Package ‘officer’

July 21, 2021

Type Package
Title Manipulation of Microsoft Word and PowerPoint Documents
Version 0.3.19
Description Access and manipulate 'Microsoft Word' and 'Microsoft PowerPoint' documents from R. The package focuses on tabular and graphical reporting from R; it also provides two functions that let users get document content into data objects. A set of functions lets add and remove images, tables and paragraphs of text in new or existing documents. The package does not require any installation of Microsoft products to be able to write Microsoft files.
License GPL-3
Imports R6, grDevices, stats, graphics, utils, zip (>= 2.1.0), uuid (>= 0.1-4), xml2 (>= 1.1.0)
Encoding UTF-8
BugReports https://github.com/davidgohel/officer/issues
RoxygenNote 7.1.1
Suggests testthat, devEMF, ggplot2, rmarkdown, base64enc, knitr, rsvg
VignetteBuilder knitr
NeedsCompilation no
Author David Gohel [aut, cre], Frank Hangler [ctb] (function body_replace_all_text), Liz Sander [ctb] (several documentation fixes), Anton Victorson [ctb] (fixes xml structures), Jon Calder [ctb] (update vignettes), John Harrold [ctb] (function annotate_base), John Muschelli [ctb] (google doc compatibility)
Maintainer David Gohel <david.gohel@ardata.fr>
Repository CRAN
Date/Publication 2021-07-21 16:30:02 UTC
R topics documented:

add_sheet .................................................. 4
add_slide .................................................. 5
annotate_base ........................................... 5
block_caption ............................................. 6
block_list .................................................. 7
block_pour_docx ......................................... 8
block_section ............................................. 9
block_table ................................................ 10
block_toc .................................................. 11
body_add ................................................... 12
body_add_blocks ......................................... 15
body_add_break ......................................... 16
body_add_caption ........................................ 16
body_add_docx ............................................ 17
body_add_fpar ............................................ 18
body_add_gg .............................................. 19
body_add_img ............................................ 20
body_add_par ............................................ 21
body_add_plot ........................................... 22
body_add_table .......................................... 23
body_add_toc ............................................. 24
body_bookmark .......................................... 25
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 ......................... 30
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 ............................................. 37
docx_bookmarks ......................................... 39
docx_dim .................................................. 40
docx_show_chunk ....................................... 40
docx_summary ........................................... 41
doc_properties ........................................... 41
empty_content .......................................... 42
external_img ............................................ 43
fpar ....................................................... 44
fp_border .................................................. 45
fp_cell ..................................................... 46
fp_par ...................................................... 48
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fp_text</strong></td>
</tr>
<tr>
<td><strong>ftext</strong></td>
</tr>
<tr>
<td><strong>hyperlink_ftext</strong></td>
</tr>
<tr>
<td><strong>layout_properties</strong></td>
</tr>
<tr>
<td><strong>layout_summary</strong></td>
</tr>
<tr>
<td><strong>length.docx</strong></td>
</tr>
<tr>
<td><strong>length.rpptx</strong></td>
</tr>
<tr>
<td><strong>media_extract</strong></td>
</tr>
<tr>
<td><strong>move_slide</strong></td>
</tr>
<tr>
<td><strong>officer</strong></td>
</tr>
<tr>
<td><strong>officer-defunct</strong></td>
</tr>
<tr>
<td><strong>on_slide</strong></td>
</tr>
<tr>
<td><strong>page_mar</strong></td>
</tr>
<tr>
<td><strong>page_size</strong></td>
</tr>
<tr>
<td><strong>ph_add_fpar</strong></td>
</tr>
<tr>
<td><strong>ph_add_par</strong></td>
</tr>
<tr>
<td><strong>ph_add_text</strong></td>
</tr>
<tr>
<td><strong>ph_hyperlink</strong></td>
</tr>
<tr>
<td><strong>ph_location</strong></td>
</tr>
<tr>
<td><strong>ph_location_fullsize</strong></td>
</tr>
<tr>
<td><strong>ph_location_label</strong></td>
</tr>
<tr>
<td><strong>ph_location_left</strong></td>
</tr>
<tr>
<td><strong>ph_location_right</strong></td>
</tr>
<tr>
<td><strong>ph_location_template</strong></td>
</tr>
<tr>
<td><strong>ph_location_type</strong></td>
</tr>
<tr>
<td><strong>ph_remove</strong></td>
</tr>
<tr>
<td><strong>ph_slidelink</strong></td>
</tr>
<tr>
<td><strong>ph_with</strong></td>
</tr>
<tr>
<td><strong>plot_instr</strong></td>
</tr>
<tr>
<td><strong>plot_layout_properties</strong></td>
</tr>
<tr>
<td><strong>pptx_summary</strong></td>
</tr>
<tr>
<td><strong>print.rpptx</strong></td>
</tr>
<tr>
<td><strong>prop_section</strong></td>
</tr>
<tr>
<td><strong>prop_table</strong></td>
</tr>
<tr>
<td><strong>read_docx</strong></td>
</tr>
<tr>
<td><strong>read_pptx</strong></td>
</tr>
<tr>
<td><strong>read_xlsx</strong></td>
</tr>
<tr>
<td><strong>remove_slide</strong></td>
</tr>
<tr>
<td><strong>run_autonum</strong></td>
</tr>
<tr>
<td><strong>run_bookmark</strong></td>
</tr>
<tr>
<td><strong>run_columnbreak</strong></td>
</tr>
<tr>
<td><strong>run_linebreak</strong></td>
</tr>
<tr>
<td><strong>run_pagebreak</strong></td>
</tr>
<tr>
<td><strong>run_reference</strong></td>
</tr>
<tr>
<td><strong>run_word_field</strong></td>
</tr>
<tr>
<td><strong>sanitize_images</strong></td>
</tr>
<tr>
<td><strong>section_columns</strong></td>
</tr>
<tr>
<td><strong>set_doc_properties</strong></td>
</tr>
</tbody>
</table>
add_sheet

Description

add a sheet into an xlsx worksheet

Usage

```r
add_sheet(x, label)
```

Arguments

- `x`: rxlsx object
- `label`: sheet label

Examples

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

add a slide

Description
add a slide into a pptx presentation

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

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

See Also
- print.rpptx()
- read_pptx()
- plot_layout_properties()
- ph_with()
- layout_summary()

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

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

annotate_base
PowerPoint placeholder parameters annotation

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

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

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

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

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

Value

rpptx object of the annotated PowerPoint file

See Also

Other functions for reading presentation informations: color_scheme(), 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_anotate.pptx'
# annotate_base(path = 'mydoc.pptx', output_file='mydoc_anotate.pptx')

block_caption  Caption block

Description

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

Usage

block_caption(label, style, autonum = NULL)

