Package ‘officer’

February 24, 2024

Type Package

Title Manipulation of Microsoft Word and PowerPoint Documents

Version 0.6.5

Description
Access and manipulate 'Microsoft Word', 'RTF' 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 MIT + file LICENSE

Imports grDevices, stats, graphics, utils, zip (>= 2.1.0), xml2 (>= 1.1.0), openssl, R6, uuid, ragg

URL https://ardata-fr.github.io/officeverse/
https://davidgohel.github.io/officer/

Encoding UTF-8

BugReports https://github.com/davidgohel/officer/discussions

RoxygenNote 7.3.1

Suggests testthat, devEMF, ggplot2, rmarkdown, doconv (>= 0.3.0), knitr, rsvg, magick

NeedsCompilation no

Author David Gohel [aut, cre],
Stefan Moog [aut],
ArData [cph],
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),
Greg Leleu [ctb] (fields functionality in ppt),
Hongyuan Jia [ctb] (<https://orcid.org/0000-0002-0075-8183>)

Maintainer David Gohel <david.gohel@ardata.fr>
Repository CRAN
Date/Publication 2024-02-24 16:10:06 UTC

R topics documented:

add_sheet .................................................. 4
add_slide .................................................. 5
annotate_base .............................................. 6
as.matrix.rpptx ........................................... 7
block_caption .............................................. 8
block_list .................................................. 9
block_pour_docx .......................................... 10
block_section ............................................. 11
block_table ................................................. 11
block_toc .................................................. 12
body_add_blocks ......................................... 13
body_add_break ............................................ 14
body_add_caption ......................................... 14
body_add_docx ............................................. 15
body_add_fpar ............................................. 16
body_add_gg ............................................... 18
body_add_img .............................................. 19
body_add_par .............................................. 20
body_add_plot ............................................. 21
body_add_table ........................................... 22
body_add_toc .............................................. 23
body_bookmark ............................................ 24
body_end_block_section .................................. 25
body_end_section_columns ................................ 26
body_end_section_columns_landscape ..................... 27
body_end_section_continuous ................................ 28
body_end_section_landscape ................................ 29
body_end_section_portrait ................................ 29
body_remove ............................................... 30
body_replace_all_text .................................... 31
body_replace_text_at_bkm ................................ 33
body_set_default_section .................................. 34
change_styles ............................................. 35
color_scheme ............................................... 36
cursor_begin .............................................. 36
docx_bookmarks ............................................ 39
docx_comments .............................................. 40
### Topics Documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>docx_dim</td>
<td>41</td>
</tr>
<tr>
<td>docx_set_character_style</td>
<td>41</td>
</tr>
<tr>
<td>docx_set_paragraph_style</td>
<td>42</td>
</tr>
<tr>
<td>docx_show_chunk</td>
<td>44</td>
</tr>
<tr>
<td>docx_summary</td>
<td>44</td>
</tr>
<tr>
<td>doc_properties</td>
<td>45</td>
</tr>
<tr>
<td>empty_content</td>
<td>46</td>
</tr>
<tr>
<td>external_img</td>
<td>46</td>
</tr>
<tr>
<td>fpar</td>
<td>48</td>
</tr>
<tr>
<td>fp_border</td>
<td>49</td>
</tr>
<tr>
<td>fp_cell</td>
<td>51</td>
</tr>
<tr>
<td>fp_par</td>
<td>53</td>
</tr>
<tr>
<td>fp_tab</td>
<td>55</td>
</tr>
<tr>
<td>fp_tabs</td>
<td>55</td>
</tr>
<tr>
<td>fp_text</td>
<td>56</td>
</tr>
<tr>
<td>ftext</td>
<td>58</td>
</tr>
<tr>
<td>hyperlink_ftext</td>
<td>59</td>
</tr>
<tr>
<td>layout_properties</td>
<td>60</td>
</tr>
<tr>
<td>layout_summary</td>
<td>61</td>
</tr>
<tr>
<td>length.docx</td>
<td>61</td>
</tr>
<tr>
<td>length.rpptx</td>
<td>62</td>
</tr>
<tr>
<td>media_extract</td>
<td>62</td>
</tr>
<tr>
<td>move_slide</td>
<td>63</td>
</tr>
<tr>
<td>notes_location_label</td>
<td>64</td>
</tr>
<tr>
<td>notes_location_type</td>
<td>64</td>
</tr>
<tr>
<td>officer</td>
<td>65</td>
</tr>
<tr>
<td>officer-defunct</td>
<td>66</td>
</tr>
<tr>
<td>on_slide</td>
<td>67</td>
</tr>
<tr>
<td>page_mar</td>
<td>68</td>
</tr>
<tr>
<td>page_size</td>
<td>69</td>
</tr>
<tr>
<td>ph_hyperlink</td>
<td>69</td>
</tr>
<tr>
<td>ph_location</td>
<td>70</td>
</tr>
<tr>
<td>ph_location_fullsize</td>
<td>72</td>
</tr>
<tr>
<td>ph_location_label</td>
<td>72</td>
</tr>
<tr>
<td>ph_location_left</td>
<td>73</td>
</tr>
<tr>
<td>ph_location_right</td>
<td>74</td>
</tr>
<tr>
<td>ph_location_template</td>
<td>75</td>
</tr>
<tr>
<td>ph_location_type</td>
<td>76</td>
</tr>
<tr>
<td>ph_remove</td>
<td>78</td>
</tr>
<tr>
<td>ph_slidelink</td>
<td>79</td>
</tr>
<tr>
<td>ph_with</td>
<td>80</td>
</tr>
<tr>
<td>plot_instr</td>
<td>86</td>
</tr>
<tr>
<td>plot_layout_properties</td>
<td>87</td>
</tr>
<tr>
<td>pptx_summary</td>
<td>87</td>
</tr>
<tr>
<td>print.rpptx</td>
<td>88</td>
</tr>
<tr>
<td>print.rtf</td>
<td>89</td>
</tr>
<tr>
<td>prop_section</td>
<td>89</td>
</tr>
<tr>
<td>prop_table</td>
<td>92</td>
</tr>
</tbody>
</table>
add_sheet

Description

Add a sheet into an xlsx worksheet.

Usage

add_sheet(x, label)
add_slide

Arguments

x
rxlsx object

label
sheet label

Examples

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

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"
)
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')
as.matrix.rpptx  PowerPoint table to matrix

Description

Convert the data in an 'PowerPoint' table to a matrix or all data to a list of matrices.

Usage

```r
## S3 method for class 'rpptx'
as.matrix(
  x,
  ..., 
  slide_id = NA_integer_,
  id = NA_character_,
  span = c(NA_character_, "fill")
)
```

Arguments

- `x` The rpptx object to convert (as created by `officer::read_pptx()`)
- `...` Ignored
- `slide_id` The slide number to load from (NA indicates first slide with a table, NULL indicates all slides and all tables)
- `id` The table ID to load from (ignored if `slide_id` is not provided, 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

Caption block

Description

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

Usage

block_caption(label, style = NULL, 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

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 )

doc_1 <- read_docx()
doc_1 <- body_add(doc_1, "A title", style = "heading 1")
doc_1 <- body_add(doc_1, "Hello world!", style = "Normal")
doc_1 <- body_add(doc_1, caption)
doc_1 <- body_add(doc_1, mtcars, style = "table_template")

print(doc_1, target = tempfile(fileext = ".docx"))
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" )
fp_blue_bold <- fp_text(color = "#006699", bold = TRUE)
fp_red_italic <- fp_text(color = "#C32900", italic = TRUE)

value <- block_list(
  fpar(ftext("hello world", fp_blue_bold)),
  fpar(ftext("hello", fp_blue_bold), " ",
       ftext("world", fp_red_italic)),
  fpar(
      ftext("hello world", fp_red_italic),
      external_img(
          src = img.file, height = 1.06, width = 1.39)))

