cursor_reach**

Set the cursor on the first element of the document that contains text specified in argument `keyword`. The argument `keyword` is a regexpr pattern.
cursor_forward

Move the cursor forward, it increments the cursor in the document.

cursor_backward

Move the cursor backward, it decrements the cursor in the document.

Examples

library(officer)

doc <- read_docx()
doc <- body_add_par(doc, "paragraph 1", style = "Normal")
doc <- body_add_par(doc, "paragraph 2", style = "Normal")
doc <- body_add_par(doc, "paragraph 3", style = "Normal")
doc <- body_add_par(doc, "paragraph 4", style = "Normal")
doc <- body_add_par(doc, "paragraph 5", style = "Normal")
doc <- body_add_par(doc, "paragraph 6", style = "Normal")
doc <- body_add_par(doc, "paragraph 7", style = "Normal")

# default template contains only an empty paragraph
# Using cursor_begin and body_remove, we can delete it
doc <- cursor_begin(doc)
doc <- body_remove(doc)

doc <- cursor_reach(doc, keyword = "paragraph 4")

# move the cursor forward
doc <- cursor_forward(doc)

# move the cursor at the end of the document
doc <- cursor_end(doc)

# cursor_bookmark ----

doc <- read_docx()
doc <- body_add_par(doc, "centered text", style = "centered")
doc <- body_bookmark(doc, "text_to_replace")
doc <- body_add_par(doc, "A title", style = "heading 1")
doc <- body_add_par(doc, "Hello world!", style = "Normal")
doc <- cursor_bookmark(doc, "text_to_replace")
doc <- body_add_table(doc, value = iris, style = "table_template")

print(doc, target = tempfile(fileext = ".docx"))
**docx_bookmarks**

**Description**

List bookmarks id that can be found in a 'Word' document.

**Usage**

```
docx_bookmarks(x)
```

**Arguments**

- `x` an `rdocx` object

**See Also**

Other functions for Word document informations: `doc_properties()`, `docx_dim()`, `length.rdocx()`, `set_doc_properties()`, `styles_info()

**Examples**

```r
library(officer)
doc_1 <- read_docx()
doc_1 <- body_add_par(doc_1, "centered text", style = "centered")
doc_1 <- body_bookmark(doc_1, "text_to_replace_1")
doc_1 <- body_add_par(doc_1, "centered text", style = "centered")
doc_1 <- body_bookmark(doc_1, "text_to_replace_2")
docx_bookmarks(doc_1)
docx_bookmarks(read_docx())
```

---

**docx_dim**

'Word' page layout

**Description**

Get page width, page height and margins (in inches). The return values are those corresponding to the section where the cursor is.

**Usage**

```
docx_dim(x)
```

**Arguments**

- `x` an `rdocx` object
See Also

Other functions for Word document informations: `doc_properties()`, `docx_bookmarks()`, `length.rdocx()`, `set_doc_properties()`, `styles_info()`

Examples

docx_dim(read_docx())

```r
# Show text chunk at cursor
docx_show_chunk(doc) # Output is 'Placeholder two'
```

Description

Show the structure of text tags at the current cursor. This is most useful when trying to troubleshoot search-and-replace functionality using `body_replace_all_text`.

Usage

docx_show_chunk(x)

Arguments

x a docx device

See Also

`body_replace_all_text`

Examples

doc <- read_docx()
doc <- body_add_par(doc, "Placeholder one")
doc <- body_add_par(doc, "Placeholder two")

# Show text chunk at cursor
docx_show_chunk(doc) # Output is 'Placeholder two'
**docx_summary**

get Word content in a data.frame

**Description**

read content of a Word document and return a data.frame representing the document.

**Usage**

docx_summary(x)

**Arguments**

x an rdocx object

**Note**

Documents included with body_add_docx() will not be accessible in the results.

**Examples**

```r
example_pptx <- system.file(package = "officer", "doc_examples/example.docx")
doc <- read_docx(example_pptx)
docx_summary(doc)
```

**doc_properties**

Read document properties

**Description**

Read Word or PowerPoint document properties and get results in a data.frame.

**Usage**

doc_properties(x)

**Arguments**

x an rdocx or rpptx object

**Value**

a data.frame
empty_content

See Also

Other functions for Word document informations: `docx_bookmarks()`, `docx_dim()`, `length.rdocx()`, `set_doc_properties()`, `styles_info()`

Other functions for reading presentation informations: `annotate_base()`, `color_scheme()`, `layout_properties()`, `layout_summary()`, `length.rpptx()`, `plot_layout_properties()`, `slide_size()`, `slide_summary()`

Examples

```r
x <- read_docx()
doc_properties(x)
```

```
empty_content  Empty block for 'PowerPoint'
```

Description

Create an empty object to include as an empty placeholder shape in a presentation. This comes in handy when presentation are updated through R, but a user still wants to add some comments in this new content.

Empty content also works with layout fields (slide number and date) to preserve them: they are included on the slide and keep being updated by PowerPoint, i.e. update to the when the slide number when the slide moves in the deck, update to the date.

Usage

```r
empty_content()
```

See Also

`ph_with()`, `body_add_blocks()`

Examples

```r
fileout <- tempfile(fileext = ".pptx")
doc <- read_pptx()
doc <- add_slide(doc, layout = "Two Content",
         master = "Office Theme")
doc <- ph_with(x = doc, value = empty_content(),
               location = ph_location_type(type = "title") )

doc <- add_slide(doc)
# add slide number as a computer field
doc <- ph_with(
     x = doc, value = empty_content(),
     location = ph_location_type(type = "sldNum"))

print(doc, target = fileout )
```
Description

Wraps an image in an object that can then be embedded in a PowerPoint slide or within a Word paragraph.

The image is added as a shape in PowerPoint (it is not possible to mix text and images in a PowerPoint form). With a Word document, the image will be added inside a paragraph.

Usage

external_img(src, width = 0.5, height = 0.2, alt = "")

Arguments

- **src**: image file path
- **width**: height in inches.
- **height**: height in inches
- **alt**: alternative text for images

Usage

You can use this function in conjunction with fpar to create paragraphs consisting of differently formatted text parts. You can also use this function as an *r chunk* in an R Markdown document made with package officedown.

See Also

- ph_with, body_add, fpar
- Other run functions for reporting: ftext(), hyperlink_ftext(), run_autonum(), run_bookmark(), run_columnbreak(), run_footnoteref(), run_footnote(), run_linebreak(), run_pagebreak(), run_reference(), run_word_field()

Examples

```r
# wrap r logo with external_img ----
srcfile <- file.path( R.home("doc"), "html", "logo.jpg" )
extimg <- external_img(src = srcfile, height = 1.06/2, width = 1.39/2)

# pptx example ----
doc <- read_pptx()
doc <- add_slide(doc)
doc <- ph_with(x = doc, value = extimg,
               location = ph_location_type(type = "body"),
               use_loc_size = FALSE )
```
print(doc, target = tempfile(fileext = ".pptx"))

fp_t <- fp_text(font.size = 20, color = "red")
an_fpar <- fpar(extimg, ftext(" is cool!", fp_t))

# docx example ----
x <- read_docx()
x <- body_add(x, an_fpar)
print(x, target = tempfile(fileext = ".docx"))

---

fpar

Formatted paragraph

Description

Create a paragraph representation by concatenating formatted text or images. The result can be inserted in a Word document or a PowerPoint presentation and can also be inserted in a block_list() call.

All its arguments will be concatenated to create a paragraph where chunks of text and images are associated with formatting properties.

fpar supports ftext(), external_img(), run_* functions (i.e. run_autonum(), run_seqfield()) when output is Word, and simple strings.

Default text and paragraph formatting properties can also be modified with function update().

Usage

fpar(..., fp_p = fp_par(), fp_t = fp_text_lite(), values = NULL)

## S3 method for class 'fpar'
update(object, fp_p = NULL, fp_t = NULL, ...)

Arguments

... cot objects (ftext(), external_img())
fp_p paragraph formatting properties, see fp_par()
fp_t default text formatting properties. This is used as text formatting properties when simple text is provided as argument, see fp_text().
values a list of cot objects. If provided, argument ... will be ignored.
object fpar object

See Also

block_list(), body_add_fpar(), ph_with()

Other block functions for reporting: block_caption(), block_list(), block_pour_docx(), block_section(), block_table(), block_toc(), plot_instr(), unordered_list()
Examples

```r
fpar(ftext("hello", shortcuts$fp_bold()))
```

# mix text and image -----
img.file <- file.path( R.home("doc"), "html", "logo.jpg" )

```r
bold_face <- shortcuts$fp_bold(font.size = 12)
bold_redface <- update(bold_face, color = "red")

fpar_1 <- fpar("Hello World, ",
  ftext("how ", prop = bold_redface ),
  external_img(src = img.file, height = 1.06/2, width = 1.39/2),
  ftext(" you?", prop = bold_face ) )
fpar_1
```

```r
img_in_par <- fpar(
  external_img(src = img.file, height = 1.06/2, width = 1.39/2),
  fp_p = fp_par(text.align = "center") )
```

---

**fp_border**

*Border properties object*

Description

create a border properties object.

Usage

```r
fp_border(color = "black", style = "solid", width = 1)
```

## S3 method for class 'fp_border'
update(object, color, style, width, ...)

Arguments

- **color**  
  border color - single character value (e.g. 
  "#000000" or "black")
- **style**  
  border style - single character value : "none" or "solid" or "dotted" or "dashed"
- **width**  
  border width - an integer value : 0>= value
- **object**  
  fp_border object
- **...**  
  further arguments - not used

See Also

Other functions for defining formatting properties: **fp_cell()**, **fp_par()**, **fp_text()**
Examples

```r
cell <- fp_cell()
cell <- fp_cell(color = "orange", style = "solid", width = 1)
cell <- fp_cell(color = "gray", style = "dotted", width = 1)

# modify object ------
border <- fp_border()
update(border, style = "dotted", width = 3)
```

---

**fp_cell**  
*Cell formatting properties*

Description

Create a `fp_cell` object that describes cell formatting properties.

Usage

```r
fp_cell(
  border = fp_border(width = 0),
  border.bottom,
  border.left,
  border.top,
  border.right,
  vertical.align = "center",
  margin = 0,
  margin.bottom,
  margin.top,
  margin.left,
  margin.right,
  background.color = "transparent",
  text.direction = "ltrb"
)
```

S3 methods for class `fp_cell`

- `format(x, type = "wml", ...)`
- `print(x, ...)`
- `update(object, border, border.bottom, border.left, border.top, ...)`
fp_cell

border.right,
vertical.align,
margin = 0,
margin.bottom,
margin.top,
margin.left,
margin.right,
background.color,
text.direction,
...)

Arguments

border shortcut for all borders.
border.bottom, border.left, border.top, border.right
fp_border for borders.
vertical.align cell content vertical alignment - a single character value, expected value is one of "center" or "top" or "bottom"
margin shortcut for all margins.
margin.bottom, margin.top, margin.left, margin.right
cell margins - 0 or positive integer value.
background.color cell background color - a single character value specifying a valid color (e.g. "#000000" or "black").
text.direction cell text rotation - a single character value, expected value is one of "lrtb", "tbrl", "btlr".
x, object fp_cell object
type output type - one of 'wml', 'pml', 'html'.
... further arguments - not used

See Also

Other functions for defining formatting properties: fp_border(), fp_par(), fp_text()

Examples

obj <- fp_cell(margin = 1)
update(obj, margin.bottom = 5)
Paragraph formatting properties

Description

Create a `fp_par` object that describes paragraph formatting properties.