Arguments

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

See Also

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

Examples

```r
library(officer)

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

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

```r
block_list(...)```

Arguments

```r
...```

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

```r
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

```r
# block list ------

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

## This can be only be used in a MS word output as pptx does 
## not support paragraphs made of text and images. 
## (actually it can be used but image will not appear in the 
## pptx output)
value <- block_list(
  fpar(ftext("hello world", fpt_blue_bold)),
  fpar(ftext("hello", fpt_blue_bold), " ",
       ftext("world", fpt_red_italic)),
  fpar( 
      ftext("hello world", fpt_red_italic),
      external_img(
          src = img.file, height = 1.06, width = 1.39)))

value

doc <- read_docx()
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**  

*Pour external Word document in the current document*

---

**Description**

Pour the content of a docx file in the resulting docx generated by the main 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))
print(doc_1, target = target)
```

---

**block_section**

*New Word section*

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`
block_table

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()
Description

Create a representation of a table of content.

Usage

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

Arguments

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

See Also

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

Examples

```
block_toc(level = 2)
block_toc(style = "Table Caption")
```
Description

This function adds objects into a Word document. Values are added as new paragraphs or tables.

This function is experimental and will replace the `body_add_*` functions later. For now it is only to be used for successive additions and cannot be used in conjunction with the `body_add_*` functions.

Usage

```r
body_add(x, value, ...)

## S3 method for class 'character'
body_add(x, value, style = NULL, ...)

## S3 method for class 'numeric'
body_add(x, value, style = NULL, format_fun = formatC, ...)

## S3 method for class 'factor'
body_add(x, value, style = NULL, format_fun = as.character, ...)

## S3 method for class 'fpar'
body_add(x, value, style = NULL, ...)

## S3 method for class 'data.frame'
body_add(
  x,
  value,
  style = NULL,
  header = TRUE,
  tcf = table_conditional_formatting(),
  alignment = NULL,
  ...
)

## S3 method for class 'block_caption'
body_add(x, value, ...)

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

## S3 method for class 'block_toc'
body_add(x, value, ...)

## S3 method for class 'external_img'
```
body_add(x, value, style = "Normal", ...)

## S3 method for class 'run_pagebreak'
body_add(x, value, style = NULL, ...)

## S3 method for class 'run_columnbreak'
body_add(x, value, style = NULL, ...)

## S3 method for class 'gg'
body_add(x, value, width = 6, height = 5, res = 300, style = "Normal", ...)

## S3 method for class 'plot_instr'
body_add(x, value, width = 6, height = 5, res = 300, style = "Normal", ...)

## S3 method for class 'block_pour_docx'
body_add(x, value, ...)

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

Arguments

x an rdocx object

value object to add in the document. Supported objects are vectors, data.frame, graphics, block of formatted paragraphs, unordered list of formatted paragraphs, pretty tables with package flextable, 'Microsoft' charts with package mschart.

... 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 method signatures to see what arguments can be used.

style paragraph style name. These names are available with function styles_info and are the names of the Word styles defined in the base document (see argument path from read_docx).

format_fun a function to be used to format values.

header display header if TRUE

tcf conditional formatting settings defined by table_conditional_formatting()

alignment columns alignment, argument length must match with columns length, values must be "l" (left), "r" (right) or "c" (center).

width height in inches

height height in inches

res resolution of the png image in ppi

Methods (by class)

- character: add a character vector.
- numeric: add a numeric vector.
body_add

- factor: add a factor vector.
- fpar: add a fpar object. These objects enable the creation of formatted paragraphs made of formatted chunks of text.
- data.frame: add a data.frame object with block_table().
- block_caption: add a block_caption object. These objects enable the creation of set of formatted paragraphs made of formatted chunks of text.
- block_list: add a block_list object.
- block_toc: add a table of content (a block_toc object).
- external_img: add an image (a external_img object).
- run_pagebreak: add a run_pagebreak object.
- run_columnbreak: add a run_columnbreak object.
- gg: add a ggplot object.
- plot_instr: add a base plot with a plot_instr object.
- block_pour_docx: pour content of an external docx file with with a block_pour_docx object
- block_section: ends a section with a block_section object

Illustrations

Examples

doc_1 <- read_docx()
doc_1 <- body_add(doc_1, "Table of content", style = "heading 1")
doc_1 <- body_add(doc_1, block_toc())
doc_1 <- body_add(doc_1, run_pagebreak())
doc_1 <- body_add(doc_1, "A title", style = "heading 1")
doc_1 <- body_add(doc_1, head(iris), style = "table_template")
doc_1 <- body_add(doc_1, "Another title", style = "heading 1")
doc_1 <- body_add(doc_1, letters, style = "Normal")
doc_1 <- body_add(doc_1,
  block_section(prop_section(type = "continuous")))
)
doc_1 <- body_add(doc_1, plot_instr(code = barplot(1:5, col = 2:6)))
doc_1 <- body_add(doc_1,
  block_section(prop_section(page_size = page_size(orient = "landscape"))))
)
print(doc_1, target = tempfile(fileext = ".docx"))
# print(doc_1, target = "test.docx")
body_add_blocks

Description

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

Usage

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

Arguments

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

See Also

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

Examples

library(officer)

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

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

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

**Description**

add a page break into an rdocx object

**Usage**

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

**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**

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

body_add_caption

**Description**

add a Word caption into an rdocx object.

**Usage**

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

**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".
body_add_docx

See Also

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

Examples

```r
doc <- read_docx()

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

---

**body_add_docx**

*insert an external docx*

**Description**

add content of a docx into an rdocx object.

**Usage**

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

**Arguments**

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

**Note**

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

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

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

Examples

```r
file1 <- tempfile(fileext = ".docx")
file2 <- tempfile(fileext = ".docx")
file3 <- tempfile(fileext = ".docx")
x <- read_docx()
x <- body_add_par(x, "hello world 1", style = "Normal")
print(x, target = file1)

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

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

---

**body_add_fpar**  
**add fpar**

**Description**

add an fpar (a formatted paragraph) into an rdocx object

**Usage**

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

**Arguments**

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

fpar

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

Examples

```r
bold_face <- shortcuts$fp_bold(font.size = 30)
bold_redface <- update(bold_face, color = "red")
fpars <- 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, fpars)
print(doc, target = tempfile(fileext = ".docx"))
```

```r
# 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"))
```
**body_add_img**

```r
height  height in inches
res     resolution of the png image in ppi
style   paragraph style
...     Arguments to be passed to png function.
```