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

value <- block_list(
  fpar(ftext("hello world", fpt_blue_bold)),
  fpar(ftext("hello", fpt_blue_bold), " ",
       ftext("world", fpt_red_italic)),
  fpar(
    ftext("blah blah blah", fpt_red_italic)))
value

doc <- read_pptx()
doc <- add_slide(doc)
doc <- ph_with(doc, value, location = ph_location_type(type = "body"))
print(doc, target = tempfile(fileext = ".pptx"))
```

---

**block_pour_docx**

**External Word document placeholder**

**Description**

Pour the content of a docx file in the resulting docx from an `R Markdown` document.

**Usage**

`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))
pert.print(doc_1, target = target)
```
block_section  

Section for 'Word'

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

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

block_table  

Table block

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

```r
block_table(x = head(iris))

block_table(x = mtcars, header = TRUE,
            properties = prop_table(
              tcf = table conditional formatting(
                first row = TRUE, first column = TRUE)
            ))
```

Description

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

Usage

```r
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 ","
**body_add_blocks**

### See Also

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

### Examples

```r
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

```r
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

```r
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")
  )
)
```
body_add_break  Add a page break in a 'Word' document

Description

add a page break into an rdocx object

Usage

body_add_break(x, pos = "after")

Arguments

x 
an rdocx object

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_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

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

doc <- body_add_caption()
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

doc <- read_docx()

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

---

`body_add_fpar` *Add fpar in a 'Word' document*

Description

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

Usage

`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

bold_face <- shortcuts$fp_bold(font.size = 30)
bold_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(doc, target = tempfile(fileext = ".docx"))

# a way of using fpar to center an image in a Word doc ----
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",
    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"))
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,
  pos = "after",
  ...
)
```

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

```r
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

```r
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

```r
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"))
```
body_add_plot

Add plot in a 'Word' document

Description

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

Usage

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

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

doc <- read_docx()

if (capabilities(what = "png")) {
  doc <- body_add_plot(doc,
    value = plot_instr(
      code = {

Add table in a 'Word' document

Description

Add a table into an rdocx object.

Usage

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

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

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

---

**body_bookmark**

Add **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
body_bookmark(x, id)
```

Arguments

- `x` an rdocx object
- `id` bookmark name

Examples

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

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

body_end_block_section(x, value)

Arguments

x an rdocx object
value a block_section object

Illustrations

See Also

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

Examples

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

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
cstr1 <- "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
cstr1 <- rep(cstr1, 5)
cstr1 <- paste(cstr1, collapse = " ")

doc_1 <- read_docx()
doc_1 <- body_add_par(doc_1, value = cstr1, style = "Normal")
doc_1 <- body_add_par(doc_1, value = cstr1, style = "Normal")
doc_1 <- body_end_section_columns(doc_1)
doc_1 <- body_add_par(doc_1, value = cstr1, 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
doc_1 <- 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
)
```

### 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

```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_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"))
```
**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**

```r
doc_1 <- read_docx()
doc_1 <- body_end_section_continuous(doc_1)
```

**Arguments**

- `x` an rdocx object

**See Also**

Other functions for Word sections: `body_end_block_section()`, `body_end_section_columns()`, `body_end_section_columns_landscape()`, `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
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(), body_end_section_columns_landscape(), body_end_section_continuous(), 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_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)
### body_remove

#### 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(), body_end_section_columns_landscape(), body_end_section_continuous(), body_end_section_landscape(), body_set_default_section()

#### Examples

```r
library(officer)

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**

```r
body_remove(x)
```

**Arguments**

- **x** an rdocx object

**Examples**

```r
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 \texttt{grepl/gsub} for pattern matching; you may supply arguments as required (and therefore use \texttt{regex} features) using the optional \ldots argument.

Note that by default, \texttt{grepl/gsub} will use \texttt{fixed=FALSE}, which means that \texttt{old_value} and \texttt{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 \texttt{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

\begin{verbatim}
body_replace_all_text(
  x, 
  old_value, 
  new_value, 
  only_at_cursor = FALSE, 
  warn = TRUE,
\end{verbatim}
... )

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

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.
  Replacements will be performed in each footer of all sections.

Author(s)

  Frank Hangler, <frank@plotandscatter.com>

See Also

  grep, regex, docx_show_chunk
Examples

```r
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'

# Simple search-and-replace at current cursor, with regex turned off
doc <- body_replace_all_text(doc, old_value = "Placeholder", new_value = "new", only_at_cursor = TRUE, fixed = TRUE)
docx_show_chunk(doc)  # Output is 'new two'

# Do the same, but in the entire document and ignoring case
doc <- body_replace_all_text(doc, old_value = "placeholder", new_value = "new", only_at_cursor=FALSE, ignore.case = TRUE)
doc <- cursor_backward(doc)
docx_show_chunk(doc)  # Output is 'new one'

# Use regex : replace all words starting with "n" with the word "example"
doc <- body_replace_all_text(doc, "\bn.*?\b", "example")
docx_show_chunk(doc)  # Output is 'example one'
```

body_replace_text_at_bkm

Replace text at a bookmark location

Description

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.

Usage

```r
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)
```
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


change_styles

See Also

Other functions for Word sections: `body_end_block_section()`, `body_end_section_columns()`, `body_end_section_columns_landscape()`, `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",  
  pageMargins = 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`

```r
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

```r
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

```r
x <- read_pptx()
color_scheme(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

cursor_begin(x)
cursor_bookmark(x, id)
cursor_end(x)
cursor_reach(x, keyword)
cursor_reach_test(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 reexpr pattern.

cursor_reach_test

Test if an expression has a match in the document that contains text specified in argument keyword. The argument keyword is a reexpr 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**

```r
library(officer)

# create a template ----
doc <- read_docx()
doc <- body_add_par(doc, "blah blah blah")
doc <- body_add_par(doc, "blah blah blah")
doc <- body_add_par(doc, "blah blah blah")
doc <- body_add_par(doc, "Hello text to replace")
doc <- body_add_par(doc, "blah blah blah")
doc <- body_add_par(doc, "blah blah blah")
doc <- body_add_par(doc, "Hello text to replace")
doc <- body_add_par(doc, "blah blah blah")
template_file <- print(
  x = doc,
  target = tempfile(fileext = ".docx"
)
)

# replace all pars containing "to replace" ----
doc <- read_docx(path = template_file)
while (cursor_reach_test(doc, "to replace")) {
  doc <- cursor_reach(doc, "to replace")
  doc <- body_add_fpar(  
    x = doc,
    pos = "on",
    value = fpar(  
      "Here is a link: ",
      hyperlink_ftext(
        text = "yopyop",
        href = "https://cran.r-project.org/"
      )
    )
  )
}

doc <- cursor_end(doc)
doc <- body_add_par(doc, "Yap yap yap yap...")

result_file <- print(
  x = doc,
  target = tempfile(fileext = ".docx"
)
)

# cursor_bookmark ----
doc <- read_docx()
```
**docx_bookmarks**

`docx_bookmarks(x)`

**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())
```
**Description**

return a data.frame representing the comments in a Word document.

**Usage**

docx_comments(x)

**Arguments**

x an rdocx object

**Details**

Each row of the returned data frame contains data for one comment. The columns contain the following information:

- "comment_id" - unique comment id
- "author" - name of the comment author
- "initials" - initials of the comment author
- "date" - timestamp of the comment
- "text" - a list column of characters containing the comment text. Elements can be vectors of length > 1 if a comment contains multiple paragraphs, blocks or runs or of length 0 if the comment is empty.
- "para_id" - a list column of characters containing the parent paragraph IDs. Elements can be vectors of length > 1 if a comment spans multiple paragraphs or of length 0 if the comment has no parent paragraph.
- "commented_text" - a list column of characters containing the commented text. Elements can be vectors of length > 1 if a comment spans multiple paragraphs or runs or of length 0 if the commented text is empty.