Usage

```r
fp_par(
  text.align = "left",
  padding = 0,
  line_spacing = 1,
  border = fp_border(width = 0),
  padding.bottom,
  padding.top,
  padding.left,
  padding.right,
  border.bottom,
  border.left,
  border.top,
  border.right,
  shading.color = "transparent",
  keep_with_next = FALSE
)
```

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

## S3 method for class 'fp_par'
update(
  object,
  text.align,
  padding,
  border,
  padding.bottom,
  padding.top,
  padding.left,
  padding.right,
  border.bottom,
  border.left,
  border.top,
  border.right,
  shading.color,
  ...
)
**Arguments**

- `text.align`: text alignment - a single character value, expected value is one of 'left', 'right', 'center', 'justify'.
- `padding`: paragraph paddings - 0 or positive integer value. Argument padding overwrites arguments `padding.bottom, padding.top, padding.left, padding.right`.
- `line_spacing`: line spacing. 1 is single line spacing, 2 is double line spacing.
- `border`: shortcut for all borders.
- `padding.bottom, padding.top, padding.left, padding.right`: paragraph paddings - 0 or positive integer value.
- `border.bottom, border.left, border.top, border.right`: `fp_border` for borders. overwrite other border properties.
- `shading.color`: shading color - a single character value specifying a valid color (e.g. "#000000" or "black").
- `keep_with_next`: a scalar logical. Specifies that the paragraph (or at least part of it) should be rendered on the same page as the next paragraph when possible.
- `x, object`: `fp_par` object
- `...`: further arguments - not used

**Value**

A `fp_par` object

**See Also**

*`fp`*

Other functions for defining formatting properties: `fp_border()`, `fp_cell()`, `fp_text()`

**Examples**

```r
fp_par(text.align = "center", padding = 5)
obj <- fp_par(text.align = "center", padding = 1)
update(obj, padding.bottom = 5)
```

---

**Description**

Create a `fp_text` object that describes text formatting properties.

Function `fp_text_lite()` is generating properties with only entries for the parameters users provided. The undefined properties will inherit from the default settings.
Usage

fp_text(
  color = "black",
  font.size = 10,
  bold = FALSE,
  italic = FALSE,
  underlined = FALSE,
  font.family = "Arial",
  cs.family = NULL,
  eastasia.family = NULL,
  hansi.family = NULL,
  vertical.align = "baseline",
  shading.color = "transparent"
)

fp_text_lite(
  color = NA,
  font.size = NA,
  font.family = NA,
  cs.family = NA,
  eastasia.family = NA,
  hansi.family = NA,
  bold = NA,
  italic = NA,
  underlined = NA,
  vertical.align = "baseline",
  shading.color = NA
)

## S3 method for class 'fp_text'
format(x, type = "wml", ...)

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

## S3 method for class 'fp_text'
update(
  object,
  color,
  font.size,
  bold,
  italic,
  underlined,
  font.family,
  cs.family,
  eastasia.family,
  hansi.family,
  vertical.align,
\texttt{fp_text}

\begin{verbatim}
  shading.color,
  ...
)

Arguments

\begin{itemize}
  \item \texttt{color} \quad font color - a single character value specifying a valid color (e.g. "#000000" or "black")
  \item \texttt{font.size} \quad font size (in point) - 0 or positive integer value.
  \item \texttt{bold} \quad is bold
  \item \texttt{italic} \quad is italic
  \item \texttt{underlined} \quad is underlined
  \item \texttt{font.family} \quad single character value. Specifies the font to be used to format characters in the Unicode range (U+0000-U+007F).
  \item \texttt{cs.family} \quad optional font to be used to format characters in a complex script Unicode range. For example, Arabic text might be displayed using the "Arial Unicode MS" font.
  \item \texttt{eastasia.family} \quad optional font to be used to format characters in an East Asian Unicode range. For example, Japanese text might be displayed using the "MS Mincho" font.
  \item \texttt{hansi.family} \quad optional. Specifies the font to be used to format characters in a Unicode range which does not fall into one of the other categories.
  \item \texttt{vertical.align} \quad single character value specifying font vertical alignments. Expected value is one of the following: default ‘baseline’ or ‘subscript’ or ‘superscript’
  \item \texttt{shading.color} \quad shading color - a single character value specifying a valid color (e.g. "#000000" or "black").
  \item \texttt{x} \quad \texttt{fp_text} object
  \item \texttt{type} \quad output type - one of ‘wml’, ‘pml’, ‘html’.
  \item \texttt{...} \quad further arguments - not used
  \item \texttt{object} \quad \texttt{fp_text} object to modify
  \item \texttt{format} \quad format type, wml for MS word, pml for MS PowerPoint and html.
\end{itemize}

Value

\texttt{a \texttt{fp_text} object}

See Also

\texttt{ftext, fpar}

Other functions for defining formatting properties: \texttt{fp_border()}, \texttt{fp_cell()}, \texttt{fp_par()}

Examples

\begin{verbatim}
fp_text()
fp_text(color = "red")
fp_text(bold = TRUE, shading.color = "yellow")
print( fp_text (color="red", font.size = 12) )
\end{verbatim}
ftext  

*Formatted chunk of text*

**Description**

Format a chunk of text with text formatting properties (bold, color, ...). The function allows you to create pieces of text formatted the way you want.

**Usage**

```r
ftext(text, prop = NULL)
```

**Arguments**

- `text`: text value, a single character value
- `prop`: formatting text properties returned by `fp_text`. It also can be NULL in which case, no formatting is defined (the default is applied).

**usage**

You can use this function in conjunction with `fpar` to create paragraphs consisting of differently formatted text parts. You can also use this function as an *r chunk* in an R Markdown document made with package officedown.

**See Also**

- `fp_text`
- Other run functions for reporting: `external_img()`, `hyperlink_ftext()`, `run_autonum()`, `run_bookmark()`, `run_columnbreak()`, `run_footnoteref()`, `run_footnote()`, `run_linebreak()`, `run_pagebreak()`, `run_reference()`, `run_word_field()`

**Examples**

```r
ftext("hello", fp_text())

properties1 <- fp_text(color = "red")
properties2 <- fp_text(bold = TRUE, shading.color = "yellow")
ftext1 <- ftext("hello", properties1)
ftext2 <- ftext("World", properties2)
paragraph <- fpar(ftext1, " ", ftext2)

x <- read_docx()
x <- body_add(x, paragraph)
print(x, target = tempfile(fileext = ".docx"))
```
Description

Format a chunk of text with text formatting properties (bold, color, ...), the chunk is associated with an hyperlink.

Usage

hyperlink_ftext(text, prop = NULL, href)

Arguments

text text value, a single character value
prop formatting text properties returned by fp_text. It also can be NULL in which case, no formatting is defined (the default is applied).
href URL value

usage

You can use this function in conjunction with fpar to create paragraphs consisting of differently formatted text parts. You can also use this function as an r chunk in an R Markdown document made with package officedown.

See Also

Other run functions for reporting: external_img(), ftext(), run_autonum(), run_bookmark(), run_columnbreak(), run_footnoteref(), run_footnote(), run_linebreak(), run_pagebreak(), run_reference(), run_word_field()

Examples

ft <- fp_text(font.size = 12, bold = TRUE)
hyperlink_ftext(
  href = "https://cran.r-project.org/index.html",
  text = "some text", prop = ft)
Description

Get information about a particular slide layout into a data.frame.

Usage

layout_properties(x, layout = NULL, master = NULL)

Arguments

x an rpptx object
layout slide layout name to use
master master layout name where layout is located

See Also

Other functions for reading presentation informations: annotate_base(), color_scheme(), doc_properties(), layout_summary(), length.rpptx(), plot_layout_properties(), slide_size(), slide_summary()

Examples

x <- read_pptx()
layout_properties ( x = x, layout = "Title Slide", master = "Office Theme" )
layout_properties ( x = x, master = "Office Theme" )
layout_properties ( x = x, layout = "Two Content" )
layout_properties ( x = x )

---

Description

Get informations about slide layouts and master layouts into a data.frame. This function returns a data.frame containing all layout and master names.

Usage

layout_summary(x)

Arguments

x an rpptx object
length.rdocx

See Also

Other functions for reading presentation informations: `annotate_base()`, `color_scheme()`, `doc_properties()`, `layout_properties()`, `length.rpptx()`, `plot_layout_properties()`, `slide_size()`, `slide_summary()`

Examples

```r
my_pres <- read_pptx()
layout_summary(x = my_pres)
```

---

length.rdocx  Number of blocks inside an rdocx object

Description

return the number of blocks inside an rdocx object. This number also include the default section definition of a Word document - default Word section is an uninvisible element.

Usage

```r
## S3 method for class 'rdocx'
length(x)
```

Arguments

- `x` an rdocx object

See Also

Other functions for Word document informations: `doc_properties()`, `docx_bookmarks()`, `docx_dim()`, `set_doc_properties()`, `styles_info()`

Examples

```r
# how many elements are there in an new document produced
# with the default template.
length( read_docx() )
```
### length.rpptx

**Number of slides**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function <code>length</code> will return the number of slides.</td>
</tr>
</tbody>
</table>

#### Usage

```r
def length(x) |
```

#### Arguments

- `x`  
an `rpptx` object

#### See Also

Other functions for reading presentation informations: `annotate_base()`, `color_scheme()`, `doc_properties()`, `layout_properties()`, `layout_summary()`, `plot_layout_properties()`, `slide_size()`, `slide_summary()`

#### Examples

```r
my_pres <- read_pptx()  
my_pres <- add_slide(my_pres)  
my_pres <- add_slide(my_pres)  
length(my_pres)
```

---

### media_extract

**Extract media from a document object**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extract files from a <code>rpptx</code> object.</td>
</tr>
</tbody>
</table>

#### Usage

```r
media_extract(x, path, target)
```

#### Arguments

- `x`  
an `rpptx` object
- `path`  
media path, should be a relative path
- `target`  
target file
move_slide

Examples

```r
example_pptx <- system.file(package = "officer", "doc_examples/example.pptx")
doc <- read_pptx(example_pptx)
content <- pptx_summary(doc)
image_row <- content[content$content_type %in% "image", ]
media_file <- image_row$media_file
png_file <- tempfile(fileext = ".png")
media_extract(doc, path = media_file, target = png_file)
```

move_slide

Move a slide

Description

Move a slide in a pptx presentation.

Usage

```r
move_slide(x, index, to)
```

Arguments

- `x`: an rpptx object
- `index`: slide index, default to current slide position.
- `to`: new slide index.

Note

cursor is set on the last slide.

See Also

- `read_pptx()`

Other functions slide manipulation: `add_slide()`, `on_slide()`, `remove_slide()`, `set_notes()`

Examples

```r
x <- read_pptx()
x <- add_slide(x)
x <- ph_with(x, "Hello world 1", location = ph_location_type())
x <- add_slide(x)
x <- ph_with(x, "Hello world 2", location = ph_location_type())
x <- move_slide(x, index = 1, to = 2)
```
notes_location_label  Location of a named placeholder for notes

Description

The function will use the label of a placeholder to find the corresponding location in the slide notes.

Usage

notes_location_label(ph_label, ...)

Arguments

ph_label  placeholder label of the used notes master
...  unused arguments

notes_location_type  Location of a placeholder for notes

Description

The function will use the type name of the placeholder (e.g. body, hdr), to find the corresponding location.

Usage

notes_location_type(type = "body", ...)

Arguments

type  placeholder label of the used notes master
...  unused arguments
Description

The officer package facilitates access to and manipulation of 'Microsoft Word' and 'Microsoft PowerPoint' documents from R.

Examples of usage are:

- Create Word documents with tables, titles, TOC and graphics
- Importation of Word and PowerPoint files into data objects
- Write updated content back to a PowerPoint presentation
- Clinical reporting automation
- Production of reports from a shiny application

To start with officer, read about `read_docx()` and `read_pptx()`.

The package is also providing several objects that can be printed in 'R Markdown' documents for advanced Word or PowerPoint reporting as `run_autonum()` and `block_caption()`.