**See Also**

Other functions for adding content: `body_add_blocks()`, `body_add_break()`, `body_add_caption()`, `body_add_docx()`, `body_add_fpar()`, `body_add_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"))
}
```

---

**Description**

add an image into an rdocx object.

**Usage**

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

**Arguments**

- **x**  an rdocx object
- **src** image filename, the basename of the file must not contain any blank.
- **style** paragraph style
- **width** height in inches
- **height** height in inches
- **pos** where to add the new element relative to the cursor, one of "after", "before", "on".
See Also

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

Examples

doc <- read_docx()

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

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

---

**body_add_par** add paragraph of text

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

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**

**Description**

add a plot as a png image into an rdocx object

**Usage**

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

**Arguments**

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

**See Also**

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

**Examples**

```r
doc <- read_docx()

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

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

add a table into an rdocx object

Usage

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

Arguments

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

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

Examples

doc <- read_docx()
doc <- body_add_toc(doc)

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

---

**Description**

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

**Usage**

```
body_bookmark(x, id)
```

**Arguments**

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

**Examples**

```
# 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**

---

**Description**

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

**Usage**

```
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_landscape()`, `body_end_section_columns()`, `body_end_section_continuous()`, `body_end_section_portrait()`, `body_set_default_section()`

Examples

```r
library(officer)

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

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

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

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

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

---

**body_end_section_columns**

*add multi columns section*

Description

A section with multiple columns is added to the document.

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.
body_end_section_columns_landscape

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

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

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

body_end_section_columns_landscape

*add multi columns section within landscape orientation*

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

Usage

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

Arguments

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

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

Examples

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

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

body_end_section_continuous

add continuous section

Description

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

Usage

body_end_section_continuous(x)

Arguments

x

an rdocx object

See Also

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

Examples

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)
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_landscape(), body_end_section_columns(), 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)
```

**Arguments**

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

**See Also**

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

**Examples**

```r
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_end_section_portrait(doc_1)
doc_1 <- body_add_par(doc_1, value = cstr1, style = "Normal")
print(doc_1, target = tempfile(fileext = ".docx"))
```

**body_remove**

*remove an element*

**Description**

remove element pointed by cursor from a Word document

**Usage**

```
body_remove(x)
```
Arguments

x  an rdocx object

Examples

library(officer)

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

str2 <- "Drop that text"

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

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

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

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

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

body_replace_all_text  

Replace text anywhere in the document, or at a cursor

Description

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

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

Chunking of text

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

**Usage**

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

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

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

**Arguments**

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

**header_replace_all_text**

Replacements will be performed in each header of all sections.

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

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

body_replace_text_at_bkm(x, bookmark, value)

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

Arguments

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

Examples

doc <- read_docx()
doc <- body_add_par(doc, "centered text", style = "centered")
doc <- slip_in_text(doc, ". How are you", style = "strong")
doc <- bodyBookmark(doc, "text_to_replace")
doc <- body_replace_text_at_bkm(doc, "text_to_replace", "not left aligned")

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

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

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

body_set_default_section

Define Default Section

Description

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

Usage

body_set_default_section(x, value)
**change_styles**

**Arguments**

- **x**: an rdocx object
- **value**: a prop_section object

**Illustrations**

**See Also**

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

**Examples**

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

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

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

**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, "", style = "Normal")
doc_1 <- slip_in_text(doc_1, "Message is: ",
    style = "Default Paragraph Font"
)
doc_1 <- body_add_par(doc_1, "Hello ", style = "Normal")
doc_1 <- slip_in_text(doc_1, "world", style = "Default Paragraph Font")
doc_1 <- slip_in_text(doc_1, " with a link",
    style = "strong",
    pos = "after", hyperlink = "https://davidgohel.github.io/officer/
"
)
doc_1 <- body_add_par(doc_1, "Another title", style = "heading 2")
doc_1 <- body_add_par(doc_1, "Hello world!", style = "Normal")

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

# now we can illustrate how
# to change styles with 'change_styles'

doc_2 <- read_docx(path = file)

mapstyles <- list(
  "centered" = c("Normal", "heading 2"),
  "strong" = "Default Paragraph Font"
)
doc_2 <- change_styles(doc_2, mapstyles = mapstyles)

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

color_scheme

Description

get master layout color scheme into a data.frame.

Usage

color_scheme(x)

Arguments

x

an rpptx object

See Also

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

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

---

**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_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_begin

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

cursor_forward

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

cursor_backward

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

Examples

```r
library(officer)

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

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

# Let add text at the beginning of the
# paragraph containing text "paragraph 4"
doc <- cursor_reach(doc, keyword = "paragraph 4")
doc <- slip_in_text(doc, "This is ", pos = "before", style = "Default Paragraph Font")

# move the cursor forward and end a section
doc <- cursor_forward(doc)
doc <- body_add_par(doc, "The section stop here", style = "Normal")
doc <- body_end_section_landscape(doc)

# move the cursor at the end of the document
doc <- cursor_end(doc)
doc <- body_add_par(doc, "The document ends now", style = "Normal")

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

# cursor_bookmark ----

doc <- read_docx()
doc <- body_add_par(doc, "centered text", style = "centered")
doc <- body_bookmark(doc, "text_to_replace")
doc <- body_add_par(doc, "A title", style = "heading 1")
```
### docx_bookmarks

`docx_bookmarks(x)`

**Arguments**

- `x` - an `rdocx` object

**Description**

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

**Usage**

```r
docx_bookmarks(x)
```

**See Also**

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

**Examples**

```r
library(officer)

doc_1 <- read_docx()
doc_1 <- body_add_par(doc_1, "centered text", style = "centered")
doc_1 <- body_bookmark(doc_1, "text_to_replace_1")
doc_1 <- body_add_par(doc_1, "centered text", style = "centered")
doc_1 <- body_bookmark(doc_1, "text_to_replace_2")

docx_bookmarks(doc_1)
docx_bookmarks(read_docx())
```
**docx_dim**

**Word page layout**

**Description**

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

**Usage**

docx_dim(x)

**Arguments**

x an rdocx object

**See Also**

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

**Examples**

docx_dim(read_docx())

---

**docx_show_chunk**

**Show underlying text tag structure**

**Description**

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

**Usage**

docx_show_chunk(x)

**Arguments**

x a docx device

**See Also**

`body_replace_all_text`
Examples

```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'
```