**Examples**

```r
bl <- block_list(
  fpar("Comment multiple words."),
  fpar("Second line")
)

a_par <- fpar(
  "This paragraph contains",
  run_comment(
    cmt = bl,
    run = ftext("a comment."),
    author = "Author Me",
  )
)
```
docx_dim

\[
\text{date = "2023-06-01"}
\]

\[
\)
\]

\[
doc <- read_docx()
\]

\[
doc <- body_add_fpar(doc, value = a_par, style = "Normal")
\]

\[
docx_file <- print(doc, target = tempfile(fileext = ".docx"))
\]

\[
docx_comments(read_docx(docx_file))
\]

---

### 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 \texttt{rdocx} object

**See Also**

Other functions for Word document informations: \texttt{doc_properties()}, \texttt{docx_bookmarks()}, \texttt{length.rdocx()}, \texttt{set_doc_properties()}, \texttt{styles_info()}

**Examples**

\[
docx_dim(read_docx())
\]

---

### docx_set_character_style

#### Add character style in a Word document

**Description**

The function lets you add or modify Word character styles.
docx_set_paragraph_style

Add or replace paragraph style in a Word document

Description

The function lets you add or replace a Word paragraph style.

Usage

`docx_set_character_style(
  x,
  style_id,
  style_name,
  base_on,
  fp_t = fp_text_lite()
)

Arguments

- `x`: an rdocx object
- `style_id`: a unique style identifier for Word.
- `style_name`: a unique label associated with the style identifier. This label is the name of the style when Word edits the document.
- `base_on`: the character style name used as base style
- `fp_t`: Text formatting properties, see `fp_text()`.

Examples

```r
library(officer)
doc <- read_docx()
doc <- docx_set_character_style(
  doc,
  style_id = "newcharstyle",
  style_name = "label for char style",
  base_on = "Default Paragraph Font",
  fp_text_lite(
    shading.color = "red",
    color = "white")
)
paragraph <- fpar(
  run_wordtext("hello",
    style_id = "newcharstyle")
)
doc <- body_add_fpar(doc, value = paragraph)
docx_file <- print(doc, target = tempfile(fileext = ".docx"))
docx_file
```
Usage

docx_set_paragraph_style(
  x,
  style_id,
  style_name,
  base_on = "Normal",
  fp_p = fp_par(),
  fp_t = NULL
)

Arguments

x an rdocx object

style_id a unique style identifier for Word.

style_name a unique label associated with the style identifier. This label is the name of the style when Word edit the document.

base_on the style name used as base style

fp_p paragraph formatting properties, see fp_par().

fp_t default text formatting properties. This is used as text formatting properties, see fp_text(). If NULL (default), the paragraph will used the default text formatting properties (defined by the base_on argument).

Examples

library(officer)

doc <- read_docx()

doc <- docx_set_paragraph_style(
  doc,
  style_id = "rightaligned",
  style_name = "Explicit label",
  fp_p = fp_par(text.align = "right", padding = 20),
  fp_t = fp_text_lite(
    bold = TRUE,
    shading.color = "#FD34F0",
    color = "white"
  )
)

doc <- body_add_par(doc,
  value = "This is a test",
  style = "Explicit label")

docx_file <- print(doc, target = tempfile(fileext = ".docx"))
docx_file
**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, preserve = FALSE)

**Arguments**

- **x**
  
an rdocx object

- **preserve**
  
  If FALSE (default), text in table cells is collapsed into a single line. If TRUE, line breaks in table cells are preserved as a "\n" character. This feature is adapted from docxtractr::docx_extract_tbl() published under a MIT licensed in the {docxtractr} package by Bob Rudis.
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)
docx_summary(doc, preserve = TRUE)[28, ]
```

---

**Description**

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

**Usage**

```r
doc_properties(x)
```

**Arguments**

- `x`: an rdocx or rpptx object

**Value**

a data.frame

**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 presentations 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

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)
```

external_img  External image

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

```r
eexternal_img(
  src,
  width = 0.5,
  height = 0.2,
  unit = "in",
  guess_size = FALSE,
  alt = ""
)
```

Arguments

- `src` image file path
- `width`, `height` size of the image file. It can be ignored if parameter `guess_size=TRUE`, see parameter `guess_size`.
- `unit` unit for width and height, one of "in", "cm", "mm".
- `guess_size` If package 'magick' is installed, this option can be used (set it to TRUE). The images will be read and width and height will be guessed.
- `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_columbreak()`, `run_comment()`, `run_footnote()`, `run_footnoteref()`, `run_linebreak()`, `run_pagebreak()`, `run_reference()`, `run_tab()`, `run_word_field()`, `run_wordtext()`

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"),
)```
```r
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_word_field()`) when output is Word, and simple strings.

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

**Usage**

```r
def.par(..., 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

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

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

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

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

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 : See Details for supported border styles.
width | border width - an integer value : 0>= value
object | fp_border object
... | further arguments - not used

Details

For Word output the following border styles are supported:

- "none" or "nil" - No Border
- "solid" or "single" - Single Line Border
• "thick" - Single Line Border
• "double" - Double Line Border
• "dotted" - Dotted Line Border
• "dashed" - Dashed Line Border
• "dotDash" - Dot Dash Line Border
• "dotDotDash" - Dot Dot Dash Line Border
• "triple" - Triple Line Border
• "thinThickSmallGap" - Thin, Thick Line Border
• "thickThinSmallGap" - Thick, Thin Line Border
• "thinThinThickSmallGap" - Thin, Thick, Thin Line Border
• "thinThinThickMediumGap" - Thin, Thick, Thin Line Border
• "thinThinThickLargeGap" - Thin, Thick, Thin Line Border
• "thickThinMediumGap" - Thick, Thin Line Border
• "thickThinLargeGap" - Thick, Thin Line Border
• "thinThinLargeGap" - Thin, Thick Line Border
• "thickThinThinLargeGap" - Thin, Thick, Thin Line Border
• "thickThinLargeGap" - Thick, Thin Line Border
• "thickThinThinLargeGap" - Thin, Thick, Thin Line Border
• "wave" - Wavy Line Border
• "doubleWave" - Double Wave Line Border
• "dashSmallGap" - Dashed Line Border
• "dashDotStroked" - Dash Dot Strokes Line Border
• "threeDEmboss" or "ridge" - 3D Embossed Line Border
• "threeDEngrave" or "groove" - 3D Engraved Line Border
• "outset" - Outset Line Border
• "inset" - Inset Line Border

For HTML output only a limited amount of border styles are supported:

• "none" or "nil" - No Border
• "solid" or "single" - Single Line Border
• "double" - Double Line Border
• "dotted" - Dotted Line Border
• "dashed" - Dashed Line Border
• "threeDEmboss" or "ridge" - 3D Embossed Line Border
• "threeDEngrave" or "groove" - 3D Engraved Line Border
• "outset" - Outset Line Border
• "inset" - Inset Line Border

Non-supported Word border styles will default to "solid".
**fp_cell**

**See Also**

Other functions for defining formatting properties: `fp_cell()`, `fp_par()`, `fp_tab()`, `fp_tabs()`, `fp_text()`

**Examples**

```r
fp_border()
fp_border(color = "orange", style = "solid", width = 1)
fp_border(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",
  colspan = 1,
  rowspan = 1
)
```