Author(s)

**Maintainer:** David Gohel <david.gohel@ardata.fr>

Other contributors:

- Frank Hangler <frank@plotandscatter.com> (function body_replace_all_text) [contributor]
- Liz Sander <l.sander@civisanalytics.com> (several documentation fixes) [contributor]
- Anton Victorson <anton@victorson.se> (fixes xml structures) [contributor]
- Jon Calder <jonmcalder@gmail.com> (update vignettes) [contributor]
- John Harrold <john.m.harrold@gmail.com> (function annotate_base) [contributor]
- John Muschelli <muschellij2@gmail.com> (google doc compatibility) [contributor]
- Bill Denney <wdenney@humanpredictions.com> (ORCID) (function as.matrix.rpptx) [contributor]
- Nikolai Beck <beck.nikolai@gmail.com> (set speaker notes for .pptx documents) [contributor]
- Stefan Moog <moogs@gmx.de> (added functionality to set shape geometry and outline) [contributor]
- Greg Leleu <gregoire.leleu@gmail.com> (fields functionality in ppt) [contributor]

See Also

officer-defunct  Defunct Functions in Package officer

Description
Defunct Functions in Package officer

Usage
slip_in_seqfield(...)
slip_in_column_break(...)
slip_in_xml(...)
slip_in_text(...)

Arguments
...   unused arguments

Details
slip_in_seqfield() is replaced by run_word_field().
slip_in_column_break() is replaced by run_columnbreak().
slip_in_xml() is replaced by fpar().
slip_in_text() is replaced by fpar().

on_slide  Change current slide

Description
Change current slide index of an rpptx object.

Usage
on_slide(x, index)

Arguments
x  an rpptx object
index  slide index
See Also

read_pptx(), ph_with()

Other functions slide manipulation: add_slide(), move_slide(), remove_slide(), set_notes()

Examples

doc <- read_pptx()
doc <- add_slide(doc, layout = "Title and Content", master = "Office Theme")
doc <- add_slide(doc, layout = "Title and Content", master = "Office Theme")
doc <- add_slide(doc, layout = "Title and Content", master = "Office Theme")
doc <- on_slide(doc, index = 1)
doc <- ph_with(x = doc, "First title",
   location = ph_location_type(type="title"))
doc <- on_slide(doc, index = 3)
doc <- ph_with(x = doc, "Third title",
   location = ph_location_type(type="title"))

file <- tempfile(fileext = ".pptx")
print(doc, target = file)

---

page_mar

Page margins object

Description

The margins for each page of a section. The function creates a representation of the dimensions of a page. The dimensions are defined by length, width and orientation. If the orientation is in landscape mode then the length becomes the width and the width becomes the length.

Usage

page_mar(
   bottom = 1,
   top = 1,
   right = 1,
   left = 1,
   header = 0.5,
   footer = 0.5,
   gutter = 0.5
)

Arguments

bottom, top    distance (in inches) between the bottom/top of the text margin and the bottom/top of the page. The text is placed at the greater of the value of this attribute and the extent of the header/footer text. A negative value indicates that the content should be measured from the bottom/topp of the page regardless of the footer/header, and so will overlap the footer/header. For example,
header=-0.5, bottom=1 means that the footer must start one inch from the bottom of the page and the main document text must start a half inch from the bottom of the page. In this case, the text and footer overlap since bottom is negative.

left, right distance (in inches) from the left/right edge of the page to the left/right edge of the text.

header distance (in inches) from the top edge of the page to the top edge of the header.

footer distance (in inches) from the bottom edge of the page to the bottom edge of the footer.

header distance (in inches) from the top edge of the page to the top edge of the header.

gutter page gutter (in inches).

See Also

Other functions for section definition: `page_size()`, `prop_section()`, `section_columns()`

Examples

```r
page_size()
```

---

**Description**

The function creates a representation of the dimensions of a page. The dimensions are defined by length, width and orientation. If the orientation is in landscape mode then the length becomes the width and the width becomes the length.

**Usage**

```r
page_size(width = 21/2.54, height = 29.7/2.54, orient = "portrait")
```

**Arguments**

- `width`, `height` page width, page height (in inches).
- `orient` page orientation, either 'landscape', either 'portrait'.

See Also

Other functions for section definition: `page_mar()`, `prop_section()`, `section_columns()`

Examples

```r
page_size(orient = "landscape")
```
Description

Add hyperlink to a placeholder in the current slide.

Usage

ph_hyperlink(x, type = "body", id = 1, id_chr = NULL, ph_label = NULL, href)

Arguments

- **x**: an rpptx object
- **type**: placeholder type
- **id**: placeholder index (integer) for a duplicated type. This is to be used when a placeholder type is not unique in the layout of the current slide, e.g. two placeholders with type 'body'. To add onto the first, use id = 1 and id = 2 for the second one. Values can be read from `slide_summary`.
- **id_chr**: deprecated.
- **ph_label**: label associated to the placeholder. Use column `ph_label` of result returned by `slide_summary`.
- **href**: hyperlink (do not forget http or https prefix)

See Also

- `ph_with`

Other functions for placeholders manipulation: `ph_remove()`, `ph_slidelink()`

Examples

```r
fileout <- tempfile(fileext = ".pptx")
loc_manual <- ph_location(bg = "red", newlabel = "mytitle")
doc <- read_pptx()
doc <- add_slide(doc)
doc <- ph_with(x = doc, "Un titre 1", location = loc_manual)
slide_summary(doc) # read column ph_label here
doc <- ph_hyperlink(x = doc, ph_label = "mytitle",
                        href = "https://cran.r-project.org")

print(doc, target = fileout )
```
**ph_location**  
*Location for a placeholder from scratch*

**Description**

The function will return a list that complies with expected format for argument `location` of function `ph_with`.

**Usage**

```r
ph_location(
  left = 1,
  top = 1,
  width = 4,
  height = 3,
  newlabel = "",
  bg = NULL,
  rotation = NULL,
  ln = NULL,
  geom = NULL,
  ...
)
```

**Arguments**

- `left, top, width, height`  
  placeholder coordinates in inches.
- `newlabel`  
  a label for the placeholder. See section details.
- `bg`  
  background color
- `rotation`  
  rotation angle
- `ln`  
  a `sp_line()` object specifying the outline style.
- `geom`  
  shape geometry, see [http://www.datypic.com/sc/ooxml/t-a_ST_ShapeType.html](http://www.datypic.com/sc/ooxml/t-a_ST_ShapeType.html)
- `...`  
  unused arguments

**Details**

The location of the bounding box associated to a placeholder within a slide is specified with the left top coordinate, the width and the height. These are defined in inches:

- **left**  
  left coordinate of the bounding box
- **top**  
  top coordinate of the bounding box
- **width**  
  width of the bounding box
- **height**  
  height of the bounding box
In addition to these attributes, a label can be associated with the shape. Shapes, text boxes, images and other objects will be identified with that label in the Selection Pane of PowerPoint. This label can then be reused by other functions such as `ph_location_label()`. It can be set with argument `newlabel`.

**See Also**

Other functions for placeholder location: `ph_location_fullsize()`, `ph_location_label()`, `ph_location_left()`, `ph_location_right()`, `ph_location_template()`, `ph_location_type()`

**Examples**

```r
doc <- read_pptx()
doc <- add_slide(doc)
doc <- ph_with(doc, "Hello world",
   location = ph_location(width = 4, height = 3, newlabel = "hello")
) print(doc, target = tempfile(fileext = ".pptx") )
```

```r
# Set geometry and outline
doc <- read_pptx()
doc <- add_slide(doc)
loc <- ph_location(left = 1, top = 1, width = 4, height = 3, bg = "steelblue",
   ln = sp_line(color = "red", lwd = 2.5),
   geom = "trapezoid")
doc <- ph_with(doc, "", loc = loc)
print(doc, target = tempfile(fileext = ".pptx") )
```

---

**ph_location_fullsize**  
*Location of a full size element*

**Description**

The function will return the location corresponding to a full size display.

**Usage**

`ph_location_fullsize(newlabel = "", ...)`

**Arguments**

- `newlabel` a label to associate with the placeholder.
- `...` unused arguments

**See Also**

Other functions for placeholder location: `ph_location_label()`, `ph_location_left()`, `ph_location_right()`, `ph_location_template()`, `ph_location_type()`, `ph_location()`
Examples

```r
doc <- read_pptx()
doc <- add_slide(doc)
doc <- ph_with(doc, "Hello world", location = ph_location_fullsize() )
print(doc, target = tempfile(fileext = ".pptx") )
```

---

**`ph_location_label`**  
*Location of a named placeholder*

### Description

The function will use the label of a placeholder to find the corresponding location.

### Usage

```r
ph_location_label(ph_label, newlabel = NULL, ...)
```

### Arguments

- **`ph_label`**  
  placeholder label of the used layout. It can be read in PowerPoint or with function `layout_properties()` in column `ph_label`.

- **`newlabel`**  
  a label to associate with the placeholder.

- **`...`**  
  unused arguments

### Details

The location of the bounding box associated to a placeholder within a slide is specified with the left top coordinate, the width and the height. These are defined in inches:

- **`left`**  
  left coordinate of the bounding box

- **`top`**  
  top coordinate of the bounding box

- **`width`**  
  width of the bounding box

- **`height`**  
  height of the bounding box

In addition to these attributes, a label can be associated with the shape. Shapes, text boxes, images and other objects will be identified with that label in the Selection Pane of PowerPoint. This label can then be reused by other functions such as `ph_location_label()`. It can be set with argument `newlabel`.

### See Also

Other functions for placeholder location:  
`ph_location_fullsize()`,  
`ph_location_left()` ,  
`ph_location_right()` ,  
`ph_location_template()` ,  
`ph_location_type()` ,  
`ph_location()`
Examples

# ph_location_label demo ----

doc <- read_pptx()
doc <- add_slide(doc, layout = "Title and Content")

# all ph_label can be read here
layout_properties(doc, layout = "Title and Content")

doc <- ph_with(doc, head(iris),
    location = ph_location_label(ph_label = "Content Placeholder 2") )
doc <- ph_with(doc, format(Sys.Date()),
    location = ph_location_label(ph_label = "Date Placeholder 3") )
doc <- ph_with(doc, "This is a title",
    location = ph_location_label(ph_label = "Title 1") )

print(doc, target = tempfile(fileext = ".pptx"))

---

ph_location_left  Location of a left body element

Description

The function will return the location corresponding to a left bounding box. The function assumes
the layout 'Two Content' is existing. This is an helper function, if you don’t have a layout named
'Two Content', use ph_location_type() and set arguments to your specific needs.

Usage

ph_location_left(newlabel = NULL, ...)

Arguments

newlabel  a label to associate with the placeholder.
...

unused arguments

See Also

Other functions for placeholder location: ph_location_fullsize(), ph_location_label(), ph_location_right(),
ph_location_template(), ph_location_type(), ph_location()

Examples

doc <- read_pptx()
doc <- add_slide(doc)
doc <- ph_with(doc, "Hello left", location = ph_location_left() )
doc <- ph_with(doc, "Hello right", location = ph_location_right() )
print(doc, target = tempfile(fileext = ".pptx"))
**ph_location_right**  
*Location of a right body element*

**Description**

The function will return the location corresponding to a right bounding box. The function assume the layout 'Two Content' is existing. This is an helper function, if you don’t have a layout named 'Two Content', use `ph_location_type()` and set arguments to your specific needs.