---

**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**

```r
docx_summary(x)
```

**Arguments**

- `x` an rdocx object

**Note**

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

**Examples**

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

---

**doc_properties**

*read document properties*

**Description**

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

**Usage**

```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()`

Examples

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

---

empty_content  
create empty blocks

Description

an empty object to include as an empty placeholder shape in a presentation. This comes in handy when presentation are updated through R, but a user still wants to write the takeaway statements in PowerPoint.

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")
print(doc, target = fileout)
```
**external_img**

---

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

**Arguments**

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

**usage**

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

**See Also**

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

**Examples**

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

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

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

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

---

fpar

**Concatenate formatted text as a paragraph**

### Description

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

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

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

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

### Usage

```r
fpar(..., fp_p = fp_par(), fp_t = fp_text(), values = NULL)
```

### Arguments

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

### See Also

`block_list()`, `body_add_fpar()`, `ph_with()`

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

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

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

bold_face <- shortcuts$fp_bold(font.size = 12)
bold_redface <- update(bold_face, color = "red")
fpar_1 <- fpar(
  "Hello World, ",
  ftext("how ", prop = bold_redface ),
  external_img(src = img.file, height = 1.06/2, width = 1.39/2),
  ftext(" you?", prop = bold_face )
)
fpar_1
```

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

fp_border

### border properties object

Description

create a border properties object.

Usage

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

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

Arguments

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

See Also

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

```
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

```
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 = "lrtb"
)
```

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

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

## S3 method for class 'fp_cell'
update(
  object,
  border,
  border-bottom,
  border.left,
  border.top,
  ...)
```
Args

- `border` shortcut for all 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.
- `background.color` cell background color - a single character value specifying a valid color (e.g. "#000000" or "black").
- `text.direction` cell text rotation - a single character value, expected value is one of "lrtb", "tbrl", "btlr".
- `x, object` fp_cell object
- `type` output type - one of 'wml', 'pml', 'html'.
- `...` further arguments - not used

See Also

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

Examples

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

Create a \texttt{fp_par} object that describes paragraph formatting properties.

Usage

\begin{verbatim}
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
)
\end{verbatim}

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

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

- **text.align**
  - text alignment - a single character value, expected value is one of 'left', 'right', 'center', 'justify'.

- **padding**
  - paragraph paddings - 0 or positive integer value. Argument padding overwrites arguments padding.bottom, padding.top, padding.left, padding.right.

- **line_spacing**
  - line spacing. 1 is single line spacing, 2 is double line spacing.

- **border**
  - shortcut for all borders.

- **padding.bottom**, **padding.top**, **padding.left**, **padding.right**
  - paragraph paddings - 0 or positive integer value.

- **border.bottom**, **border.left**, **border.top**, **border.right**
  - **fp_border** for borders. overwrite other border properties.

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

- **keep_with_next**
  - a scalar logical. Specifies that the paragraph (or at least part of it) should be rendered on the same page as the next paragraph when possible.

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

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

Value

- a **fp_par** object

See Also

- **fp**

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

Examples

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

---

**fp_text**

**Text formatting properties**

Description

Create a **fp_text** object that describes text formatting properties.
Usage

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

## 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.

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").

x fp_text object
type output type - one of 'wml', 'pml', 'html'.
... further arguments - not used
object fp_text object to modify
format format type, wml for MS word, pml for MS PowerPoint and html.

Value

a fp_text object

See Also

ftext, fpar

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

Examples

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

ftext formatted chunk of text

Description

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

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

Arguments

text text value, a single character value

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

Usage

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

See Also

`fp_text`

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

Examples

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

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

```r
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)
```
layout_properties

Arguments

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

usage

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

See Also

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

Examples

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

layout_properties    slide layout properties

Description

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

Usage

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

Arguments

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

See Also

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

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

---

**layout_summary**  
**presentation layouts summary**

**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()`, `layout_properties()`, `length.rpptx()`, `plot_layout_properties()`, `slide_size()`, `slide_summary()`

**Examples**

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

---

**length.rdocx**  
**number of blocks inside an rdocx object**

**Description**

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

**Usage**

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

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

# how many elements are there in a new document produced
# with the default template.
length( read_docx() )

length.rptx number of slides

Description

Function length will return the number of slides.

Usage

## S3 method for class 'rptx'
length(x)

Arguments

x an rptx object

See Also

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

Examples

my_pres <- read_pptx()
my_pres <- add_slide(my_pres)
my_pres <- add_slide(my_pres)
length(my_pres)
**media_extract**  
*Extract media from a document object*

**Description**
Extract files from an rdocx or rpptx object.

**Usage**
```
media_extract(x, path, target)
```

**Arguments**
- `x`: an rpptx object or an rdocx object
- `path`: media path, should be a relative path
- `target`: target file

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

**move_slide**  
*move a slide*

**Description**
move a slide in a pptx presentation

**Usage**
```
move_slide(x, index, to)
```

**Arguments**
- `x`: an rpptx object
- `index`: slide index, default to current slide position.
- `to`: new slide index.
Note
cursor is set on the last slide.

See Also

read_pptx()
Other functions slide manipulation: add_slide(), on_slide(), remove_slide()

Examples

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)

Description

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

Details

Examples of manipulations are:

- read Word and PowerPoint files into data objects
- add/edit/remove image, table and text content from documents and slides
- write updated content back to Word and PowerPoint files

To learn more about officer, start with the vignettes: browseVignettes(package = "officer")

Author(s)

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

- 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 <jonnicalder@gmail.com> (update vignettes) [contributor]
- John Harrold <john.m.harrold@gmail.com> (function annotate_base) [contributor]
- John Muschelli <muschellij2@gmail.com> (google doc compatibility) [contributor]
See Also

https://davidgohel.github.io/officer/

---

**officer-defunct**  Defunct Functions in Package officer

**Description**

Defunct Functions in Package officer

**Usage**

ph_with_gg_at(...)

ph_with_table_at(...)

ph_with_text(...)

**Arguments**

... unused arguments

**Details**

ph_with() is replaced by ph_with.gg.

ph_with_table_at() is replaced by ph_with.data.frame.

ph_with_text() is replaced by ph_with.character.

---

**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()`

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

page_size()
Description

append fpar (a formatted paragraph) in a placeholder The function let you add a new formatted paragraph (fpar) to an existing content in an existing shape, existing paragraphs will be preserved.

Usage

```r
ph_add_fpar(
  x,
  value,
  type = "body",
  id = 1,
  id_chr = NULL,
  ph_label = NULL,
  level = 1,
  par_default = TRUE
)
```

Arguments

- **x** an rpptx object
- **value** fpar 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`.
- **level** paragraph level
- **par_default** specify if the default paragraph formatting should be used.

Usage

If your goal is to add formatted text in a new shape, use `ph_with` with a `block_list` instead of this function.

Note

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

fpar

Examples

```r
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_pptx()
doc <- add_slide(doc, layout = "Title and Content", master = "Office Theme")
doc <- ph_with(doc, ",", location = ph_location(bg = "wheat", newlabel = "myph"))
doc <- ph_add_fpar(doc, value = fpar_, ph_label = "myph", level = 2)

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

Description

append a new empty paragraph in a placeholder. The function let you add a new empty paragraph to an existing content in an existing shape, existing paragraphs will be preserved.

