Package 'officer'

May 15, 2026

Type Package

Title Manipulation of Microsoft Word and PowerPoint Documents

Version 0.7.5

Description Access and manipulate 'Microsoft Word', 'RTF' and 'Microsoft PowerPoint' documents from R. The package focuses on tabular and graphical reporting from R; it also provides two functions that let users get document content into data objects. A set of functions lets add and remove images, tables and paragraphs of text in new or existing documents. The package does not require any installation of Microsoft products to be able to write Microsoft files.

License MIT + file LICENSE

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

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

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

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

Encoding UTF-8

Collate 'core_properties.R' 'custom_properties.R' 'defunct.R' 'dev-utils.R' 'docx_add.R' 'docx_comments.R' 'docx_cursor.R' 'docx_embed_font.R' 'docx_part.R' 'docx_replace.R' 'docx_section.R' 'docx_settings.R' 'docx_styles.R' 'docx_utils_funs.R' 'empty_content.R' 'formatting_properties.R' 'fortify_docx.R' 'fortify_pptx.R' 'knitr_utils.R' 'officer.R' 'ooxml.R' 'ooxml_block_objects.R' 'ooxml_run_objects.R' 'openxml_content_type.R' 'openxml_document.R' 'pack_folder.R' 'ph_location.R' 'post-proc.R' 'ppt_class_dir_collection.R' 'ppt_classes.R' 'ppt_notes.R' 'ppt_ph_dedupe_layout.R' 'ppt_ph_manipulate.R' 'ppt_ph_rename_layout.R' 'ppt_ph_with_methods.R' 'pptx_informations.R' 'pptx_layout_helper.R' 'pptx_matrix.R' 'utils.R'

'pptx_slide_manip.R' 'read_docx.R' 'docx_write.R'
'read_docx_styles.R' 'read_pptx.R' 'read_xlsx.R'
'relationship.R' 'rtf.R' 'shape_properties.R' 'shorcuts.R'
'docx_append_context.R' 'utils-xml.R' 'deprecated.R' 'zzz.R'

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author David Gohel [aut, cre],
Stefan Moog [aut],
Mark Heckmann [aut] (ORCID: <https://orcid.org/0000-0002-0736-7417>),
ArData [cph],
Frank Hangler [ctb] (function body_replace_all_text),
Liz Sander [ctb] (several documentation fixes),
Anton Victorson [ctb] (fixes xml structures),
Jon Calder [ctb] (update vignettes),
John Harrold [ctb] (function annotate_base),
John Muschelli [ctb] (google doc compatibility),
Bill Denney [ctb] (ORCID: <https://orcid.org/0000-0002-5759-428X>,
function as.matrix.rpptx),
Nikolai Beck [ctb] (set speaker notes for .pptx documents),
Greg Leleu [ctb] (fields functionality in ppt),
Majid Eismann [ctb],
Wahiduzzaman Khan [ctb] (vectorization of remove_slide),
Hongyuan Jia [ctb] (ORCID: <https://orcid.org/0000-0002-0075-8183>),
Michael Stackhouse [ctb]

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

Repository CRAN

Date/Publication 2026-05-15 14:10:02 UTC

Contents

`add_sheet` *Add a sheet*

Description

Add a sheet into an xlsx worksheet.

Usage

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

Arguments

x	rxlsx object
label	sheet label

Details

read_xlsx() returns a workbook that already contains one default sheet shipped with the template (named "Sheet1" or "Feuil1" depending on the locale). add_sheet() is purely additive: the default sheet is kept as-is. Remove it explicitly with sheet_remove() if it is not wanted.

Examples

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

---

add_slide *Add a slide*

Description

Add a slide into a pptx presentation.

Usage

```
add_slide(x, layout = NULL, master = NULL, ..., .dots = NULL)
```

Arguments

x	an rpptx object.
layout	slide layout name to use. Can be ommited of a default layout is set via layout_default().
master	master layout name where layout is located. Only required in case of several masters if layout is not unique.
...	Key-value pairs of the form "short form location" = object passed to phs_with. See section "Short forms" in phs_with for details, available short forms and examples.
.dots	List of key-value pairs of the form list("short form location" = object). Alternative to .... See phs_with for details.