**Usage**

```r
ph_location_right(newlabel = NULL, ...)
```

**Arguments**

- `newlabel` a label to associate with the placeholder.
- `...` unused arguments

**See Also**

Other functions for placeholder location: `ph_location_fullsize()`, `ph_location_label()`, `ph_location_left()`, `ph_location_template()`, `ph_location_type()`, `ph_location()`

**Examples**

```r
doc <- read_pptx()
doc <- add_slide(doc)
doc <- ph_with(doc, "Hello left", location = ph_location_left() )
doc <- ph_with(doc, "Hello right", location = ph_location_right() )
print(doc, target = tempfile(fileext = ".pptx") )
```

---

**ph_location_template**  
*Location for a placeholder based on a template*

**Description**

The function will return a list that complies with expected format for argument location of function `ph_with`. A placeholder will be used as template and its positions will be updated with values `left`, `top`, `width`, `height`. 

```r
doc <- read_pptx()
doc <- add_slide(doc)
doc <- ph_with(doc, "Hello left", location = ph_location_left() )
doc <- ph_with(doc, "Hello right", location = ph_location_right() )
print(doc, target = tempfile(fileext = ".pptx") )
```
Usage

```r
ph_location_template(
  left = 1,
  top = 1,
  width = 4,
  height = 3,
  newlabel = "",
  type = NULL,
  id = 1,
  ...
)
```

Arguments

- `left, top, width, height`: placeholder coordinates in inches.
- `newlabel`: a label for the placeholder. See section details.
- `type`: placeholder type to look for in the slide layout, one of 'body', 'title', 'ctrTitle', 'subTitle', 'dt', 'fr', 'sldNum'. It will be used as a template placeholder.
- `id`: index of the placeholder template. If two body placeholders, there can be two different index: 1 and 2 for the first and second body placeholders defined in the layout.
- `...`: unused arguments

Details

The location of the bounding box associated to a placeholder within a slide is specified with the left top coordinate, the width and the height. These are defined in inches:

- `left`: left coordinate of the bounding box
- `top`: top coordinate of the bounding box
- `width`: width of the bounding box
- `height`: height of the bounding box

In addition to these attributes, a label can be associated with the shape. Shapes, text boxes, images and other objects will be identified with that label in the Selection Pane of PowerPoint. This label can then be reused by other functions such as `ph_location_label()`. It can be set with argument `newlabel`.

See Also

Other functions for placeholder location: `ph_location_fullsize()`, `ph_location_label()`, `ph_location_left()`, `ph_location_right()`, `ph_location_type()`, `ph_location()`
Examples

doc <- read_pptx()
doc <- add_slide(doc)
doc <- ph_with(doc, "Title",
    location = ph_location_type(type = "title") )
doc <- ph_with(doc, "Hello world",
    location = ph_location_template(top = 4, type = "title") )
print(doc, target = tempfile(fileext = ".pptx") )

---

**ph_location_type**

*Location of a placeholder based on a type*

**Description**

The function will use the type name of the placeholder (e.g. body, title), the layout name and few other criterias to find the corresponding location.

**Usage**

```r
ph_location_type(
    type = "body",
    position_right = TRUE,
    position_top = TRUE,
    newlabel = NULL,
    id = NULL,
    ...)
```

**Arguments**

- **type**: placeholder type to look for in the slide layout, one of 'body', 'title', 'ctrTitle', 'subTitle', 'dt', 'fr', 'sldNum'.
- **position_right**: the parameter is used when a selection with above parameters does not provide a unique position (for example layout 'Two Content' contains two element of type 'body'). If TRUE, the element the most on the right side will be selected, otherwise the element the most on the left side will be selected.
- **position_top**: same than position_right but applied to top versus bottom.
- **newlabel**: a label to associate with the placeholder.
- **id**: index of the placeholder. If two body placeholder, there can be two different index: 1 and 2 for the first and second body placeholders defined in the layout. If this argument is used, position_right and position_top will be ignored.
- **...**: unused arguments
ph_location_type

Details

The location of the bounding box associated to a placeholder within a slide is specified with the left top coordinate, the width and the height. These are defined in inches:

- **left** left coordinate of the bounding box
- **top** top coordinate of the bounding box
- **width** width of the bounding box
- **height** height of the bounding box

In addition to these attributes, a label can be associated with the shape. Shapes, text boxes, images and other objects will be identified with that label in the Selection Pane of PowerPoint. This label can then be reused by other functions such as ph_location_label(). It can be set with argument newlabel.

See Also

Other functions for placeholder location: ph_location_fullsize(), ph_location_label(), ph_location_left(), ph_location_right(), ph_location_template(), ph_location()

Examples

```r
# ph_location_type demo ----
loc_title <- ph_location_type(type = "title")
loc_footer <- ph_location_type(type = "ftr")
loc_dt <- ph_location_type(type = "dt")
loc_slidenum <- ph_location_type(type = "sldNum")
loc_body <- ph_location_type(type = "body")

doc <- read_pptx()
doc <- add_slide(doc)
doc <- ph_with(x = doc, "Un titre", location = loc_title)
doc <- ph_with(x = doc, "pied de page", location = loc_footer)
doc <- ph_with(x = doc, format(Sys.Date()), location = loc_dt)
doc <- ph_with(x = doc, "slide 1", location = loc_slidenum)
doc <- ph_with(x = doc, letters[1:10], location = loc_body)

loc_subtitle <- ph_location_type(type = "subTitle")
loc_ctrtitle <- ph_location_type(type = "ctrTitle")
doc <- add_slide(doc, layout = "Title Slide", master = "Office Theme")
doc <- ph_with(x = doc, "Un sous titre", location = loc_subtitle)
doc <- ph_with(x = doc, "Un titre", location = loc_ctrtitle)

fileout <- tempfile(fileext = ".pptx")
print(doc, target = fileout)
```
**ph_remove**

Remove a shape

**Description**

Remove a shape in a slide.

**Usage**

```r
ph_remove(x, type = "body", id = 1, ph_label = NULL, id_chr = NULL)
```

**Arguments**

- `x`: an rpptx object
- `type`: placeholder type
- `id`: placeholder index (integer) for a duplicated type. This is to be used when a placeholder type is not unique in the layout of the current slide, e.g. two placeholders with type 'body'. To add onto the first, use `id = 1` and `id = 2` for the second one. Values can be read from `slide_summary`.
- `ph_label`: label associated to the placeholder. Use column `ph_label` of result returned by `slide_summary`.
- `id_chr`: deprecated.

**See Also**

`ph_with`

Other functions for placeholders manipulation: `ph_hyperlink()`, `ph_slidelink()`

**Examples**

```r
dummy_fun <- function(doc){
  doc <- add_slide(doc, layout = "Two Content",
                   master = "Office Theme")
  doc <- ph_with(x = doc, value = "Un titre",
                 location = ph_location_type(type = "title"))
  doc <- ph_with(x = doc, value = "Un corps 1",
                 location = ph_location_type(type = "body", id = 1))
  doc <- ph_with(x = doc, value = "Un corps 2",
                 location = ph_location_type(type = "body", id = 2))
  doc
}
doc <- read_pptx()
for(i in 1:3)
  doc <- dummy_fun(doc)

doc <- on_slide(doc, index = 1)
```
ph_slidelink

```r
doc <- ph_remove(x = doc, type = "title")
doc <- on_slide(doc, index = 2)
doc <- ph_remove(x = doc, type = "body", id = 2)
doc <- on_slide(doc, index = 3)
doc <- ph_remove(x = doc, type = "body", id = 1)
print(doc, target = fileout)
```

---

**ph_slidelink**  
*Slide link to a placeholder*

**Description**

Add slide link to a placeholder in the current slide.

**Usage**

```r
ph_slidelink(
  x,
  type = "body",
  id = 1,
  id_chr = NULL,
  ph_label = NULL,
  slide_index
)
```

**Arguments**

- `x` an rpptx object  
- `type` placeholder type  
- `id` placeholder index (integer) for a duplicated type. This is to be used when a placeholder type is not unique in the layout of the current slide, e.g. two placeholders with type 'body'. To add onto the first, use `id = 1` and `id = 2` for the second one. Values can be read from `slide_summary`.  
- `id_chr` deprecated.  
- `ph_label` label associated to the placeholder. Use column `ph_label` of result returned by `slide_summary`.  
- `slide_index` slide index to reach

**See Also**

`ph_with`

Other functions for placeholders manipulation: `ph_hyperlink()`, `ph_remove()`
Examples

```r
fileout <- tempfile(fileext = ".pptx")
loc_title <- ph_location_type(type = "title")
doc <- read_pptx()
doc <- add_slide(doc)
doc <- ph_with(x = doc, "Un titre 1", location = loc_title)
doc <- on_slide(doc, 1)
slide_summary(doc)  # read column ph_label here
doc <- ph_slidelink(x = doc, ph_label = "Title 1", slide_index = 2)

print(doc, target = fileout)
```

---

**ph_with**

*Add objects on the current slide*

**Description**

add object into a new shape in the current slide. This function is able to add all supported outputs to a presentation. See section **Methods (by class)** to see supported outputs.

**Usage**

```r
ph_with(x, value, location, ...)
```

## S3 method for class 'character'

```r
ph_with(x, value, location, ...)
```

## S3 method for class 'numeric'

```r
ph_with(x, value, location, format_fun = format, ...)
```

## S3 method for class 'factor'

```r
ph_with(x, value, location, ...)
```

## S3 method for class 'logical'

```r
ph_with(x, value, location, format_fun = format, ...)
```

## S3 method for class 'block_list'

```r
ph_with(x, value, location, level_list = integer(0), ...)
```

## S3 method for class 'unordered_list'

```r
ph_with(x, value, location, ...)
```

## S3 method for class 'data.frame'

```r
ph_with(x,)
```
ph_with

value,
location,
header = TRUE,
tcf = table_conditional_formatting(),
alignment = NULL,
...)

## S3 method for class 'gg'
ph_with(x, value, location, res = 300, alt_text, scale = 1, ...)

## S3 method for class 'plot_instr'
ph_with(x, value, location, res = 300, ...)

## S3 method for class 'external_img'
ph_with(x, value, location, use_loc_size = TRUE, ...)

## S3 method for class 'fpar'
ph_with(x, value, location, ...)

## S3 method for class 'empty_content'
ph_with(x, value, location, ...)

## S3 method for class 'xml_document'
ph_with(x, value, location, ...)

Arguments

x an rpptx object

value object to add as a new shape. Supported objects are vectors, data.frame, graphics, block of formatted paragraphs, unordered list of formatted paragraphs, pretty tables with package flextable, editable graphics with package rvg, 'Microsoft' charts with package mschart.

location a placeholder location object. It will be used to specify the location of the new shape. This location can be defined with a call to one of the ph_location functions. See section "see also".

... further arguments passed to or from other methods. When adding a ggplot object or plot_instr, these arguments will be used by png function.

format_fun format function for non character vectors

level_list The list of levels for hierarchy structure as integer values. If used the object is formatted as an unordered list. If 1 and 2, item 1 level will be 1, item 2 level will be 2.

header display header if TRUE

tcf conditional formatting settings defined by table_conditional_formatting()

alignment alignment for each columns, 'l' for left, 'r' for right and 'c' for center. Default to NULL.
res        resolution of the png image in ppi
alt_text   Alt-text for screen-readers
scale      Multiplicative scaling factor, same as in ggsave
use_loc_size  if set to FALSE, external_img width and height will be used.

Methods (by class)

• character: add a character vector to a new shape on the current slide, values will be added as paragraphs.
• numeric: add a numeric vector to a new shape on the current slide, values will be be first formatted then added as paragraphs.
• factor: add a factor vector to a new shape on the current slide, values will be be converted as character and then added as paragraphs.
• block_list: add a block_list made of fpar to a new shape on the current slide.
• unordered_list: add a unordered_list made of fpar to a new shape on the current slide.
• data.frame: add a data.frame to a new shape on the current slide with function block_table(). Use package flextable instead for more advanced formattings.
• gg: add a ggplot object to a new shape on the current slide. Use package rvg for more advanced graphical features.
• plot_instr: add an R plot to a new shape on the current slide. Use package rvg for more advanced graphical features.
• external_img: add a external_img to a new shape on the current slide. When value is a external_img object, image will be copied into the PowerPoint presentation. The width and height specified in call to external_img will be ignored, their values will be those of the location, unless use_loc_size is set to FALSE.
• fpar: add an fpar to a new shape on the current slide as a single paragraph in a block_list.
• empty_content: add an empty_content to a new shape on the current slide.
• xml_document: add an xml_document object to a new shape on the current slide. This function is to be used to add custom openxml code.