Usage

```r
ph_add_par(x, type = "body", id = 1, id_chr = NULL, level = 1, ph_label = 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`.
- **id_chr**: deprecated.
- **level**: paragraph level
- **ph_label**: label associated to the placeholder. Use column `ph_label` of result returned by `slide_summary`.

Usage

If your goal is to add formatted text in a new shape, use `ph_with` with a `block_list` instead of this function.
Note

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

Examples

```r
fileout <- tempfile(fileext = ".pptx")
default_text <- fp_text(font.size = 0, bold = TRUE, color = "red")

doc <- read_pptx()
doc <- add_slide(doc, layout = "Title and Content", master = "Office Theme")
doc <- ph_with(doc, "A text", location = ph_location_type(type = "body"))
doc <- ph_add_par(doc, level = 2)
doc <- ph_add_text(doc, str = "and another," , style = default_text )
doc <- ph_add_par(doc, level = 3)
doc <- ph_add_text(doc, str = "and another!",
                   style = update(default_text, color = "blue"))

print(doc, target = fileout)
```

---

**Description**

Append text in a placeholder. The function let you add text to an existing content in an existing shape, existing text will be preserved.

**Usage**

```r
ph_add_text(
  x,
  str,
  type = "body",
  id = 1,
  id_chr = NULL,
  ph_label = NULL,
  style = fp_text(font.size = 0),
  pos = "after",
  href = NULL,
  slide_index = NULL
)
```

**Arguments**

- `x` an rpptx object
- `str` text to add
type
id
id_chr
ph_label
style
pos
href
slide_index

**Usage**

If your goal is to add formatted text in a new shape, use `ph_with` with a `block_list` instead of this function.

**Note**

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

**Examples**

```r
fileout <- tempfile(fileext = ".pptx")
my_pres <- read_pptx()
my_pres <- add_slide(my_pres)
my_pres <- ph_with(my_pres, ",",
    location = ph_location_type(type = "body")))

small_red <- fp_text(color = "red", font.size = 14)

my_pres <- ph_add_text(my_pres, str = "A small red text.",
    style = small_red)
my_pres <- ph_add_par(my_pres, level = 2)
my_pres <- ph_add_text(my_pres, str = "Level 2")
print(my_pres, target = fileout)

# another example ----
fileout <- tempfile(fileext = ".pptx")

doc <- read_pptx()
doc <- add_slide(doc)
doc <- ph_with(doc, "Un titre 2",
    location = ph_location_type(type = "title"))
doc <- ph_with(doc, ",",
```

**Description**

add hyperlink to a placeholder in the current slide.

**Usage**

```r
doc <- ph_hyperlink(x, type = "body", id = 1, id_chr = NULL, ph_label = NULL, href)
```

**Arguments**

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

**See Also**

- `ph_with`

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

**Examples**

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

print(doc, target = fileout )
```
**ph_location**  
create a location for a placeholder

---

**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,
  ...
)
```

**Arguments**

- `left`, `top`, `width`, `height`  
  place holder coordinates in inches.
- `newlabel`  
  a label for the placeholder. See section details.
- `bg`  
  background color
- `rotation`  
  rotation angle
- `...`  
  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

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") )

---

**ph_location_fullsize**  
*location of a full size element*

Description

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

Usage

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

Arguments

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

See Also

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

Examples

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

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

Usage

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

Arguments

- `ph_label`: placeholder label of the used layout. It can be read in PowerPoint or with function `layout_properties()` in column `ph_label`.
- `newlabel`: a label to associate with the placeholder.
- `...`: unused arguments

Details

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

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

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

See Also

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

Examples

```r
# ph_location_label demo ----
doc <- read_pptx()
doc <- add_slide(doc, layout = "Title and Content")

# all ph_label can be read here
```
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**

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

**Arguments**

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

**See Also**

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

**Examples**

```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_right**  
*location of a right body element*

**Description**

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

**Usage**

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

**Arguments**

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

**See Also**

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

**Examples**

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

**ph_location_template**  
*create a location for a placeholder based on a template*

**Description**

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

```r
```
Usage

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

Arguments

- `left, top, width, height` place holder coordinates in inches.
- `newlabel` a label for the placeholder. See section details.
- `type` placeholder type to look for in the slide layout, one of `‘body’, ‘title’, ‘ctrTitle’, ‘subTitle’, ‘dt’, ‘fr’, ‘sldNum’`. It will be used as a template placeholder.
- `id` index of the placeholder template. If two body 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:

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

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

See Also

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

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

**Details**

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

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

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

**See Also**

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

**Examples**

```r
# ph_location_type demo ----

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

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

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

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

remove a shape in a slide

**Usage**

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

**Arguments**

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

**See Also**

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

**Examples**

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

doc <- on_slide(doc, index = 1)
```
```r
doc <- ph_remove(x = doc, type = "title")

doc <- on_slide(doc, index = 2)
doc <- ph_remove(x = doc, type = "body", id = 2)

doc <- on_slide(doc, index = 3)
doc <- ph_remove(x = doc, type = "body", id = 1)

print(doc, target = fileout)
```

---

**ph_slidelink**  
*slide link to a placeholder*

**Description**

add slide link to a placeholder in the current slide.