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

```r
## S3 method for class 'fp_cell'
print(x, ...)
```
## S3 method for class 'fp_cell'
update(
    object,
    border,
    border.bottom, border.left, border.top, border.right,
    vertical.align, margin = 0,
    margin.bottom, margin.top, margin.left, margin.right,
    background.color, text.direction,
    rowspan = 1,
    colspan = 1,
    ...
)

### 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 "lrb", "tlbr", "btlr".
- **rowspan**: specify how many rows the cell is spanned over
- **colspan**: specify how many columns the cell is spanned over
- **x, object**: `fp_cell` object
- **type**: output type - one of 'wml', 'pml', 'html', 'rtf'.
- **...**: further arguments - not used

### See Also

Other functions for defining formatting properties: `fp_border()`, `fp_par()`, `fp_tab()`, `fp_tabs()`, `fp_text()`
Examples

```r
obj <- fp_cell(margin = 1)
update(obj, margin.bottom = 5)
```

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,
  tabs = NULL,
  word_style = "Normal"
)
```

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

```r
## 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.right,
  shading.color = "transparent",
  keep_with_next = FALSE,
  tabs = NULL,
  word_style = "Normal"
)
```
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.

- **tabs**  
  NULL (default) for no tabulation marks setting or an object returned by `fp_tabs()`. Note this can only have effect with Word or RTF outputs.

- **word_style**  
  Word paragraph style name

- **x, object**  
  `fp_par` object

- **...**  
  further arguments - not used

Value

- a `fp_par` object

See Also

- `fpar`

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

Examples

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

Tabulation mark properties object

**Description**

create a tabulation mark properties setting object for Word or RTF. Results can be used as arguments of `fp_tabs()`.

Once tabulation marks settings are defined, tabulation marks can be added with `run_tab()` inside a call to `fpar()` or with `\t` within 'flextable' content.

**Usage**

```r
fp_tab(pos, style = "decimal")
```

**Arguments**

- `pos` Specifies the position of the tab stop (in inches).
- `style` style of the tab. Possible values are: "decimal", "left", "right" or "center".

**See Also**

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

**Examples**

```r
fp_tab(pos = 0.4, style = "decimal")
fp_tab(pos = 1, style = "right")
```

**fp_tabs**

Tabs properties object

**Description**

create a set of tabulation mark properties object for Word or RTF. Results can be used as arguments of `fp_par()` and will only have effects in Word or RTF outputs.

Once a set of tabulation marks settings is defined, tabulation marks can be added with `run_tab()` inside a call to `fpar()` or with `\t` within 'flextable' content.

**Usage**

```r
fp_tabs(...)```

**Arguments**

```r
... fp_tab objects```

...
See Also

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

Examples

```r
z <- fp_tabs(
  fp_tab(pos = 0.4, style = "decimal"),
  fp_tab(pos = 1, style = "decimal")
)

fpar(
  run_tab(), ftext("88."),
  run_tab(), ftext("987.45"),
  fp_p = fp_par(
    tabs = z
  )
)
```

### fp_text

<table>
<thead>
<tr>
<th>Text formatting properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a <code>fp_text</code> object that describes text formatting properties.</td>
</tr>
<tr>
<td>Function <code>fp_text_lite()</code> is generating properties with only entries for the parameters users provided. The undefined properties will inherit from the default settings.</td>
</tr>
</tbody>
</table>

#### Usage

```r
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"
)
```

```r
fp_text_lite(
  color = NA,
  font.size = NA,
  font.family = NA,
```
\begin{verbatim}

fp_text

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,
  shading.color,
  ...
)

Arguments

- **color**
  - font color - a single character value specifying a valid color (e.g. "#000000" or "black").

- **font.size**
  - font size (in point) - 0 or positive integer value.

- **bold**
  - is bold

- **italic**
  - is italic

- **underlined**
  - is underlined

- **font.family**
  - single character value. Specifies the font to be used to format characters in the Unicode range (U+0000-U+007F).

- **cs.family**
  - 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.

- **eastasia.family**
  - 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.
\end{verbatim}
hansi.family  
optional. Specifies the font to be used to format characters in a Unicode range which does not fall into one of the other categories.

vertical.align  
single character value specifying font vertical alignments. Expected value is one of the following: default 'baseline' or 'subscript' or 'superscript'.

shading.color  
shading color - a single character value specifying a valid color (e.g. "#000000" or "black").

Value  
a fp_text object

See Also  
ftext, fpar

Other functions for defining formatting properties: fp_border(), fp_cell(), fp_par(), fp_tab(), fp_tabs()

Examples  
  fp_text()
  fp_text(color = "red")
  fp_text(bold = TRUE, shading.color = "yellow")
  print(fp_text(color = "red", font.size = 12))

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  
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).
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_columbreak()`, `run_comment()`, `run_footnote()`, `run_footnoteref()`, `run_linebreak()`, `run_pagebreak()`, `run_reference()`, `run_tab()`, `run_word_field()`, `run_wordtext()`

**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"))
```

---

**hyperlink_ftext**  
*Formatted chunk of text with hyperlink*

**Description**

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

**Usage**

```r
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 \textit{r} chunk in an R Markdown document made with package officedown.

See Also

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

Examples

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

---

layout_properties

\textit{Slide layout properties}

Description

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

Usage

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

Arguments

- \texttt{x}: an \texttt{rpptx} object
- \texttt{layout}: slide layout name to use
- \texttt{master}: master layout name where \texttt{layout} is located

See Also

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

Examples

```r
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

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)
```

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 a new document produced
# with the default template.
length( read_docx() )
```

<table>
<thead>
<tr>
<th>length.rpptx</th>
<th>Number of slides</th>
</tr>
</thead>
</table>

Description

Function `length` will return the number of slides.

Usage

```r
## S3 method for class 'rpptx'
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)
my_pres <- add_slide(my_pres)
length(my_pres)
```

media_extract

Extract media from a document object

Description

Extract files from a `rpptx` object.

Usage

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

Arguments

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

Examples

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

move_slide(x, index = NULL, 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**

```r
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**

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

**Arguments**

- `type`
  - placeholder label of the used notes master
- `...`
  - unused arguments
Manipulate Microsoft Word and PowerPoint Documents with 'officer'

Description

The officer package facilitates access to and manipulation of 'Microsoft Word' and 'Microsoft PowerPoint' documents from R. It also supports the writing of 'RTF' documents.

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(), read_pptx() or rtf_doc().

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>

Authors:

- Stefan Moog <moogs@gmx.de>

Other contributors:

- ArData [copyright holder]
- Frank Hangler <frank@plotandscatter.com> (function body_replace_all_text) [contributor]
- Liz Sander <lsander@civisanalytics.com> (several documentation fixes) [contributor]
- Anton Victorson <anton@victorson.se> (fixes xml structures) [contributor]
- Jon Calder <jonncalder@gmail.com> (update vignettes) [contributor]
- John Harrold <john.m.harrold@gmail.com> (function annotate_base) [contributor]
- John Muschelli <muschelli2@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]
- Greg Leleu <gregoire.leleu@gmail.com> (fields functionality in ppt) [contributor]
- Hongyuan Jia <hongyuanjia@cqust.edu.cn> (ORCID) [contributor]
Defunct Functions in Package officer

Usage

slip_in_seqfield(...)
slip_in_column_break(...)
slip_in_xml(...)
slip_in_text(...)
slip_in_footnote(...)

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().
slip_in_footnote() is replaced by run_footnote().
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**

```r
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/top 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.
- `gutter` page gutter (in inches).

**See Also**

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

**Examples**

```r
page_mar()
```
**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")
```

---

**ph_hyperlink**  
Hyperlink a placeholder

**Description**

Add hyperlink to a placeholder in the current slide.

**Usage**

```r
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`. If used, type and id are ignored.