See Also

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

Other functions to manipulate slides: move_slide(), on_slide(), remove_slide(), set_notes()

Examples

```
x <- read_pptx()

layout_summary(x) # available layouts

x <- add_slide(x, layout = "Two Content")

x <- layout_default(x, "Title Slide") # set default layout for `add_slide()`
x <- add_slide(x) # uses default layout

# use `...` to fill placeholders when adding slide
x <- add_slide(
  x,
  layout = "Two Content",
  `Title 1` = "A title",
  dt = "Jan. 26, 2025",
  `body[2]` = "Body 2",
  left = "Left side",
  `6` = "Footer"
)
```

---

annotate_base *Placeholder parameters annotation*

Description

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

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

Note that if there are duplicated `ph_label`, you should not use `ph_location_label()`. Hint: You can dedupe labels using `layout_dedupe_ph_labels()`.

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 information: color_scheme(), doc_properties(), layout_properties(), layout_summary(), length.rpptx(), plot_layout_properties(), slide_size(), slide_summary()

Examples

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

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

---

as.matrix.rpptx *PowerPoint table to matrix*

Description

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

Usage

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

Arguments

x	The rpptx object to convert (as created by read_pptx())
...	Ignored
slide_id	The slide number to load from (NA indicates first slide with a table, NULL indicates all slides and all tables)
id	The table ID to load from (ignored it is.null(slide_id), NA indicates to load the first table from the slide_id)
span	How should col_span/row_span values be handled? NA means to leave the value as NA, and "fill" means to fill matrix cells with the value.

Value

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

Examples

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

---

block_caption *Caption block*

Description

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

Usage

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

Arguments

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

See Also

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

Examples

```
library(officer)

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

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

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

block_gg *'ggplot' block*

Description

A simple wrapper to add a 'ggplot' object as a png in a document. It produces an object of class 'block_gg' with a corresponding method `to_wml()` that can be used to convert the object to a WordML string.

Usage

```
block_gg(
  value,
  fp_p = fp_par(),
  width = 6,
  height = 5,
  res = 300,
  scale = 1,
  unit = "in"
)
```

Arguments

value	'ggplot' object
fp_p	paragraph formatting properties, see `fp_par()`
width, height	image size in units expressed by the unit argument. Defaults to "in"ches.
res	resolution of the png image in ppi
scale	Multiplicative scaling factor, same as in ggsave
unit	One of the following units in which the width and height arguments are expressed: "in", "cm" or "mm".

See Also

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

Examples

```
library(officer)
if (require("ggplot2")) {
  set.seed(2)
  doc <- read_docx()

  z <- body_append_start_context(doc)
```

```
  for (i in seq_len(3)) {
    df <- data.frame(x = runif(10), y = runif(10))
    gg <- ggplot(df, aes(x = x, y = y)) + geom_point()

    write_elements_to_context(
      context = z,
      block_gg(
        value = gg
      )
    )
  }

  doc <- body_append_stop_context(z)

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

`block_list` *List of blocks*

Description

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

Usage

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

Arguments

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

See Also

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

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

Examples

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

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

```
## This can be only be used in a MS word output as pptx does
## not support paragraphs made of text and images.
## (actually it can be used but image will not appear in the
## pptx output)
value <- block_list(
  fpar(ftext("hello world", fpt_blue_bold)),
  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, "Title and Content")
doc <- ph_with(doc, value, location = ph_location_type(type = "body"))
print(doc, target = tempfile(fileext = ".pptx"))
```

`block_list_items` *List of items for Word and PowerPoint*

Description

Create a bullet or numbered list from `list_item()` elements. Supports rich text via `fpar()` and multi-level nesting. Works in both Word and PowerPoint documents.

Usage

```
block_list_items(..., list_type = "bullet")
```

Arguments

`...`	`list_item()` objects
`list_type`	`"bullet"` for an unordered list or `"decimal"` for a numbered list

See Also

list_item(), body_add(), ph_with()

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

Examples

```
items <- block_list_items(
  list_item(fpar(
    ftext("Item 1", fp_text(color = "red"))
  ), level = 1),
  list_item(fpar("Sub-item"), level = 2),
  list_item(fpar("Item 2"), level = 1),
  list_type = "bullet"
)
items
```