**Usage**

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

**Arguments**

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

**See Also**

`ph_with`

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

```r
default <- 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 = default)
```

---

**ph_with**

*add objects into a new shape on the current slide*

Description

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

Usage

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

## S3 method for class 'character'
```r
ph_with(x, value, location, ...)
```

## S3 method for class 'numeric'
```r
ph_with(x, value, location, format_fun = format, ...)
```

## S3 method for class 'factor'
```r
ph_with(x, value, location, ...)
```

## S3 method for class 'logical'
```r
ph_with(x, value, location, format_fun = format, ...)
```

## S3 method for class 'block_list'
```r
ph_with(x, value, location, level_list = integer(0), ...)
```

## S3 method for class 'unordered_list'
```r
ph_with(x, value, location, ...)
```

## S3 method for class 'data.frame'
```r
ph_with(
  x,
```
**Arguments**

- **x**: an rpptx object
- **value**: object to add as a new shape. Supported objects are vectors, data.frame, graphics, block of formatted paragraphs, unordered list of formatted paragraphs, pretty tables with package flextable, editable graphics with package rvg, 'Microsoft' charts with package mschart.
- **location**: a placeholder location object. It will be used to specify the location of the new shape. This location can be defined with a call to one of the ph_location functions. See section "see also".
- **...**: further arguments passed to or from other methods. When adding a ggplot object or plot_instr, these arguments will be used by png function.
- **format_fun**: format function for non character vectors
- **level_list**: The list of levels for hierarchy structure as integer values. If used the object is formatted as an unordered list. If 1 and 2, item 1 level will be 1, item 2 level will be 2.
- **header**: display header if TRUE
- **tcf**: conditional formatting settings defined by `table_conditional_formatting()`
- **alignment**: alignment for each columns, 'l' for left, 'r' for right and 'c' for center. Default to NULL.
res       resolution of the png image in ppi
alt_text  Alt-text for screen-readers
use_loc_size  if set to FALSE, external_img width and height will be used.

Methods (by class)

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

Illustrations

See Also

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

Examples

# this name will be used to print the file
# change it to "youfile.pptx" to write the pptx
# file in your working directory.
fileout <- tempfile(fileext = ".pptx")
ph_with

```r
doc_1 <- read_pptx()
sz <- slide_size(doc_1)
# add text and a table ----
doc_1 <- add_slide(doc_1, layout = "Two Content", master = "Office Theme")
doc_1 <- ph_with(x = doc_1, value = c("Table cars"),
    location = ph_location_type(type = "title")
)
doc_1 <- ph_with(x = doc_1, value = names(cars),
    location = ph_location_left()
)
doc_1 <- ph_with(x = doc_1, value = cars,
    location = ph_location_right()
)

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

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

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

svg_file <- file.path(R.home(component = "doc"), "html/Rlogo.svg")
if( require("rsvg") ){  
doc_1 <- ph_with(x = doc_1, value = "External images",
                   location = ph_location_type(type = "title")
    }

svg_file <- file.path(R.home(component = "doc"), "html/Rlogo.svg")
if( require("rsvg") ){  
doc_1 <- ph_with(x = doc_1, value = "External images",
                   location = ph_location_type(type = "title")
    }
```

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()`.

**Description**

Wrap plot instructions for png plotting in Powerpoint or Word
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

Plot slide layout properties

Description

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

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

Arguments

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

See Also

Other functions for reading presentation informations: annotate_base(), color_scheme(), 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 get PowerPoint content in a data.frame

Description

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

Usage

pptx_summary(x)

Arguments

x an rpptx object

Examples

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

write a 'PowerPoint' file.

Description
write a 'PowerPoint' file.

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)

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.

Usage

prop_section(
  page_size = NULL,
  page_margins = NULL,
  type = NULL,
  section_columns = NULL
)
Arguments

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

Illustrations

Note

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

See Also

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

Examples

```r
library(officer)

landscape_one_column <- block_section(prop_section(
  page_size = page_size(orient = "landscape"), type = "continuous"
))

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

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

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

# there stops section with landscape_two_columns

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

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

---

prop_table  

*Table properties*

**Description**

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

**Usage**

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

**Arguments**

- `style`  
  table style to be used to format table

- `layout`  
  layout defined by `table_layout()`.

- `width`  
  table width in the document defined by `table_width()`

- `stylenames`  
  columns styles defined by `table_stylenames()`

- `colwidths`  
  column widths defined by `table_colwidths()`

- `tcf`  
  conditional formatting settings defined by `table_conditional_formatting()`

- `align`  
  table alignment (one of left, center or right)

**See Also**

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

**Examples**

```r
prop_table()
to_wml(prop_table())
```
read_docx | Create a 'Word' document object

Description

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

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

Usage

read_docx(path = NULL)

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

Arguments

path | path to the docx file to use as base document.

x | an rdocx object

target | path to the docx file to write

... | unused

Value

an object of class rdocx.

Methods (by generic)

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

styles

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

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

See body_add_* functions to add content.

Illustrations

See Also

body_add_par, body_add_plot, body_add_table
Examples

```r
library(officer)

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

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

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

read_pptx

open a connexion to a 'PowerPoint' file

Description

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

Usage

`read_pptx(path = NULL)`

Arguments

path

path to the pptx file to use as base document.

master layouts and slide layouts

`read_pptx()` uses a PowerPoint file as the initial document. This is the original PowerPoint document where all slide layouts, placeholders for shapes and styles come from. Major points to be aware of are:
Slide layouts are relative to a master layout. A document can contain one or more master layouts; a master layout can contain one or more slide layouts.

A slide layout inherits design properties from its master layout but some properties can be overwritten.

Designs and formatting properties of layouts and shapes (placeholders in a layout) are defined within the initial document. There is no R function to modify these values - they must be defined in the initial document.

See Also

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

Examples

read_pptx()

---

read_xlsx open a connexion to an 'Excel' file

Description

read and import an xlsx file as an R object representing the document. This function is experimental.

Usage

read_xlsx(path = NULL)

## S3 method for class 'rxlsx'
length(x)

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

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

read_xlsx()

x <- read_xlsx()

print(x, target = tempfile(fileext = ".xlsx"))
remove_slide

Description
remove a slide from a pptx presentation

Usage
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()

Examples
my_pres <- read_pptx()
my_pres <- add_slide(my_pres)
my_pres <- remove_slide(my_pres)

run_autonum

auto number

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

```r
run_autonum(
  seq_id = "table",
  pre_label = "Table ",
  post_label = " : ",
  bkm = NULL,
  bkm_all = FALSE,
  prop = NULL,
  start_at = NULL
)
```

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.

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_linebreak()`, `run_pagebreak()`, `run_reference()`, `run_word_field()`

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

Examples

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

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_linebreak(), run_pagebreak(), run_reference(), run_word_field()

Examples

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

run_columnbreak

column break

Description

Create a representation of a column break

Usage

run_columnbreak()
run_linebreak

usage

You can use this function in conjunction with `fpars` 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_pagebreak()`, `run_reference()`, `run_word_field()`

Examples

```r
fp_t <- fp_text(font.size = 12, bold = TRUE)
an_fpar <- fpars("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"))
```
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_linebreak(), run_reference(), run_word_field()

Examples

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

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

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 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_linebreak(), run_pagebreak(), run_word_field()

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

Examples

run_reference('a_ref')

Description

Create a Word computed field.