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") )

# 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") )
```
\textit{ph\_location\_fullsize} \hspace{1cm} \textit{Location of a full size element}

\textbf{Description}

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

\textbf{Usage}

\begin{verbatim}
ph_location_fullsize(newlabel = "", ...)
\end{verbatim}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{newlabel} \hspace{1cm} a label to associate with the placeholder.
  \item \texttt{...} \hspace{1cm} unused arguments
\end{itemize}

\textbf{See Also}

Other functions for placeholder location: \texttt{ph\_location()}, \texttt{ph\_location\_label()}, \texttt{ph\_location\_left()}, \texttt{ph\_location\_right()}, \texttt{ph\_location\_template()}, \texttt{ph\_location\_type()}

\textbf{Examples}

\begin{verbatim}
doc <- read_pptx()
doc <- add\_slide(doc)
doc <- ph\_with(doc, "Hello world", location = ph\_location\_fullsize() )
print(doc, target = tempfile(fileext = ".pptx") )
\end{verbatim}

\textit{ph\_location\_label} \hspace{1cm} \textit{Location of a named placeholder}

\textbf{Description}

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

\textbf{Usage}

\begin{verbatim}
ph_location_label(ph\_label, newlabel = NULL, ...)
\end{verbatim}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{ph\_label} \hspace{1cm} placeholder label of the used layout. It can be read in PowerPoint or with function \texttt{layout\_properties()} in column \texttt{ph\_label}.
  \item \texttt{newlabel} \hspace{1cm} a label to associate with the placeholder.
  \item \texttt{...} \hspace{1cm} unused arguments
\end{itemize}
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()`, `ph_location_fullsize()`, `ph_location_left()`,
`ph_location_right()`, `ph_location_template()`, `ph_location_type()`

Examples

```r
# 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 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

```
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()`, `ph_location_fullsize()`, `ph_location_label()`, `ph_location_right()`, `ph_location_template()`, `ph_location_type()

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") )
```

---

**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

```
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()`, `ph_location_fullsize()`, `ph_location_label()`, `ph_location_right()`, `ph_location_template()`, `ph_location_type()`
**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`.

**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', 'ftr', 'sldNum'. It will be used as a template placeholder.
- `id`: index of the placeholder template. If two body placeholder, 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:
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()`, `ph_location_fullsize()`, `ph_location_label()`, `ph_location_left()`, `ph_location_right()`, `ph_location_type()`

**Examples**

```r
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', 'ftr', '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.

**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()**, **ph_location_fullsize()**, **ph_location_label()**, **ph_location_left()**, **ph_location_right()**, **ph_location_template()**

**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`. If used, type and id are ignored.

- **id_chr**  
deprecated.

**See Also**

- **ph_with**

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

```r
fileout <- tempfile(fileext = ".pptx")
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)
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",
)```
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`. If used, type and id are ignored.
- **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 <- add_slide(doc)
doc <- ph_with(x = doc, "Un titre 2", 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

\texttt{ph\_with(x, value, location, \ldots)}

## S3 method for class 'character'
\texttt{ph\_with(x, value, location, \ldots)}

## S3 method for class 'numeric'
\texttt{ph\_with(x, value, location, format\_fun = format, \ldots)}

## S3 method for class 'factor'
\texttt{ph\_with(x, value, location, \ldots)}

## S3 method for class 'logical'
\texttt{ph\_with(x, value, location, format\_fun = format, \ldots)}

## S3 method for class 'block\_list'
\texttt{ph\_with(x, value, location, level\_list = integer(\emptyset), \ldots)}

## S3 method for class 'unordered\_list'
\texttt{ph\_with(x, value, location, \ldots)}

## S3 method for class 'data\_frame'
\texttt{ph\_with(x, value, location, header = TRUE, tcf = table\_conditional\_formatting(), alignment = NULL, \ldots)}

## S3 method for class 'gg'
\texttt{ph\_with(x, value, location, res = 300, alt\_text, scale = 1, \ldots)}

## S3 method for class 'plot\_instr'
\texttt{ph\_with(x, value, location, res = 300, \ldots)}

## S3 method for class 'external\_img'
\texttt{ph\_with(x, value, location, use\_loc\_size = TRUE, \ldots)}

## S3 method for class 'fpar'
\texttt{ph\_with(x, value, location, \ldots)}

## S3 method for class 'empty\_content'
\texttt{ph\_with(x, value, location, \ldots)}

## S3 method for class 'xml\_document'
\texttt{ph\_with(x, value, location, \ldots)}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{x} \quad \text{an \texttt{rpptx} object}
  \item \texttt{value} \quad \text{object to add as a new shape. Supported objects are vectors, \texttt{data.frame}, \texttt{graphics}, block of formatted paragraphs, unordered list of formatted paragraphs, pretty tables with package \texttt{flextable}, editable graphics with package \texttt{rvg}. ‘Microsoft’ charts with package \texttt{mschart}.}
  \item \texttt{location} \quad \text{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 \texttt{ph\_location} functions. See section "see also".}
  \item \texttt{\ldots} \quad \text{further arguments passed to or from other methods. When adding a ggplot object or \texttt{plot\_instr}, these arguments will be used by \texttt{png} function.}
  \item \texttt{format\_fun} \quad \text{format function for non character vectors}
  \item \texttt{level\_list} \quad \text{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.}
  \item \texttt{header} \quad \text{display header if \texttt{TRUE}}
  \item \texttt{tcf} \quad \text{conditional formatting settings defined by \texttt{table\_conditional\_formatting()}}
  \item \texttt{alignment} \quad \text{alignment for each columns, ‘l’ for left, ‘r’ for right and ‘c’ for center. Default to \texttt{NULL}.}
  \item \texttt{res} \quad \text{resolution of the \texttt{png} image in \texttt{ppi}}
  \item \texttt{alt\_text} \quad \text{Alt-text for screen-readers}
  \item \texttt{scale} \quad \text{Multiplicative scaling factor, same as in \texttt{ggsave}}
  \item \texttt{use\_loc\_size} \quad \text{if set to \texttt{FALSE}, \texttt{external\_img} width and height will be used.}
\end{itemize}

\textbf{Methods (by class)}

\begin{itemize}
  \item \texttt{ph\_with(character)}: add a character vector to a new shape on the current slide, values will be added as paragraphs.
  \item \texttt{ph\_with(numeric)}: add a numeric vector to a new shape on the current slide, values will be be first formatted then added as paragraphs.
  \item \texttt{ph\_with(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.
  \item \texttt{ph\_with(block\_list)}: add a \texttt{block\_list} made of \texttt{fpar} to a new shape on the current slide.
  \item \texttt{ph\_with(unordered\_list)}: add a \texttt{unordered\_list} made of \texttt{fpar} to a new shape on the current slide.
  \item \texttt{ph\_with(data\_frame)}: add a \texttt{data.frame} to a new shape on the current slide with function \texttt{block\_table()}. Use package \texttt{flextable} instead for more advanced formatings.
  \item \texttt{ph\_with(gg)}: add a ggplot object to a new shape on the current slide. Use package \texttt{rvg} for more advanced graphical features.
  \item \texttt{ph\_with(plot\_instr)}: add an R plot to a new shape on the current slide. Use package \texttt{rvg} for more advanced graphical features.
\end{itemize}
The function \texttt{ph\_with} can be used in various ways:

- \texttt{ph\_with(external\_img)}: add an \texttt{external\_img} to a new shape on the current slide. When value is a \texttt{external\_img} object, image will be copied into the PowerPoint presentation. The width and height specified in call to \texttt{external\_img} will be ignored, their values will be those of the location, unless \texttt{use\_loc\_size} is set to \texttt{FALSE}.
- \texttt{ph\_with(fpar)}: add an \texttt{fpar} to a new shape on the current slide as a single paragraph in a \texttt{block\_list}.
- \texttt{ph\_with(empty\_content)}: add an \texttt{empty\_content} to a new shape on the current slide.
- \texttt{ph\_with(xml\_document)}: add an \texttt{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**

\texttt{ph\_location\_type}, \texttt{ph\_location}, \texttt{ph\_location\_label}, \texttt{ph\_location\_left}, \texttt{ph\_location\_right}, \texttt{ph\_location\_fullsize}, \texttt{ph\_location\_template}

**Examples**