Illustrations

See Also

ph_location_type, ph_location, ph_location_label, ph_location_left, ph_location_right, ph_location_fullsize, ph_location_template

Examples

# this name will be used to print the file
# change it to "youfile.pptx" to write the pptx file in your working directory.
fileout <- tempfile(fileext = ".pptx")
doc_1 <- read_pptx()
sz <- slide_size(doc_1)
# add text and a table ----
doc_1 <- add_slide(doc_1, layout = "Two Content", master = "Office Theme")
doc_1 <- ph_with(x = doc_1, value = c("Table cars"),
location = ph_location_type(type = "title") )
doc_1 <- ph_with(x = doc_1, value = names(cars),
location = ph_location_left() )
doc_1 <- ph_with(x = doc_1, value = cars,
location = ph_location_right() )

# add a base plot ----
anyplot <- plot_instr(code = {
  col <- c("#440154FF", "#443A83FF", "#31688EFF",
         "#21908CFF", "#35B779FF", "#8FD744FF", "#FDE725FF")
  barplot(1:7, col = col, yaxt="n")
})
doc_1 <- add_slide(doc_1)
doc_1 <- ph_with(doc_1, anyplot,
location = ph_location_fullsize(),
bg = "#006699")

# add a ggplot2 plot ----
if( require("ggplot2") ){
  doc_1 <- add_slide(doc_1)
  gg_plot <- ggplot(data = iris ) +
    geom_point(mapping = aes(Sepal.Length, Petal.Length),
               size = 3) +
    theme_minimal()
  doc_1 <- ph_with(x = doc_1, value = gg_plot,
                   location = ph_location_type(type = "body"),
                   bg = "transparent")
  doc_1 <- ph_with(x = doc_1, value = "graphic title",
                   location = ph_location_type(type="title") )
}

# add a external images ----
doc_1 <- add_slide(doc_1, layout = "Title and Content",
                   master = "Office Theme")
doc_1 <- ph_with(x = doc_1, value = empty_content(),
                 location = ph_location(left = 0, top = 0,
                                     width = sz$width, height = sz$height, bg = "black") )

svg_file <- file.path(R.home(component = "doc"), "html/Rlogo.svg")
if( require("rsvg") ){
  doc_1 <- ph_with(x = doc_1, value = "External images",
                   location = ph_location_type(type = "title") )
  doc_1 <- ph_with(x = doc_1, external_img(svg_file, 100/72, 76/72),
                   location = ph_location_right(), use_loc_size = FALSE )
  doc_1 <- ph_with(x = doc_1, external_img(svg_file),}
location = ph_location_left(),
use_loc_size = TRUE)

# add a block_list ----
dummy_text <- readLines(system.file(package = "officer",
"doc_examples/text.txt"))
fp_1 <- fp_text(bold = TRUE, color = "pink", font.size = 0)
fp_2 <- fp_text(bold = TRUE, font.size = 0)
fp_3 <- fp_text(italic = TRUE, color="red", font.size = 0)
bl <- block_list(
  fpar(ftext("hello world", fp_1)),
  fpar(
    ftext("hello", fp_2),
    ftext("hello", fp_3)
  ),
  dummy_text
)
doc_1 <- add_slide(doc_1)
doc_1 <- ph_with(x = doc_1, value = bl,
  location = ph_location_type(type="body") )

# fpar ------
fpt <- fp_text(bold = TRUE, font.family = "Bradley Hand",
font.size = 150, color = "#F5595B")
hw <- fpar(
  ftext("hello ", fpt),
  hyperlink_ftext(  
    href = "https://cran.r-project.org/index.html",  
    text = "cran", prop = fpt)
)
doc_1 <- add_slide(doc_1)
doc_1 <- ph_with(x = doc_1, value = hw,
  location = ph_location_type(type="body") )

# unordered_list ----
ul <- unordered_list(
  level_list = c(1, 2, 2, 3, 3, 1),
  str_list = c("Level1", "Level2", "Level2", "Level3", "Level3", "Level1"),
  style = fp_text(color = "red", font.size = 0) )
doc_1 <- add_slide(doc_1)
doc_1 <- ph_with(x = doc_1, value = ul,
  location = ph_location_type() )

print(doc_1, target = fileout )

---

plot_instr

Wrap plot instructions for png plotting in Powerpoint or Word
plot_layout_properties

Description

A simple wrapper to capture plot instructions that will be executed and copied in a document. It produces an object of class 'plot_instr' with a corresponding method `ph_with()` and `body_add_plot()`. The function enable usage of any R plot with argument code. Wrap your code between curly bracket if more than a single expression.

Usage

`plot_instr(code)`

Arguments

code plotting instructions

See Also

`ph_with()`, `body_add_plot()`

Other block functions for reporting: `block_caption()`, `block_list()`, `block_pour_docx()`, `block_section()`, `block_table()`, `block_toc()`, `fpar()`, `unordered_list()`

Examples

```r
# plot_instr demo ----

anyplot <- plot_instr(code = {
  barplot(1:5, col = 2:6)
})

doc <- read_docx()
doc <- body_add(doc, anyplot, width = 5, height = 4)
print(doc, target = tempfile(fileext = ".docx"))

doc <- read_pptx()
doc <- add_slide(doc)
doc <- ph_with(
  doc, anyplot,
  location = ph_location_fullsize(),
  bg = "#00000066", fontsize = 12
)
print(doc, target = tempfile(fileext = ".pptx"))
```

plot_layout_properties

Description

Plot slide layout properties and print informations into defined placeholders. This can be useful to help visualise placeholders locations and identifier.
Usage

plot_layout_properties(x, layout = NULL, master = NULL, labels = TRUE)

Arguments

x  
an rpptx object
layout  
slide layout name to use
master  
master layout name where layout is located
labels  
if TRUE, placeholder labels will be printed, if FALSE placeholder types and identifiers will be printed.

See Also

Other functions for reading presentation informations: annotate_base(), color_scheme(), doc_properties(), layout_properties(), layout_summary(), length.rpptx(), slide_size(), slide_summary()

Examples

x <- read_pptx()
plot_layout_properties(x = x, layout = "Title Slide", master = "Office Theme")
plot_layout_properties(x = x, layout = "Two Content")

ppto_summary

PowerPoint content in a data.frame

Description

Read content of a PowerPoint document and return a dataset representing the document.

Usage

ppto_summary(x)

Arguments

x  
an rpptx object

Examples

eample_ppto <- system.file(package = "officer", "doc_examples/example.ppto")
doc <- read_pptx(example_ppto)
ppto_summary(doc)
ppto_summary(example_ppto)
### Print_rpptx

**Description**

Write a 'PowerPoint' file with an object of class 'rpptx' (created with `read_pptx()`).

**Usage**

```r
## S3 method for class 'rpptx'
print(x, target = NULL, ...)
```

**Arguments**

- `x`: an rpptx object
- `target`: path to the pptx file to write
- `...`: unused

**See Also**

- `read_pptx`

**Examples**

```r
# write a rdocx object in a docx file ----
file <- tempfile(fileext = ".pptx")
doc <- read_pptx()
print(doc, target = file)
```

### Prop_section

**Description**

A section is a grouping of blocks (ie. paragraphs and tables) that have a set of properties that define pages on which the text will appear.

A Section properties object stores information about page composition, such as page size, page orientation, borders and margins.

**Usage**

```r
prop_section(
  page_size = NULL,
  pageMargins = NULL,
  type = NULL,
  section_columns = NULL
)
```
Arguments

page_size  page dimensions, an object generated with function page_size.
page_margins page margins, an object generated with function page_mar.
type  Section type. It defines how the contents of the section will be placed relative to the previous section. Available types are "continuous" (begins the section on the next paragraph), "evenPage" (begins on the next even-numbered page), "nextColumn" (begins on the next column on the page), "nextPage" (begins on the following page), "oddPage" (begins on the next odd-numbered page).
section_columns section columns, an object generated with function section_columns.

Illustrations

Note

There is no support yet for header and footer contents definition.

See Also

block_section

Other functions for section definition: page_mar(), page_size(), section_columns()

Examples

library(officer)

landscape_one_column <- block_section(
  prop_section(
    page_size = page_size(orient = "landscape"), type = "continuous"
  )
)
landscape_two_columns <- block_section(
  prop_section(
    page_size = page_size(orient = "landscape"), type = "continuous",
    section_columns = section_columns(widths = c(4.75, 4.75))
  )
)

doc_1 <- read_docx()
# there starts section with landscape_one_column
doc_1 <- body_add_table(doc_1, value = mtcars[1:10,], style = "table_template")
doc_1 <- body_end_block_section(doc_1, value = landscape_one_column)
# there stops section with landscape_one_column

# there starts section with landscape_two_columns
doc_1 <- body_add_par(doc_1, value = paste(rep(letters, 50), collapse = " "))
doc_1 <- body_end_block_section(doc_1, value = landscape_two_columns)
# there stops section with landscape_two_columns

doc_1 <- body_add_table(doc_1, value = mtcars[1:25, ], style = "table_template")

print(doc_1, target = tempfile(fileext = ".docx"))

---

prop_table

## Table properties

### Description

Define table properties such as fixed or autofit layout, table width in the document, eventually column widths.

### Usage

```r
prop_table(
  style = NA_character_,
  layout = table_layout(),
  width = table_width(),
  stylenames = table_stylenames(),
  colwidths = table_colwidths(),
  tcf = table_conditional_formatting(),
  align = "center"
)
```

### Arguments

- **style**: table style to be used to format table
- **layout**: layout defined by `table_layout()`.  
- **width**: table width in the document defined by `table_width()`  
- **stylenames**: columns styles defined by `table_stylenames()`  
- **colwidths**: column widths defined by `table_colwidths()`  
- **tcf**: conditional formatting settings defined by `table_conditional_formatting()`  
- **align**: table alignment (one of left, center or right)

### See Also

Other functions for table definition: `table_colwidths()`, `table_conditional_formatting()`, `table_layout()`, `table_stylenames()`, `table_width()`

### Examples

```r
prop_table()
to_wml(prop_table())
```
read_docx  

Create a 'Word' document object

Description

read and import a docx file as an R object representing the document. When no file is specified, it uses a default empty file.

Use then this object to add content to it and create Word files from R.

Usage

read_docx(path = NULL)

## S3 method for class 'rdocx'
print(x, target = NULL, ...)

Arguments

path     path to the docx file to use as base document.
x        an rdocx object
target   path to the docx file to write
...      unused

Value

an object of class rdocx.

Methods (by generic)

- print: write docx to a file. It returns the path of the result file.

styles

read_docx() uses a Word file as the initial document. This is the original Word document from which the document layout, paragraph styles, or table styles come.

You will be able to add formatted text, change the paragraph style with the R api but also use the styles from the original document.

See body_add_* functions to add content.

Illustrations

See Also

body_add_par, body_add_plot, body_add_table
Examples

```r
library(officer)

pinst <- plot_instr(
  z <- c(rnorm(100), rnorm(50, mean = 5))
  plot(density(z))
)

doc_1 <- read_docx()
doc_1 <- body_add_par(doc_1, "This is a table", style = "heading 2")
doc_1 <- body_add_table(doc_1, value = mtcars, style = "table_template")
doc_1 <- body_add_par(doc_1, "This is a plot", style = "heading 2")
doc_1 <- body_add_plot(doc_1, pinst)
docx_file_1 <- print(doc_1, target = tempfile(fileext = ".docx"))

template <- system.file(package = "officer", "doc_examples", "landscape.docx")
doc_2 <- read_docx(path = template)
doc_2 <- body_add_par(doc_2, "This is a table", style = "heading 2")
doc_2 <- body_add_table(doc_2, value = mtcars)
doc_2 <- body_add_par(doc_2, "This is a plot", style = "heading 2")
doc_2 <- body_add_plot(doc_2, pinst)
docx_file_2 <- print(doc_2, target = tempfile(fileext = ".docx"))
```

---

**read_pptx**

Create a 'PowerPoint' document object

Description

Read and import a pptx file as an R object representing the document.