Usage

run_word_field(field, prop = NULL, seqfield = field)

run_seqfield(field, prop = NULL, seqfield = field)

Arguments

field, seqfield
    computed field string (seqfield will be totally superseded by field in the future).

prop
    formatting text properties returned by fp_text.

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_linebreak()`, `run_pagebreak()`, `run_reference()`.

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

Examples

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

Description

The function will scan the media directory and delete images that are not used anymore. This function is to be used when images have been replaced many times.

Usage

```r
sanitize_images(x)
```

Arguments

- `x` rdocx or rpptx object

section_columns

Description

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

Usage

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

Arguments

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

See Also

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

set_doc_properties()

---

set_doc_properties  set document properties

Description

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

Usage

set_doc_properties(
  x,
  title = NULL,
  subject = NULL,
  creator = NULL,
  description = NULL,
  created = NULL
)

Arguments

x an rdocx or rpptx object

title, subject, creator, description
text fields

created a date object

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())
x <- doc_properties(x)
**Description**

set a particular sheet selected when workbook will be edited.

**Usage**

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

**Arguments**

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

**Examples**

```
my_ws <- read_xlsx()
my_pres <- add_sheet(my_ws, label = "new sheet")
my_pres <- sheet_select(my_ws, sheet = "new sheet")
print(my_ws, target = tempfile(fileext = ".xlsx") )
```

**shortcuts**

**shortcuts for formatting properties**

**Description**

Shortcuts for `fp_text`, `fp_par`, `fp_cell` and `fp_border`.

**Usage**

```
shortcuts
```

**Examples**

```
shortcuts$fp_bold()
shortcuts$fp_italic()
shortcuts$b_null()
```
slide_size

slides width and height

Description

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

Usage

slide_size(x)

Arguments

x

an rpptx object

See Also

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

Examples

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

slide_summary

get PowerPoint slide content in a data.frame

Description

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

Usage

slide_summary(x, index = NULL)

Arguments

x

an rpptx object

index

slide index
Note

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

See Also

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

Examples

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

```
slip_in_column_break  add a column break

Description

add a column break into a Word document. A column break is used to add a break in a multi columns section in a Word Document.

Usage

slip_in_column_break(x, pos = "before")

Arguments

x  an rdocx object
pos  where to add the new element relative to the cursor, "after" or "before".
slip_in_footnote  append a footnote

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

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

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

Examples
img.file <- file.path( R.home("doc"), "html", "logo.jpg" )
bl <- block_list(
  fpar(ftext("hello", shortcuts$fp_bold())),
  fpar(
    ftext("hello world", shortcuts$fp_bold()),
    external_img(src = img.file, height = 1.06, width = 1.39)
  )
)
x <- read_docx()
x <- body_add_par(x, "Hello ", style = "Normal")
x <- slip_in_text(x, "world", style = "strong")
x <- slip_in_footnote(x, style = "reference_id", blocks = bl)
print(x, target = tempfile(fileext = ".docx"))

slip_in_img  append an image

Description
append an image into a paragraph of an rdocx object

Usage
slip_in_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**: text style
- **width**: height in inches
- **height**: height in inches
- **pos**: where to add the new element relative to the cursor, "after" or "before".

Note

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

Examples

```r
tmp.img <- file.path( R.home("doc"), "html", "logo.jpg" )
x <- read_docx()
x <- body_add_par(x, "R logo: ", style = "Normal")
x <- slip_in_img(x, src = tmp.img, style = "strong", width = .3, height = .3)

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

Description

append seq field into a paragraph of an rdocx object. This feature is only available when document are edited with Word, when edited with Libre Office or another program, seq field will not be calculated and not displayed.

Usage

```r
slip_in_seqfield(x, str, style = NULL, pos = "after")
```

Arguments

- **x**: an rdocx object
- **str**: seq field value
- **style**: text style
- **pos**: where to add the new element relative to the cursor, "after" or "before".

Note

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

```r
x <- read_docx()
x <- body_add_par(x, "Time is: ", style = "Normal")
x <- slip_in_seqfield(x,
  str = "TIME \u005C@ "HH:mm:ss\u005C* MERGEFORMAT",
  style = 'strong')

x <- body_add_par(x, " - This is a figure title", style = "centered")
x <- slip_in_seqfield(x, str = "SEQ Figure \u005C* roman",
  style = 'Default Paragraph Font', pos = "before")
x <- slip_in_text(x, "Figure: ", style = "strong", pos = "before")

x <- body_add_par(x, " - This is another figure title", style = "centered")
x <- slip_in_seqfield(x, str = "SEQ Figure \u005C* roman",
  style = 'strong', pos = "before")
x <- slip_in_text(x, "Figure: ", style = "strong", pos = "before")
x <- body_add_par(x, " - This is a symbol: ", style = "Normal")
x <- slip_in_text(x, str = "SYMBOL 100 \u005Cf Wingdings",
  style = "strong")

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

---

**slip_in_text**

**append text**

Description

append text into a paragraph of an rdocx object

Usage

```r
slip_in_text(x, str, style = NULL, pos = "after", hyperlink = NULL)
```

Arguments

- **x**: an rdocx object
- **str**: text
- **style**: text style
- **pos**: where to add the new element relative to the cursor, "after" or "before".
- **hyperlink**: turn the text into an external hyperlink

Note

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

```r
x <- read_docx()
x <- body_add_par(x, "Hello ", style = "Normal")
x <- slip_in_text(x, "world", style = "strong")
x <- slip_in_text(x, "Message is", style = "strong", pos = "before")
x <- slip_in_text(x, "with a link", style = "strong",
                   pos = "after", hyperlink = "https://davidgohel.github.io/officer/")

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

---

### styles_info

**read Word styles**

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

#### Usage

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

#### Arguments

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

#### See Also

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

#### Examples

```r
x <- read_docx()
stypes_info()
stypes_info(x)
stypes_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()`

---

**table_conditional_formatting**  
*Table conditional formatting*

**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
)
```
**table_layout**

**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)
```

---

**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()`
table_stylenames  Paragraph styles for columns

Description

The function defines the paragraph styles for columns.

Usage

table_stylenames(stylenames = list())

Arguments

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

See Also

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

Examples

library(officer)

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

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

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

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

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

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

---

**table_width**

*Preferred width for a table*

---

**Description**

Define the preferred width for a table.

**Usage**

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

**Arguments**

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

**Word**

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

**See Also**

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

---

**unordered_list**

*Unordered list*

---

**Description**

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

**Usage**

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

**Arguments**

- `str_list`: list of strings to be included in the object
- `level_list`: list of levels for hierarchy structure
- `style`: text style, a `fp_text` object list or a single `fp_text` objects. Use `fp_text(font.size = 0,...)` to inherit from default sizes of the presentation.
See Also

`ph_with`

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

Examples