```r
# 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")
})
```


# 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 ----
if (require("rsvg")) {
  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")
  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(0), use_loc_size = FALSE
  )
  doc_1
### ph_with

```r
doc_1 <- add_slide(doc_1)
doc_1 <- ph_with(
  x = doc_1, value = bl,
  location = ph_location_type(type = "body")
)

# fpar -----  

t <- 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(
```

plot_instr

Wrap plot instructions for png plotting in Powerpoint or Word

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
# 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", pointsize = 12)
print(doc, target = tempfile(fileext = ".pptx"))
plot_layout_properties

Slide layout properties plot

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" )

pptx_summary

PowerPoint content in a data.frame

Description

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

Usage

pptx_summary(x, preserve = FALSE)
print.rpptx

Arguments

x  an rpptx object

preserve  If FALSE (default), text in table cells is collapsed into a single line. If TRUE, line
            breaks in table cells are preserved as a "\n" character. This feature is adapted
            from docxtractr::docx_extract_tbl() published under a MIT licensed in
            the {docxtractr} package by Bob Rudis.

Examples

example_pptx <- system.file(package = "officer",
                            "doc_examples/example.pptx")
doc <- read_pptx(example_pptx)
print.rpptx
ppta_summary(doc)
pptx_summary(example_pptx)

print.rpptx  Write a 'PowerPoint' file.

Description

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

Usage

## 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

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

Description

Write the RTF object and its content to a file.

Usage

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

Arguments

- `x`: an 'rtf' object created with `rtf_doc()`
- `target`: path to the RTF file to write
- `...`: unused

See Also

- `rtf_doc()`

Examples

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

---

prop_section  

Section properties

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.
prop_section

Usage

prop_section(
  page_size = NULL,
  pageMargins = NULL,
  type = NULL,
  section_columns = NULL,
  header_default = NULL,
  header_even = NULL,
  header_first = NULL,
  footer_default = NULL,
  footer_even = NULL,
  footer_first = 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. Use NULL (default value) for no content.

header_default
  content as a block_list() for the default page header. Use NULL (default value) for no content.

header_even
  content as a block_list() for the even page header. Use NULL (default value) for no content.

header_first
  content as a block_list() for the first page header. Use NULL (default value) for no content.

footer_default
  content as a block_list() for the default page footer. Use NULL (default value) for no content.

footer_even
  content as a block_list() for the even page footer. Use NULL (default value) for no content.

footer_first
  content as a block_list() for the default page footer. Use NULL (default value) for no content.

Illustrations

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"))

# an example with headers and footers -----

txt_lorem <- rep("Purus lectus eros metus turpis mattis platea praesent sed. ",
  50)
txt_lorem <- paste0(txt_lorem, collapse = "")

header_first <- block_list(fpar(ftext("text for first page header")))
header_even <- block_list(fpar(ftext("text for even page header")))
header_default <- block_list(fpar(ftext("text for default page header")))
footer_first <- block_list(fpar(ftext("text for first page footer")))
footer_even <- block_list(fpar(ftext("text for even page footer")))
footer_default <- block_list(fpar(ftext("text for default page footer")))

ps <- prop_section(
  header_default = header_default, footer_default = footer_default,
  header_first = header_first, footer_first = footer_first,
  header_even = header_even, footer_even = footer_even
)
x <- read_docx()
for (i in 1:20) {
  x <- body_add_par(x, value = txt_lorem)
}

x <- body_set_default_section(
  x,
  value = ps
)

print(x, 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",
  word_title = NULL,
  word_description = NULL
)
```

**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)
- `word_title` : alternative text for Word table (used as title of the table)
- `word_description` : alternative text for Word table (used as description of the table)
See Also

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

Examples

prop_table()
to_wml(prop_table())

---

**Description**

`read_docx()` reads and imports a docx file as an R object representing the document. When no file is specified, it uses a default empty file.

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

**Usage**

```r
read_docx(path = NULL)
```

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

**Arguments**

- `path`: path to the docx file to use as base document. dotx file are supported.
- `x`: an rdocx object
- `target`: path to the docx file to write
- `...`: unused

**Value**

an object of class rdocx.

**Functions**

- `print(rdocx)`: 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

\texttt{body_add_par}, \texttt{body_add_plot}, \texttt{body_add_table}

Examples

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"))

\newpage

\section*{read_pptx}

Create a ‘PowerPoint’ document object

\subsection*{Description}

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

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

\subsection*{Usage}

\begin{verbatim}
read_pptx(path = NULL)
\end{verbatim}
Arguments

path path to the pptx file to use as base document. potx file are supported.

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()
Arguments

path       path to the xlsx file to use as base document.
x         an rxlsx object
target      path to the xlsx file to write
...        unused

Examples

```r
read_xlsx()
x <- read_xlsx()
print(x, target = tempfile(fileext = ".xlsx"))
```

---

**remove_slide**

Remove a slide

Description

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)
```
rtf_add

Add content into an RTF document

Description

This function adds 'officer' objects into an RTF document. Values are added as new paragraphs. See section 'Methods (by class)' that list supported objects.

Usage

rtf_add(x, value, ...)

## S3 method for class 'block_section'
rtf_add(x, value, ...)

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

## S3 method for class 'factor'
rtf_add(x, value, ...)

## S3 method for class 'double'
rtf_add(x, value, formatter = formatC, ...)

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

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

## S3 method for class 'gg'
rtf_add(
    x,
    value,
    width = 6,
    height = 5,
    res = 300,
    scale = 1,
    ppr = fp_par(text.align = "center"),
    ...
)

## S3 method for class 'plot_instr'
rtf_add(
    x,
    value,
    width = 6,
Arguments

- **x**: rtf object, created by `rtf_doc()`. 
- **value**: object to add in the document. Supported objects are vectors, graphics, block of formatted paragraphs. Use package 'flextable' to add tables.
- **...**: further arguments passed to or from other methods. When adding a ggplot object or `plot_instr`, these arguments will be used by `png` function. See section 'Methods' to see what arguments can be used.
- **formatter**: function used to format the numerical values
- **width**: height in inches
- **height**: height in inches
- **res**: resolution of the png image in ppi
- **scale**: Multiplicative scaling factor, same as in `ggsave`
- **ppr**: `fp_par()` to apply to paragraph.

Methods (by class)

- `rtf_add(block_section)`: add a new section definition
- `rtf_add(character)`: add characters as new paragraphs
- `rtf_add(factor)`: add a factor vector as new paragraphs
- `rtf_add(double)`: add a double vector as new paragraphs
- `rtf_add(fpar)`: add an `fpar()`
- `rtf_add(block_list)`: add a `block_list()`
- `rtf_add(plot_instr)`: add a `plot_instr()` object

Examples