The function is called `read_pptx` because it allows you to initialize an object of class `rpptx` from an existing PowerPoint file. Content will be added to the existing presentation. By default, an empty document is used.

Usage

```r
read_pptx(path = NULL)
```

Arguments

- `path` path to the pptx file to use as base document.

Master layouts and slide layouts

`read_pptx()` uses a PowerPoint file as the initial document. This is the original PowerPoint document where all slide layouts, placeholders for shapes and styles come from. Major points to be aware of are:
• Slide layouts are relative to a master layout. A document can contain one or more master
layouts; a master layout can contain one or more slide layouts.
• A slide layout inherits design properties from its master layout but some properties can be
overwritten.
• Designs and formatting properties of layouts and shapes (placeholders in a layout) are defined
within the initial document. There is no R function to modify these values - they must be
defined in the initial document.

See Also

print.rpptx(), add_slide(), plot_layout_properties(), ph_with()

Examples

read_pptx()
**remove_slide**

Remove a slide from a pptx presentation.

**Usage**

```r
remove_slide(x, index = NULL)
```

**Arguments**

- `x`: an rpptx object
- `index`: slide index, default to current slide position.

**Note**

cursor is set on the last slide.

**See Also**

`read_pptx()`, `ph_with()`, `ph_remove()`

Other functions slide manipulation: `add_slide()`, `move_slide()`, `on_slide()`, `set_notes()`

**Examples**

```r
my_pres <- read_pptx()
my_pres <- add_slide(my_pres)
my_pres <- remove_slide(my_pres)
```

---

**run_autonum**

Auto number

**Description**

Create an autonumbered chunk, i.e. a string representation of a sequence, each item will be numbered. These runs can also be bookmarked and be used later for cross references.
Usage

run_autonum(
  seq_id = "table",
  pre_label = "Table ",
  post_label = ": ",
  bkm = NULL,
  bkm_all = FALSE,
  prop = NULL,
  start_at = NULL,
  tnd = 0,
  tns = "-"
)

Arguments

seq_id sequence identifier
pre_label, post_label text to add before and after number
bkm bookmark id to associate with autonumber run. If NULL, no bookmark is added.
  Value can only be made of alpha numeric characters, ':' and '-'.
bkm_all if TRUE, the bookmark will be set on the whole string, if FALSE, the bookmark
  will be set on the number only. Default to FALSE. As an effect when a reference
  to this bookmark is used, the text can be like "Table 1" or "1" (pre_label is not
  included in the referenced text).
prop formatting text properties returned by fp_text.
start_at If not NULL, it must be a positive integer, it specifies the new number to use, at
  which number the auto numbering will restart.
tnd title number depth, a positive integer (only applies if positive) that specify the
  depth (or heading of level depth) to use for prefixing the caption number with
  this last reference number. For example, setting tnd=2 will generate numbered
  captions like '4.3-2' (figure 2 of chapter 4.3).
tns separator to use between title number and table number. Default is "-".

usage

You can use this function in conjunction with fpar to create paragraphs consisting of differently
formatted text parts. You can also use this function as an r chunk in an R Markdown document
made with package officedown.

See Also

Other run functions for reporting: external_img(), ftext(), hyperlink_ftext(), run_bookmark(),
run_columnbreak(), run_footnoteref(), run_footnote(), run_linebreak(), run_pagebreak(),
run_reference(), run_word_field()

Other Word computed fields: run_reference(), run_word_field()
**run_bookmark**

**Examples**

```r
run_autonum()
run_autonum(seq_id = "fig", pre_label = "fig.")
run_autonum(seq_id = "tab", pre_label = "Table ", bkm = "anytable")
run_autonum(seq_id = "tab", pre_label = "Table ", bkm = "anytable",
            tnd = 2, tns= " ")
```

---

**run_bookmark**  
*Bookmark for 'Word'*

**Description**

Add a bookmark on a run object.

**Usage**

```r
run_bookmark(bkm, run)
```

**Arguments**

- **bkm**
  - bookmark id to associate with run. Value can only be made of alpha numeric characters, ` '-' and '_'`.
- **run**
  - a run object, made with a call to one of the "run functions for reporting".

**usage**

You can use this function in conjunction with `fpar` to create paragraphs consisting of differently formatted text parts. You can also use this function as an *r chunk* in an R Markdown document made with package officedown.

**See Also**

Other run functions for reporting: `external_img()`, `ftext()`, `hyperlink_ftext()`, `run_autonum()`, `run_columnbreak()`, `run_footnoteref()`, `run_footnote()`, `run_linebreak()`, `run_pagebreak()`, `run_reference()`, `run_word_field()`

**Examples**

```r
ft <- fp_text(font.size = 12, bold = TRUE)
run_bookmark("par1", ftext("some text", ft))
```
**run_columnbreak**  
*Column break for 'Word'*

Description
Create a representation of a column break.

Usage
run_columnbreak()

usage
You can use this function in conjunction with fpar to create paragraphs consisting of differently formatted text parts. You can also use this function as an *r chunk* in an R Markdown document made with package officedown.

See Also
Other run functions for reporting: external_img(), ftext(), hyperlink_ftext(), run_autonum(), run_bookmark(), run_footnoteref(), run_footnote(), run_linebreak(), run_pagebreak(), run_reference(), run_word_field()

Examples
run_columnbreak()

---

**run_footnote**  
*Footnote for 'Word'*

Description
Wraps a footnote in an object that can then be inserted as a run/chunk with fpar() or within an R Markdown document.

Usage
run_footnote(x, prop = NULL)

Arguments
- **x** a set of blocks to be used as footnote content returned by function block_list().
- **prop** formatting text properties returned by fp_text_lite() or fp_text(). It also can be NULL in which case, no formatting is defined (the default is applied).
See Also

Other run functions for reporting: `external_img()`, `ftext()`, `hyperlink_ftext()`, `run_autonum()`, `run_bookmark()`, `run_columnbreak()`, `run_footnoteref()`, `run_linebreak()`, `run_pagebreak()`, `run_reference()`, `run_word_field()`

Examples

```r
library(officer)

cmp = fp_text_lite(bold = TRUE)
fp_refnote <- fp_text_lite(vertical.align = "superscript")

img.file <- file.path( R.home("doc"), "html", "logo.jpg" )
bl <- block_list(
  fpar(ftext("hello", fp_bold)),
  fpar(
    ftext("hello world", fp_bold),
    external_img(src = img.file, height = 1.06, width = 1.39)
  )
)

a_par <- fpar(
  "this paragraph contains a note ",
  run_footnote(x = bl, prop = fp_refnote),
  ".
)

doc <- read_docx()
doc <- body_add_fpar(doc, value = a_par, style = "Normal")
print(doc, target = tempfile(fileext = ".docx"))
```

---

**run_footnoteref**  
*Word footnote reference*

**Description**

Wraps a footnote reference in an object that can then be inserted as a run/chunk with `fpar()` or within an R Markdown document.

**Usage**

```r
run_footnoteref(prop = NULL)
```

**Arguments**

- `prop`: formatting text properties returned by `fp_text_lite()` or `fp_text()`. It also can be NULL in which case, no formatting is defined (the default is applied).
run_linebreak

See Also

Other run functions for reporting: `external_img()`, `ftext()`, `hyperlink_ftext()`, `run_autonum()`, `run_bookmark()`, `run_columnbreak()`, `run_footnote()`, `run_linebreak()`, `run_pagebreak()`, `run_reference()`, `run_word_field()`

Examples

```r
run_footnoteref()
to_wml(run_footnoteref())
```

---

run_linebreak  Page break for 'Word'

Description

Object representing a line break for a Word document. The result must be used within a call to `fpar`.

Usage

```r
run_linebreak()
```

usage

You can use this function in conjunction with `fpar` to create paragraphs consisting of differently formatted text parts. You can also use this function as an `r chunk` in an R Markdown document made with package officedown.

See Also

Other run functions for reporting: `external_img()`, `ftext()`, `hyperlink_ftext()`, `run_autonum()`, `run_bookmark()`, `run_columnbreak()`, `run_footnoteref()`, `run_footnote()`, `run_pagebreak()`, `run_reference()`, `run_word_field()`

Examples

```r
fp_t <- fp_text(font.size = 12, bold = TRUE)
an_fpar <- fpar("let's add a line break", run_linebreak(), ftext("and blah blah!", fp_t))

x <- read_docx()
x <- body_add(x, an_fpar)
print(x, target = tempfile(fileext = ".docx"))
```
run_pagebreak

Page break for 'Word'

Description
Object representing a page break for a Word document.

Usage
run_pagebreak()

usage
You can use this function in conjunction with fpar to create paragraphs consisting of differently formatted text parts. You can also use this function as an r chunk in an R Markdown document made with package officedown.

See Also
Other run functions for reporting: external_img(), ftext(), hyperlink_ftext(), run_autonum(), run_bookmark(), run_columnbreak(), run_footnoteref(), run_footnote(), run_linebreak(), run_reference(), run_word_field()

Examples
fp_t <- fp_text(font.size = 12, bold = TRUE)
an_fpar <- fpar("let's add a break page", run_pagebreak(), ftext("and blah blah!", fp_t))

x <- read_docx()
x <- body_add(x, an_fpar)
print(x, target = tempfile(fileext = ".docx"))

run_reference

Cross reference

Description
Create a representation of a reference

Usage
run_reference(id, prop = NULL)

Arguments
id reference id, a string
prop formatting text properties returned by fp_text.
usage

You can use this function in conjunction with \texttt{fpar} to create paragraphs consisting of differently formatted text parts. You can also use this function as an \texttt{r chunk} in an R Markdown document made with package officedown.

See Also

Other run functions for reporting: \texttt{external_img()}, \texttt{ftext()}, \texttt{hyperlink_ftext()}, \texttt{run_autonum()}, \texttt{run_bookmark()}, \texttt{run_columnbreak()}, \texttt{run_footnoteref()}, \texttt{run_footnote()}, \texttt{run_linebreak()}, \texttt{run_pagebreak()}, \texttt{run_word_field()}

Other Word computed fields: \texttt{run_autonum()}, \texttt{run_word_field()}

Examples

\begin{verbatim}
run_reference('a_ref')
\end{verbatim}

\begin{verbatim}
run_word_field 'Word' computed field
\end{verbatim}

Description

Create a 'Word' computed field.

Usage

\begin{verbatim}
run_word_field(field, prop = \texttt{NULL}, seqfield = field)
run_seqfield(field, prop = \texttt{NULL}, seqfield = field)
\end{verbatim}

Arguments

- \texttt{field, seqfield}
  - computed field string (seqfield will be totally superseded by field in the future).
- \texttt{prop}
  - formatting text properties returned by \texttt{fp_text}.

usage

You can use this function in conjunction with \texttt{fpar} to create paragraphs consisting of differently formatted text parts. You can also use this function as an \texttt{r chunk} in an R Markdown document made with package officedown.

Note

In the previous version, this function was called \texttt{run_seqfield} but the name was wrong and should have been \texttt{run_word_field}. 


section_columns

See Also

Other run functions for reporting: `external_img()`, `ftext()`, `hyperlink_ftext()`, `run_autonum()`, `run_bookmark()`, `run_columnbreak()`, `run_footnoteref()`, `run_footnote()`, `run_linebreak()`, `run_pagebreak()`, `run_reference()`

Other Word computed fields: `run_autonum()`, `run_reference()`

Examples

```r
run_word_field(field = "PAGE  \\* MERGEFORMAT")
run_word_field(field = "Date  @$ "MMMM d yyyy\"")
```

---

### Description

The function creates a representation of the columns of a section.

### Usage

```r
section_columns(widths = c(2.5, 2.5), space = 0.25, sep = FALSE)
```

### Arguments

- **widths**: columns widths in inches. If 3 values, 3 columns will be produced.
- **space**: space in inches between columns.
- **sep**: if TRUE a line is separating columns.