```r
unordered_list(
  level_list = c(1, 2, 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, 89
  run_reference, 93
  run_word_field, 94

* block functions for reporting
  block_caption, 6
  block_list, 7
  block_pour_docx, 8
  block_section, 9
  block_table, 10
  block_toc, 11
  fpar, 44
  plot_instr, 80
  unordered_list, 107

* functions for Word document informations
  doc_properties, 41
  docx_bookmarks, 39
  docx_dim, 40
  length.rdocx, 54
  set_doc_properties, 96
  styles_info, 103

* 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, 30
  body_end_section_custom_state, 31
  body_set_default_section, 34

* functions for adding content
  body_add_blocks, 15
  body_add_break, 16
  body_add_caption, 16
  body_add_docx, 17
  body_add_fpar, 18
  body_add_gg, 19
  body_add_img, 20
  body_add_par, 21
  body_add_plot, 22
  body_add_table, 23
  body_add_toc, 24

* functions for defining formatting properties
  fp_border, 45
  fp_cell, 46
  fp_par, 48
  fp_text, 49

* functions for placeholder location
  ph_location, 66
  ph_location_fullsize, 67
  ph_location_label, 68
  ph_location_left, 69
  ph_location_right, 70
  ph_location_template, 70
  ph_location_type, 72

* functions for placeholders manipulation
  ph_hyperlink, 65
  ph_remove, 74
  ph_slidelink, 75

* functions for reading presentation informations
  annotate_base, 5
  color_scheme, 36
  layout_properties, 53
  layout_summary, 54
  length.rpptx, 55
  plot_layout_properties, 81
  slide_size, 98
  slide_summary, 98

* functions for section definition
  page_mar, 59
  page_size, 60
  prop_section, 83
  section_columns, 95

* functions for table definition
  prop_table, 85
INDEX

functions slide manipulation
add_slide, 5
move_slide, 56
on_slide, 58
remove_slide, 89

* run functions for reporting
external_img, 43
ftext, 51
hyperlink_ftext, 52
run_autonum, 89
run_bookmark, 91
run_columnbreak, 91
run_linebreak, 92
run_pagbreak, 93
run_reference, 93
run_word_field, 94

add_sheet, 4
add_slide, 5, 57, 59, 89
add_slide(), 88
annotate_base, 5, 36, 53–55, 82, 98, 99

block_caption, 6, 7, 9–11, 14, 44, 81, 108
block_caption(), 7, 16
block_list, 6, 7, 9–11, 14, 44, 61, 62, 64, 78, 81, 100, 108
block_list(), 15, 44
block_pour_docx, 6, 7, 8, 10, 11, 14, 44, 81, 108
block_pour_docx(), 7
block_section, 6, 7, 9, 9, 10, 11, 14, 25, 44, 81, 84, 108
block_section(), 7
block_table, 6, 7, 9, 10, 10, 11, 44, 81, 108
block_table(), 7, 14, 78
block_toc, 6, 7, 9, 10, 11, 14, 44, 81, 108
block_toc(), 7
body_add, 12, 43
body_add_blocks, 15, 16–22, 24
body_add_blocks(), 7, 42
body_add_break, 15, 16, 17–22, 24
body_add_caption, 15, 16, 16, 18–22, 24
body_add_docx, 15–17, 17, 19–22, 24
body_add_fpar, 15–18, 18, 20–22, 24
body_add_fpar(), 44
body_add_gg, 15–19, 19, 21, 22, 24
body_add_img, 15–20, 20, 21, 22, 24
body_add_par, 15–21, 21, 22, 24, 86
body_add_plot, 15–21, 22, 24, 86
body_add_plot(), 80, 81
body_add_table, 15–22, 23, 24, 86
body_add_toc, 15–22, 24, 24
body_bookmark, 25
body_end_block_section, 25, 27–30, 35
body_end_section_columns, 26, 26, 28–30, 35
body_end_section_columns_landscape, 26, 27, 28–30, 35
body_end_section_continuous, 26–28, 29, 30, 35
body_end_section_landscape, 26–28, 29, 30, 35
body_end_section_portrait, 26–29, 30, 35
body_remove, 30
body_replace_all_text, 31, 40
body_replace_img_at_bkm
(body_replace_text_at_bkm), 33
body_replace_img_at_bkm
(body_replace_text_at_bkm), 33
body_set_default_section, 26–30, 34
change_styles, 35
color_scheme, 6, 36, 53–55, 82, 98, 99
cursor_backward(cursor_begin), 37
cursor_begin, 37
cursor_bookmark(cursor_begin), 37
cursor_end(cursor_begin), 37
cursor_forward(cursor_begin), 37
cursor_reach(cursor_begin), 37
docx_bookmarks, 39, 40, 42, 55, 96, 103
docx_dim, 39, 40, 42, 55, 96, 103
docx_summary, 41
docx_show_chunk, 32, 33, 40
docx_summary, 41
empty_content, 42, 78
external_img, 14, 43, 52, 53, 78, 90–95
external_img(), 7, 44
footers_replace_all_text
(body_replace_all_text), 31
footers_replace_img_at_bkm
(body_replace_text_at_bkm), 33
run_seqfield (run_word_field), 94
run_seqfield(), 44
run_word_field, 43, 52, 53, 90–94, 94

sanitize_images, 95
section_columns, 60, 84, 95
set_doc_properties, 39, 40, 42, 55, 96, 103
sheet_select, 97
shortcuts, 97
slide_size, 6, 36, 53–55, 82, 98, 99
slide_summary, 6, 36, 53–55, 61, 62, 64, 65, 74, 75, 82, 98, 98
slip_in_column_break, 99
slip_in_footnote, 100
slip_in_img, 100
slip_in_seqfield, 101
slip_in_text, 102
styles_info, 13, 39, 40, 42, 55, 96, 103
styles_info(), 35

table_colwidths, 85, 104, 105–107
table_colwidths(), 85
table_conditional_formatting, 85, 104, 104, 105–107
table_conditional_formatting(), 13, 77, 85
table_layout, 85, 104, 105, 106, 107
table_layout(), 85
table_stylenames, 85, 104, 105, 106, 107

table_stylenames(), 23, 85
table_width, 85, 104–106, 107
table_width(), 85

unordered_list, 6, 7, 9–11, 44, 78, 81, 107
update.fp_border (fp_border), 45
update.fp_cell (fp_cell), 46
update.fp_par (fp_par), 48
update.fp_text (fp_text), 49
update.fpar (fpar), 44