```r
library(officer)

def_text <- fp_text_lite(color = "#006699", bold = TRUE)
center_par <- fp_par(text.align = "center", padding = 3)

doc <- rtf_doc(
  normal_par = fp_par(line_spacing = 1.4, padding = 3)
)

doc <- rtf_add(
  height = 5,
  res = 300,
  scale = 1,
  ppr = fp_par(text.align = "center"),
  ...
)
```

Create an RTF document object

Description

Creation of the object representing an RTF document which can then receive contents with the rtf_add() function and be written to a file with the print(x, target="doc.rtf") function.

Usage

rtf_doc
def_sec = prop_section(),
normal_par = fp_par(),
normal_chunk = fp_text(font.family = "Arial", font.size = 11)
)

Arguments

def_sec a block_section object used to defined default section.

normal_par an object generated by fp_par()

normal_chunk an object generated by fp_text()

Value

an object of class rtf representing an empty RTF document.

See Also

read_docx(), print.rtf(), rtf_add()

Examples

rtf_doc(normal_par = fp_par(padding = 3))

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 = ":-
)
run_autonum

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_comment(), run_footnote(), run_footnoteref(), run_linebreak(), run_pagebreak(), run_reference(), run_tab(), run_word_field(), run_wordtext()

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

Examples

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 = " "
)
Description

Add a bookmark on a run object.

Usage

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_comment(), run_footnote(), run_footnoteref(), run_linebreak(), run_pagebreak(), run_reference(), run_tab(), run_word_field(), run_wordtext()

Examples

```r
ft <- fp_text(font.size = 12, bold = TRUE)
run_bookmark("par1", ftext("some text", ft))
```

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()`, `runBookmark()`, `run_comment()`, `run_footnote()`, `run_footnoteref()`, `run_linebreak()`, `run_pagebreak()`, `run_reference()`, `run_tab()`, `run_word_field()`, `run_wordtext()`

Examples

```r
run_columnbreak()
```

---

**run_comment**  
*Comment for 'Word'*

Description

Add a comment on a run object.

Usage

```r
run_comment(
  cmt,
  run = ftext(""),
  author = "",
  date = "",
  initials = "",
  prop = NULL
)
```

Arguments

- `cmt`: a set of blocks to be used as comment content returned by function `block_list()`. the "run functions for reporting".
- `run`: a run object, made with a call to one of
- `author`: comment author.
- `date`: comment date
- `initials`: comment initials
- `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_footnote()`, `run_footnoteref()`, `run_linebreak()`, `run_pagebreak()`, `run_reference()`, `run_tab()`, `run_word_field()`, `run_wordtext()`

Examples

```r
fp_bold <- fp_text_lite(bold = TRUE)
fp_red <- fp_text_lite(color = "red")

bl <- block_list(
  fpar(ftext("Comment multiple words.", fp_bold)),
  fpar(
    ftext("Second line.", fp_red)
  )
)

comment1 <- run_comment(
  cmt = bl,
  run = ftext("with a comment"),
  author = "Author Me",
  date = Sys.Date(),
  initials = "AM"
)
par1 <- fpar("A paragraph ", comment1)

bl <- block_list(
  fpar(ftext("Comment a paragraph.")))  
)

comment2 <- run_comment(
  cmt = bl, run = ftext("A commented paragraph"),
  author = "Author You",
  date = Sys.Date(),
  initials = "AY"
)
par2 <- fpar(comment2)

doc <- read_docx()
doc <- body_add_fpar(doc, value = par1, style = "Normal")
doc <- body_add_fpar(doc, value = par2, style = "Normal")

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

---

**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_comment(), run_footnoteref(), run_linebreak(), run_pagebreak(), run_reference(), run_tab(), run_word_field(), run_wordtext()

Examples

library(officer)
fp_bold <- 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

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).

See Also

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

Examples

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

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_comment(), run_footnote(), run_footnoteref(), run_pagebreak(), run_reference(), run_tab(), run_word_field(), run_wordtext()

Examples

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_comment(), run_footnote(), run_footnoteref(), run_linebreak(), run_reference(), run_tab(), run_word_field(), run_wordtext()

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 \textit{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_columnsbreak()}, \texttt{run_comment()}, \texttt{run_footnote()}, \texttt{run_footnoteref()}, \texttt{run_linebreak()}, \texttt{run_pagebreak()}, \texttt{run_tab()}, \texttt{run_word_field()}, \texttt{run_wordtext()}

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

Examples

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

\begin{verbatim}
run_tab
\end{verbatim}

Tab for 'Word'

Description

Object representing a tab in a Word document. The result must be used within a call to \texttt{fpar}. It will only have effects in Word output.

Tabulation marks settings can be defined with \texttt{fp_tabs()} in paragraph settings defined with \texttt{fp_par()}. 

Usage

\begin{verbatim}
run_tab()
\end{verbatim}

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 \textit{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_columnsbreak()}, \texttt{run_comment()}, \texttt{run_footnote()}, \texttt{run_footnoteref()}, \texttt{run_linebreak()}, \texttt{run_pagebreak()}, \texttt{run_reference()}, \texttt{run_word_field()}, \texttt{run_wordtext()}
Examples

```r
z <- fp_tabs(  
  fp_tab(pos = 0.5, style = "decimal"),  
  fp_tab(pos = 1.5, style = "decimal")
)
par1 <- fpar(  
  run_tab(), ftext("88."),  
  run_tab(), ftext("987.45"),  
  fp_p = fp_par(  
    tabs = z  
  )
)
par2 <- fpar(  
  run_tab(), ftext("8."),  
  run_tab(), ftext("670987.45"),  
  fp_p = fp_par(  
    tabs = z  
  )
)
)
x <- read_docx()
x <- body_add(x, par1)
x <- body_add(x, par2)
print(x, target = tempfile(fileext = ".docx"))
```

---

**run_wordtext**  
*Word chunk of text with a style*

**Description**

Format a chunk of text associated with a 'Word' character style. The style is defined with its unique identifier.

**Usage**

```r
run_wordtext(text, style_id = NULL)
```

**Arguments**

- `text`  
  text value, a single character value

- `style_id`  
  'Word' unique style identifier associated with the style to use.

**See Also**

`ftext()`

Other run functions for reporting: `external_img()`, `ftext()`, `hyperlink_ftext()`, `run_autonum()`, `run_bookmark()`, `run_colummbreak()`, `run_comment()`, `run_footnote()`, `run_footnoteref()`, `run_linebreak()`, `run_pagebreak()`, `run_reference()`, `run_tab()`, `run_word_field()`
Examples

```r
run1 <- run_wordtext("hello", "DefaultParagraphFont")
paragraph <- fpar(run1)

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

---

### Description

Create a 'Word' computed field.

### Usage

```r
run_word_field(field, prop = NULL, seqfield = NULL)
run_seqfield(field, prop = NULL, seqfield = NULL)
```

### Arguments

- **field**: Value for a "Word Computed Field" as a string.
- **prop**: formatting text properties returned by `fp_text`.
- **seqfield**: deprecated in favor of `field`.

### 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.

### Note

In the previous version, this function was called `run_seqfield` but the name was wrong and should have been `run_word_field`.

### See Also

Other run functions for reporting: `external_img()`, `ftext()`, `hyperlink_ftext()`, `run_autonum()`, `run_bookmark()`, `run_columnbreak()`, `run_comment()`, `run_footnote()`, `run_footnoteref()`, `run_linebreak()`, `run_pagebreak()`, `run_reference()`, `run_tab()`, `run_wordtext()`.  
Other Word computed fields: `run_autonum()`, `run_reference()`

### Examples

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


**section_columns**

*Description*

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

*Usage*

\[
\text{section_columns}(\text{widths = c(2.5, 2.5)}, \space \text{space = 0.25}, \space \text{sep = FALSE})
\]

*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*

\[
\text{section_columns()}
\]

**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*

\[
\text{set_autonum_bookmark}(x, \space bkm = \text{NULL})
\]

*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")
```

Description

set_doc_properties  Set document properties

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.