### See Also

Other functions for section definition: `page_mar()`, `page_size()`, `prop_section()`

### Examples

```r
section_columns()
```
set_autonum_bookmark  
*Update bookmark of an autonumber run*

**Description**

This function lets recycling a object made by `run_autonum()` by changing the bookmark value. This is useful to avoid calling `run_autonum()` several times because of many tables.

**Usage**

```r
set_autonum_bookmark(x, bkm = NULL)
```

**Arguments**

- `x`: an object of class `run_autonum()`
- `bkm`: bookmark id to associate with autonumber run. Value can only be made of alpha numeric characters, ':', '-' and '_'.

**See Also**

- `run_autonum()`

**Examples**

```r
z <- run_autonum(seq_id = "tab", pre_label = "Table ",
                 bkm = "anytable")
set_autonum_bookmark(z, bkm = "anothertable")
```

---

**set_doc_properties  
*Set document properties***

**Description**

set Word or PowerPoint document properties. These are not visible in the document but are available as metadata of the document.

Any character property can be added as a document property. It provides an easy way to insert arbitrary fields. Given the challenges that can be encountered with find-and-replace in word with officer, the use of document fields and quick text fields provides a much more robust approach to automatic document generation from R.
**set_doc_properties**

**Usage**

```r
set_doc_properties(
  x, 
  title = NULL, 
  subject = NULL, 
  creator = NULL, 
  description = NULL, 
  created = NULL, 
  ..., 
  values = NULL
)
```

**Arguments**

- **x**: an rdocx or rpptx object
- **title**, **subject**, **creator**, **description**: text fields
- **created**: a date object
- **...**: named arguments (names are field names), each element is a single character value specifying value associated with the corresponding field name.
- **values**: a named list (names are field names), each element is a single character value specifying value associated with the corresponding field name. If `values` is provided, argument `...` will be ignored.

**Note**

The "last modified" and "last modified by" fields will be automatically be updated when the file is written.

**See Also**

Other functions for Word document informations: `doc_properties()`, `docx_bookmarks()`, `docx_dim()`, `length.rdocx()`, `styles_info()`

**Examples**

```r
x <- read_docx()
x <- set_doc_properties(x, title = "title", 
subject = "document subject", creator = "Me me me", 
description = "this document is empty", 
created = Sys.time(), 
yoyo = "yok yok", 
glop = "pas glop")
x <- doc_properties(x)
```
set_notes

Set notes for current slide

Description
Set speaker notes for the current slide in a pptx presentation.

Usage
set_notes(x, value, location, ...)

## S3 method for class 'character'
set_notes(x, value, location, ...)

## S3 method for class 'block_list'
set_notes(x, value, location, ...)

Arguments
- `x`: an rpptx object
- `value`: text to be added to notes
- `location`: a placeholder location object. It will be used to specify the location of the new shape. This location can be defined with a call to one of the notes_ph functions. See section "see also".
- `...`: further arguments passed to or from other methods.

Methods (by class)
- `character`: add a character vector to a placeholder in the notes on the current slide, values will be added as paragraphs.
- `block_list`: add a `block_list()` to a placeholder in the notes on the current slide.

See Also
- `print.rpptx()`, `read_pptx()`, `add_slide()`, `notes_location_label()`, `notes_location_type()`

Other functions slide manipulation: `add_slide()`, `move_slide()`, `on_slide()`, `remove_slide()`

Examples
# this name will be used to print the file
# change it to "youfile.pptx" to write the pptx
# file in your working directory.
fileout <- tempfile(fileext = "pptx")
ftp_blue_bold <- fp_text_lite(color = "#006699", bold = TRUE)
doc <- read_pptx()
# add a slide with some text ----
doc <- add_slide(doc, layout = "Title and Content", master = "Office Theme")
```r
doc <- ph_with(x = doc, value = "Slide Title 1",
location = ph_location_type(type = "title") )
# set speaker notes for the slide ----
doc <- set_notes(doc, value = "This text will only be visible for the speaker.",
location = notes_location_type("body"))

# add a slide with some text ----
doc <- add_slide(doc, layout = "Title and Content", master = "Office Theme")
doc <- ph_with(x = doc, value = "Slide Title 2",
location = ph_location_type(type = "title") )
bl <- block_list(
  fpar(ftext("hello world", fpt_blue_bold)),
  fpar(ftext("Turlututu chapeau pointu", fpt_blue_bold))
)
doc <- set_notes(doc, value = bl,
location = notes_location_type("body"))

print(doc, target = fileout)
```

---

**sheet_select**

*Select sheet*

**Description**

Set a particular sheet selected when workbook will be edited.

**Usage**

```r
sheet_select(x, sheet)
```

**Arguments**

- `x`: rxlsx object
- `sheet`: sheet name

**Examples**

```r
my_ws <- read_xlsx()
my_pres <- add_sheet(my_ws, label = "new sheet")
my_pres <- sheet_select(my_ws, sheet = "new sheet")
print(my_ws, target = tempfile(fileext = ".xlsx") )
```
**shortcuts**  
*shortcuts for formatting properties*

### Description

Shortcuts for fp_text, fp_par, fp_cell and fp_border.

### Usage

shortcuts

### Examples

shortcuts$fp_bold()  
shortcuts$fp_italic()  
shortcuts$b_null()

---

**slide_size**  
*Slides width and height*

### Description

Get the width and height of slides in inches as a named vector.

### Usage

slide_size(x)

### Arguments

x  
an rpptx object

### See Also

Other functions for reading presentation informations: annotate_base(), color_scheme(), doc_properties(), layout_properties(), layout_summary(), length.rpptx(), plot_layout_properties(), slide_summary()

### Examples

my_pres <- read_pptx()  
my_pres <- add_slide(my_pres,  
  layout = "Two Content", master = "Office Theme")  
slide_size(my_pres)
slide_summary

Slide content in a data.frame

Description

Get content and positions of current slide into a data.frame. Data for any tables, images, or paragraphs are imported into the resulting data.frame.

Usage

slide_summary(x, index = NULL)

Arguments

  x
  an rpptx object

  index
  slide index

Note

The column id of the result is not to be used by users. This is a technical string id whose value will be used by office when the document will be rendered. This is not related to argument index required by functions ph_with.

See Also

Other functions for reading presentation informations: annotate_base(), color_scheme(), doc_properties(), layout_properties(), layout_summary(), length.rpptx(), plot_layout_properties(), slide_size()

Examples

my_pres <- read_pptx()
my_pres <- add_slide(my_pres)
my_pres <- ph_with(my_pres, format(Sys.Date()),
  location = ph_location_type(type="dt"))
my_pres <- add_slide(my_pres)
my_pres <- ph_with(my_pres, iris[1:2,],
  location = ph_location_type(type="body"))
slide_summary(my_pres)
slide_summary(my_pres, index = 1)
slip_in_footnote  
append a footnote

Description
append a new footnote into a paragraph of an rdocx object

Usage
slip_in_footnote(x, style = NULL, blocks, pos = "after")

Arguments
x an rdocx object
style text style to be used for the reference note
blocks set of blocks to be used as footnote content returned by function block_list.
pos where to add the new element relative to the cursor, "after" or "before".

Note
This function will be deprecated in the next release because it is not efficient and make users write complex code. Use instead fpar() to build formatted paragraphs.

sp_line  
Line properties

Description
Create a sp_line object that describes line properties.

Usage
sp_line(
color = "transparent",
lwd = 1,
lty = "solid",
linecmpd = "sng",
lineend = "rnd",
linejoin = "round",
headend = sp_lineend(type = "none"),
tailend = sp_lineend(type = "none")
)

## S3 method for class 'sp_line'
print(x, ...)
## S3 method for class 'sp_line'
update(
    object,
    color,
    lwd,
    lty,
    linecmpd,
    lineend,
    linejoin,
    headend,
    tailend,
    ...
)

### Arguments

- **color**: line color - a single character value specifying a valid color (e.g. "#000000" or "black").
- **lwd**: line width (in point) - 0 or positive integer value.
- **lty**: single character value specifying the line type. Expected value is one of the following: default 'solid' or 'dot' or 'dash' or 'lgDash' or 'dashDot' or 'lgDashDot' or 'lgDashDotDot' or 'sysDash' or 'sysDot' or 'sysDashDot' or 'sysDashDotDot'.
- **linecmpd**: single character value specifying the compound line type. Expected value is one of the following: default 'sng' or 'dbl' or 'tri' or 'thinThick' or 'thickThin'.
- **lineend**: single character value specifying the line end style. Expected value is one of the following: default 'rnd' or 'sq' or 'flat'.
- **linejoin**: single character value specifying the line join style. Expected value is one of the following: default 'round' or 'bevel' or 'miter'.
- **headend**: a sp_lineend object specifying line head end style.
- **tailend**: a sp_lineend object specifying line tail end style.
- **x, object**: sp_line object.
- **...**: further arguments - not used.

### Value

A sp_line object.

### See Also

- **sp_lineend**
- Other functions for defining shape properties: sp_lineend()
Examples

sp_line()
sp_line(color = "red", lwd = 2)
sp_line(lty = "dot", linecmpd = "dbl")
print( sp_line (color="red", lwd = 2) )
obj <- sp_line (color="red", lwd = 2)
update( obj, linecmpd = "dbl" )

sp_lineend  Line end properties

Description

Create a sp_lineend object that describes line end properties.

Usage

sp_lineend(type = "none", width = "med", length = "med")

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

## S3 method for class 'sp_lineend'
update(object, type, width, length, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>single character value specifying the line end type. Expected value is one of the following: default 'none' or 'triangle' or 'stealth' or 'diamond' or 'oval' or 'arrow'</td>
</tr>
<tr>
<td>width</td>
<td>single character value specifying the line end width. Expected value is one of the following: default 'sm' or 'med' or 'lg'</td>
</tr>
<tr>
<td>length</td>
<td>single character value specifying the line end length. Expected value is one of the following: default 'sm' or 'med' or 'lg'</td>
</tr>
<tr>
<td>x, object</td>
<td>sp_lineend object</td>
</tr>
<tr>
<td>...</td>
<td>further arguments - not used</td>
</tr>
</tbody>
</table>

Value

a sp_lineend object

See Also

sp_line

Other functions for defining shape properties: sp_line()
Examples

```r
sp_lineend()
sp_lineend(type = "triangle")
sp_lineend(type = "arrow", width = "lg", length = "lg")
print( sp_lineend (type="triangle", width = "lg") )
obj <- sp_lineend (type="triangle", width = "lg")
update( obj, type = "arrow" )
```

---

**styles_info**

*Read 'Word' styles*

**Description**

read Word styles and get results in a data.frame.

**Usage**

```r
styles_info(
  x,
  type = c("paragraph", "character", "table", "numbering"),
  is_default = c(TRUE, FALSE)
)
```

**Arguments**

- `x`: an rdocx object
- `type`, `is_default`:
  subsets for types (i.e. paragraph) and default style (when `is_default` is TRUE or FALSE)

**See Also**

Other functions for Word document informations: `doc_properties()`, `docx_bookmarks()`, `docx_dim()`, `length.rdocx()`, `set_doc_properties()`

**Examples**

```r
x <- read_docx()
styless_info()
styless_info(x)
styles_info(x, type = "paragraph", is_default = TRUE)
```
table_colwidths | \textit{Column widths of a table}

\textbf{Description}

The function defines the size of each column of a table.

\textbf{Usage}

\begin{verbatim}
table_colwidths(widths = NULL)
\end{verbatim}

\textbf{Arguments}

- \texttt{widths} | Column widths expressed in inches.

\textbf{See Also}

Other functions for table definition: \texttt{prop_table()}, \texttt{table_conditional_formatting()}, \texttt{table_layout()}, \texttt{table_stylenames()}, \texttt{table_width()}

\textbf{table_conditional_formatting}

\textit{Table conditional formatting}

\textbf{Description}

Tables can be conditionally formatted based on few properties as whether the content is in the first row, last row, first column, or last column, or whether the rows or columns are to be banded.