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

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)

- set_notes(character): add a character vector to a place holder in the notes on the current slide, values will be added as paragraphs.
- set_notes(block_list): add a block_list to a place holder 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

```r
# 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")
fpt_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")
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.
**shortcuts**

**Usage**

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

**Arguments**

- `x`  
  - `xlsx` 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**

```r
shortcuts
```

**Examples**

```r
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**

```r
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

```r
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

```r
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_summary()`

Examples

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

```
```

```r
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)
```
sp_line

**Line properties**

**Description**

Create a `sp_line` object that describes line properties.

**Usage**

```r
sp_line(
  color = "transparent",
  lwd = 1,
  lty = "solid",
  linemdp = "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, 
  linemdp, 
  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'.

- **linemdp**
- **lineend**
- **linejoin**
- **headend**
- **tailend**
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  

- **type**  
  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'

- **width**  
  single character value specifying the line end width. Expected value is one of the following: default 'sm' or 'med' or 'lg'

- **length**  
  single character value specifying the line end length. Expected value is one of the following: default 'sm' or 'med' or 'lg'

- **x, object**  
  `sp_lineend` object

- **...**  
  further arguments - not used

Value  

- a `sp_lineend` object

See Also  

- `sp_line`

Other functions for defining shape properties: `sp_line()`

Examples  

```
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**  

```
styless_info( 
  x, 
  type = c("paragraph", "character", "table", "numbering"), 
  is_default = c(TRUE, FALSE) 
)
```
**table_colwidths**

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()
styles_info(x)
styles_info(x, type = "paragraph", is_default = TRUE)
```

---

**table_colwidths**  
*Column widths of a table*

Description

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

Usage

```r
table_colwidths(widths = NULL)
```

Arguments

- **widths**
  - Column widths expressed in inches.

See Also

- Other functions for table definition: `prop_table()`, `table_conditional_formatting()`, `table_layout()`, `table_stylenames()`, `table_width()`
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.

Usage

```r
table_conditional_formatting(
  first_row = TRUE,
  first_column = FALSE,
  last_row = FALSE,
  last_column = FALSE,
  no_hband = FALSE,
  no_vband = TRUE
)
```

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

```r
table_conditional_formatting(first_row = TRUE, first_column = TRUE)
```
Algorithm for table layout

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

```r
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()`

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_stylenames()`, `table_width()`
Examples

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)
)

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

---

table_width

Preferred width for a table

Description

Define the preferred width for a table.

Usage

\texttt{table_width(width = 1, unit = "pct")}

Arguments

- \texttt{width} \hspace{1em} value of the preferred width of the table.
- \texttt{unit} \hspace{1em} 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.
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

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, 100
  run_reference, 107
  run_word_field, 110

* block functions for reporting
  block_caption, 8
  block_list, 9
  block_pour_docx, 10
  block_section, 11
  block_table, 11
  block_toc, 12
  fpar, 48
  plot_instr, 86
  unordered_list, 124

* functions for Word document informations
  doc_properties, 45
  docx_bookmarks, 39
  docx_dim, 41
  length.rdocx, 61
  set_doc_properties, 112
  styles_info, 119

* functions for Word sections
  body_end_block_section, 25
  body_end_section_columns, 26
  body_end_section_columns_landscape, 27
  body_end_section_continuous, 28
  body_end_section_landscape, 29
  body_end_section_portrait, 29
  body_set_default_section, 34

* functions for adding content
  body_add_blocks, 13
  body_add_break, 14
  body_add_caption, 14
  body_add_docx, 15
  body_add_fpar, 16
  body_add_gg, 18
  body_add_img, 19
  body_add_par, 20
  body_add_plot, 21
  body_add_table, 22
  body_add_toc, 23

* functions for defining formatting properties
  fp_border, 49
  fp_cell, 51
  fp_par, 53
  fp_tab, 55
  fp_tabs, 55
  fp_text, 56

* functions for defining shape properties
  sp_line, 117
  sp_lineend, 118

* functions for placeholder location
  ph_location, 70
  ph_location_fullsize, 72
  ph_location_label, 72
  ph_location_left, 73
  ph_location_right, 74
  ph_location_template, 75
  ph_location_type, 76

* functions for placeholders manipulation
  ph_hyperlink, 69
  ph_remove, 78
  ph_slidelink, 79

* functions for reading presentation informations
  annotate_base, 6
  color_scheme, 36
  doc_properties, 45
  layout_properties, 60
  layout_summary, 61
  length.rpptx, 62
  plot_layout_properties, 87
  slide_size, 115
  slide_summary, 116

* functions for section definition
INDEX

127

doc_properties, 6, 36, 39, 41, 45, 60–62, 87, 113, 116, 120

hyperlink_ftext, 47, 59, 59, 101–110

layout_properties, 6, 36, 45, 60, 61, 62, 87, 116

layout_summary, 6, 36, 45, 60, 61, 62, 87, 116

layout_summary(), 5

length.rdocx, 39, 41, 45, 61, 113, 120

length.rpptx, 36, 45, 60, 61, 62, 87, 116

length.rxlsx (read_xlsx), 95

media_extract, 62

move_slide, 5, 63, 67, 96, 114

notes_location_label, 64

notes_location_label(), 114

notes_location_type, 64

notes_location_type(), 114

officer, 65

officer-defunct, 66

officer-package (officer), 65

on_slide, 5, 63, 67, 96, 114

page_mar, 68, 69, 90, 111

page_size, 68, 69, 90, 111

ph_hyperlink, 69, 78, 80

ph_location, 70, 72–74, 76, 77, 83

ph_location_fullsize, 71, 72, 73, 74, 76, 77, 83

ph_location_label, 71, 72, 72, 74, 76, 77, 83

ph_location_left, 71–73, 73, 74, 76, 77, 83

ph_location_right, 71–74, 74, 76, 77, 83

ph_location_template, 71–74, 75, 77, 83

ph_location_type, 71–74, 76, 83

ph_location_type(), 73, 74

ph_remove, 70, 78, 80

ph_remove(), 96

ph_slidelink, 70, 78, 79

ph_with, 47, 70, 78, 80, 80, 124

ph_with(), 5, 9, 46, 48, 67, 86, 95, 96

plot_instr, 8–13, 48, 86, 98, 124

plot_instr(), 9, 21, 98

plot_layout_properties, 6, 36, 45, 60–62, 87, 116

plot_layout_properties(), 5, 95

pptx_summary, 87

print.fp_cell (fp_cell), 51

print.fp_par (fp_par), 53

print.fp_text (fp_text), 56

empty_content, 46, 83

external_img, 46, 59, 60, 83, 101–110

external_img(), 9, 48

footers_replace_all_text

(body_replace_all_text), 31

footers_replace_img_at_bkm

(body_replace_text_at_bkm), 33

footers_replace_text_at_bkm

(body_replace_text_at_bkm), 33

format.fp_cell (fp_cell), 51

format.fp_text (fp_text), 56

fp_border, 49, 52, 54–56, 58

fp_cell, 51, 51, 54–56, 58

fp_par, 51, 52, 53, 55, 56, 58

fp_par(), 43, 48, 55, 98, 100, 108

fp_tab, 51, 52, 54, 55, 56, 58

fp_tabs, 51, 52, 54, 55, 55, 58

fp_tabs(), 54, 55, 108

fp_text, 51, 52, 54–56, 56, 58, 59, 101, 107, 110

fp_text(), 42, 43, 48, 100, 103, 105, 106

fp_text_lite (fp_text), 56

fp_text_lite(), 103, 105, 106


fpar(), 9, 55, 98, 104, 105

ftext, 47, 58, 58, 60, 101–110

ftext(), 48, 109

grep, 32

grepl, 31

gsub, 31

headers_replace_all_text

(body_replace_all_text), 31

headers_replace_img_at_bkm

(body_replace_text_at_bkm), 33

headers_replace_text_at_bkm

(body_replace_text_at_bkm), 33

headers_replace_text_at_bkm

(body_replace_text_at_bkm), 33