\textbf{Usage}

\begin{verbatim}
table_conditional_formatting(
  first_row = TRUE,
  first_column = FALSE,
  last_row = FALSE,
  last_column = FALSE,
  no_hband = FALSE,
  no_vband = TRUE
)
\end{verbatim}
Arguments

- first_row, last_row
  - apply or remove formatting from the first or last row in the table.
- first_column, last_column
  - apply or remove formatting from the first or last column in the table.
- no_hband, no_vband
  - don’t display odd and even rows or columns with alternating shading for ease of reading.

Note

You must define a format for first_row, first_column and other properties if you need to use them. The format is defined in a docx template.

See Also

Other functions for table definition: prop_table(), table_colwidths(), table_layout(), table_stylenames(), table_width()

Examples

```
table_conditional_formatting(first_row = TRUE, first_column = TRUE)
```

---

### Description

When a table is displayed in a document, it can either be displayed using a fixed width or autofit layout algorithm:

- fixed: uses fixed widths for columns. The width of the table is not changed regardless of the contents of the cells.
- autofit: uses the contents of each cell and the table width to determine the final column widths.

### Usage

```
table_layout(type = "autofit")
```

### Arguments

- type
  - 'autofit' or 'fixed' algorithm. Default to 'autofit'.

### See Also

Other functions for table definition: prop_table(), table_colwidths(), table_conditional_formatting(), table_stylenames(), table_width()
table_stylenames  Paragraph styles for columns

Description

The function defines the paragraph styles for columns.

Usage

```r
table_stylenames(stylenames = list())
```

Arguments

- **stylenames**: a named character vector, names are column names, values are paragraph styles associated with each column. If a column is not specified, default value `Normal` is used. Another form is as a named list, the list names are the styles and the contents are column names to be formatted with the corresponding style.

See Also

Other functions for table definition: `prop_table()`, `table_colwidths()`, `table_conditional_formatting()`, `table_layout()`, `table_width()`

Examples

```r
library(officer)

stylenames <- c(
  vs = "centered", am = "centered",
  gear = "centered", carb = "centered"
)

doc_1 <- read_docx()
doc_1 <- body_add_table(doc_1,
  value = mtcars, style = "table_template",
  stylenames = table_stylenames(stylenames = stylenames))

print(doc_1, target = tempfile(fileext = ".docx"))

stylenames <- list(
  "centered" = c("vs", "am", "gear", "carb")
)

doc_2 <- read_docx()
doc_2 <- body_add_table(doc_2,
  value = mtcars, style = "table_template",
  stylenames = table_stylenames(stylenames = stylenames))
```
table_width

)  

print(doc_2, target = tempfile(fileext = "docx"))

---

**table_width**  
*Preferred width for a table*

**Description**

Define the preferred width for a table.

**Usage**

```r
table_width(width = 1, unit = "pct")
```

**Arguments**

- `width`: value of the preferred width of the table.
- `unit`: unit of the width. Possible values are `"in"` (inches) and `"pct"` (percent)

**Word**

All widths in a table are considered preferred because widths of columns can conflict and the table layout rules can require a preference to be overridden.

**See Also**

Other functions for table definition: `prop_table()`, `table_colwidths()`, `table_conditional_formatting()`, `table_layout()`, `table_stylenames()`

---

**unordered_list**  
*Unordered list*

**Description**

unordered list of text for PowerPoint presentations. Each text is associated with a hierarchy level.

**Usage**

```r
unordered_list(str_list = character(0), level_list = integer(0), style = NULL)
```

**Arguments**

- `str_list`: list of strings to be included in the object
- `level_list`: list of levels for hierarchy structure. Use 0 for 'no bullet', 1 for level 1, 2 for level 2 and so on.
- `style`: text style, a `fp_text` object list or a single `fp_text` objects. Use `fp_text(font.size = 0, ...) to inherit from default sizes of the presentation.`
See Also

- `ph_with`

Other block functions for reporting: `block_caption()`, `block_list()`, `block_pour_docx()`, `block_section()`, `block_table()`, `block_toc()`, `fpar()`, `plot_instr()`

Examples

```r
unordered_list(
    level_list = c(1, 2, 2, 3, 3, 1),
    str_list = c("Level1", "Level2", "Level2", "Level3", "Level3", "Level1"),
    style = fp_text(color = "red", font.size = 0)
)
unordered_list(
    level_list = c(1, 2, 1),
    str_list = c("Level1", "Level2", "Level1"),
    style = list(
        fp_text(color = "red", font.size = 0),
        fp_text(color = "pink", font.size = 0),
        fp_text(color = "orange", font.size = 0)
    ))
```
Index

* Word computed fields
  run_autonum, 85
  run_reference, 91
  run_word_field, 92

* block functions for reporting
  block_caption, 7
  block_list, 8
  block_pour_docx, 9
  block_section, 10
  block_table, 11
  block_toc, 12
  fpar, 42
  plot_instr, 76
  unordered_list, 107

* functions for Word document informations
  doc_properties, 39
  docx_bookmarks, 36
  docx_dim, 37
  length_rdocx, 53
  set_doc_properties, 94
  styles_info, 103

* functions for Word sections
  body_end_block_section, 23
  body_end_section_columns, 24
  body_end_section_columns_landscape, 25
  body_end_section_continuous, 26
  body_end_section_landscape, 27
  body_end_section_portrait, 28
  body_set_default_section, 32

* functions for adding content
  body_add_blocks, 12
  body_add_break, 13
  body_add_caption, 14
  body_add_docx, 15
  body_add_fpar, 16
  body_add_gg, 17
  body_add_img, 18
  body_add_par, 19
  body_add_plot, 20
  body_add_table, 21
  body_add_toc, 22

* functions for defining formatting properties
  fp_border, 43
  fp_cell, 44
  fp_par, 46
  fp_text, 47

* functions for defining shape properties
  sp_line, 100
  sp_lineend, 102

* functions for placeholder location
  ph_location, 62
  ph_location_fullsize, 63
  ph_location_label, 64
  ph_location_left, 65
  ph_location_right, 66
  ph_location_template, 66
  ph_location_type, 68

* functions for placeholders manipulation
  ph_hyperlink, 61
  ph_remove, 70
  ph_slidelink, 71

* functions for reading presentation informations
  annotate_base, 5
  color_scheme, 34
  doc_properties, 39
  layout_properties, 52
  layout_summary, 52
  length_rpptx, 54
  plot_layout_properties, 77
  slide_size, 98
  slide_summary, 99

* functions for section definition
  page_mar, 59
  page_size, 60
prop_section, 79
section_columns, 93

* functions for table definition
prop_table, 81
table_colwidths, 104
table_conditional_formatting, 104
table_layout, 105
table_stylenames, 106
table_width, 107

* functions slide manipulation
add_slide, 5
move_slide, 55
on_slide, 58
remove_slide, 85
set_notes, 96

* run functions for reporting
external_img, 41
ftext, 50
hyperlink_ftext, 51
run_autonum, 85
run_bookmark, 87
run_columnbreak, 88
run_footnote, 88
run_footnoteref, 89
run_linebreak, 90
run_pagebreak, 91
run_reference, 91
run_word_field, 92

add_sheet, 4
add_slide, 5, 55, 59, 85, 96
add_slide(), 84, 96
annotate_base, 5, 34, 40, 52–54, 78, 98, 99
as.matrix.rptx, 6

block_caption, 7, 8–12, 42, 77, 108
block_caption(), 8, 14, 57
block_list, 7, 8, 9–12, 42, 74, 77, 100, 108
block_list(), 13, 42, 88, 96
block_pour_docx, 7, 8, 9, 10–12, 42, 77, 108
block_pour_docx(), 8
block_section, 7–9, 10, 11, 12, 23, 42, 77, 80, 108
block_section(), 8
block_table, 7–10, 11, 12, 42, 77, 108
block_table(), 8, 74
block_toc, 7–11, 12, 42, 77, 108
block_toc(), 8
body_add, 41

body_add_blocks, 12, 14–20, 22
body_add_blocks(), 8, 40
body_add_break, 13, 13, 14–20, 22
body_add_caption, 13, 14, 14, 15–20, 22
body_add_docx, 13, 14, 15, 16–20, 22
body_add_fpar, 13–15, 16, 17–20, 22
body_add_fpar(), 42
body_add_gg, 13–16, 17, 18–20, 22
body_add_img, 13–17, 18, 19, 20, 22
body_add_par, 13–18, 19, 20, 22, 82
body_add_plot, 13–19, 20, 22, 82
body_add_plot(), 77
body_add_table, 13–20, 21, 22, 82
body_add_toc, 13–20, 22, 22
body_bookmark, 23
body_end_block_section, 23, 25–28, 33
body_end_block_section(), 24
body_end_section_columns, 24, 24, 26–28, 33
body_end_section_columns_landscape, 24, 25, 26–28, 33
body_end_section_continuous, 24–26, 26, 27, 28, 33
body_end_section_landscape, 24–26, 27, 28, 33
body_end_section_portrait, 24–27, 28, 33
body_remove, 28
body_replace_all_text, 29, 38
body_replace_img_at_bkm
(body_replace_text_at_bkm), 31
body_replace_text_at_bkm, 31
body_set_default_section, 24–28, 32

change_styles, 33
color_scheme, 6, 34, 40, 52–54, 78, 98, 99
cursor_backward (cursor_begin), 35
cursor_begin, 35
cursor_bookmark (cursor_begin), 35
cursor_end (cursor_begin), 35
cursor_forward (cursor_begin), 35
cursor_reach (cursor_begin), 35
docx_bookmarks, 36, 38, 40, 53, 95, 103
docx_dim, 37, 37, 40, 53, 95, 103
docx_show_chunk, 30, 31, 38
docx_summary, 39
read_pptx(), 5, 55, 57, 59, 79, 85, 96
read_xlsx, 84
regex, 29, 31
remove_slide, 5, 55, 59, 85, 96
run_autonum, 7, 41, 50, 51, 85, 87–93
run_autonum(), 12, 42, 57, 94
run_bookmark, 41, 50, 51, 86, 87, 88–93
run_columnbreak, 41, 50, 51, 86, 87, 88, 89–93
run_footnote, 41, 50, 51, 86–88, 88, 90–93
run_footnoteref, 41, 50, 51, 86–89, 89, 90–93
run_linebreak, 41, 50, 51, 86–90, 90, 91–93
run_pagebreak, 41, 50, 51, 86–90, 91, 92, 93
run_reference, 41, 50, 51, 86–91, 91, 93
run_seqfield(run_word_field), 92
run_seqfield(), 42
run_word_field, 41, 50, 51, 86–92, 92
section_columns, 60, 80, 93
set_autonum_bookmark, 94
set_doc_properties, 37, 38, 40, 53, 94, 103
set_notes, 5, 55, 59, 85, 96
sheet_select, 97
shortcuts, 98
slide_size, 6, 34, 40, 52–54, 78, 98, 99
slide_summary, 6, 34, 40, 52–54, 61, 70, 71, 78, 98, 99
slip_in_column_break (officer-defunct), 58
slip_in_footnote, 100
slip_in_seqfield (officer-defunct), 58
slip_in_text (officer-defunct), 58
slip_in_xml (officer-defunct), 58
sp_line, 100, 102
sp_line(), 62
sp_lineend, 101, 102
styles_info, 37, 38, 40, 53, 95, 103
styles_info(), 33
table_colwidths, 81, 104–106, 107
table_colwidths(), 81
table_conditional_formatting, 81, 104, 104, 105–107
table_conditional_formatting(), 73, 81
table_layout, 81, 104, 105, 106, 107
table_layout(), 81
table_stylenames, 81, 104, 105, 106, 107
table_stylenames(), 21, 81