---

block_pour_docx *External Word document placeholder*

Description

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

Usage

```
block_pour_docx(file)
```

Arguments

file	external docx file path

See Also

Other block functions for reporting: block_caption(), block_gg(), block_list(), block_list_items(), block_section(), block_table(), block_toc(), fpar(), list_item(), plot_instr(), unordered_list()

Examples

```
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 *Section for 'Word'*

Description

Create a representation of a section.

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

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

Usage

```
block_section(property)
```

Arguments

property	section properties defined with function prop_section

See Also

Other block functions for reporting: `block_caption()`, `block_gg()`, `block_list()`, `block_list_items()`, `block_pour_docx()`, `block_table()`, `block_toc()`, `fpar()`, `list_item()`, `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_gg(), block_list(), block_list_items(), block_pour_docx(), block_section(), block_toc(), fpar(), list_item(), plot_instr(), unordered_list()

Examples

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

block_table(x = mtcars, header = TRUE,
  properties = prop_table(
    tcf = table_conditional_formatting(
      first_row = TRUE, first_column = TRUE)
  ))
```

---

block_toc *Table of content for 'Word'*

Description

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

Usage

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

Arguments

level	max title level of the table
style	optional. If not NULL, its value is used as style in the document that will be used to build entries of the TOC.
seq_id	optional. If not NULL, its value is used as sequence identifier in the document that will be used to build entries of the TOC. See also run_autonum() to specify a sequence identifier.
separator	unused, no effect

See Also

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

Examples

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

---

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

Description

Add a list of blocks produced by block_list() into 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_list(), body_add_par(), body_add_plot(), body_add_table(), body_add_toc(), body_append_start_context(), body_import_docx()

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 *Add a page break in a 'Word' document*

Description

Add a page break into an rdocx object

Usage

```
body_add_break(x, pos = "after")
```

Arguments

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

See Also

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

Examples

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

---

body_add_caption *Add Word caption in a 'Word' document*

Description

Add a Word caption into an rdocx object.

Usage

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

Arguments

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

See Also

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

Examples

```
doc <- read_docx()

if (capabilities(what = "png")) {
  doc <- body_add_plot(doc,
    value = plot_instr(
      code = {
        barplot(1:5, col = 2:6)
      }
    ),
    style = "centered"
  )
}
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 *Add an external docx in a 'Word' document*

Description

Add content of a docx into an rdocx object.

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. You can use function body_import_docx() to import the content as an alternative.

The file is added when the method print() that produces the final Word file is called, so don't remove file defined with src before.

Usage

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

Arguments

x	an rdocx object
src	docx filename, the path of the file must not contain any '&' and the basename must not contain any space.
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_fpar(), body_add_gg(), body_add_img(), body_add_list(), body_add_par(), body_add_plot(), body_add_table(), body_add_toc(), body_append_start_context(), body_import_docx()

Examples

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

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

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

---

body_add_fpar *Add fpar in a 'Word' document*

Description

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

Usage

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

Arguments

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

See Also

fpar()

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

Examples

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

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

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

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

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

Description

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

Usage

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

Arguments

x	an rdocx object
value	ggplot object
width, height	plot size in units expressed by the unit argument. Defaults to a width of 6 and a height of 5 "in"ches.
res	resolution of the png image in ppi
style	paragraph style
scale	Multiplicative scaling factor, same as in ggsave
pos	where to add the new element relative to the cursor, one of "after", "before", "on".
unit	One of the following units in which the width and height arguments are expressed: "in", "cm" or "mm".
...	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_list(), body_add_par(), body_add_plot(), body_add_table(), body_add_toc(), body_append_start_context(), body_import_docx()

Examples

```
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")

    # Set the unit in which the width and height arguments are expressed
    doc <- body_add_gg(doc, value = gg_plot, style = "centered", unit = "cm")
  }

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

---

body_add_img *Add an image in a 'Word' document*

Description

Add an image into an rdocx object.

Usage

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

Arguments

x	an rdocx object
src	image filename, the basename of the file must not contain any blank.
style	paragraph style
width, height	image size in units expressed by the unit argument. Defaults to "in"ches.
pos	where to add the new element relative to the cursor, one of "after", "before", "on".
unit	One of the following units in which the width and height arguments are expressed: "in", "cm" or "mm".

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_list(), body_add_par(), body_add_plot(), body_add_table(), body_add_toc(), body_append_start_context(), body_import_docx()

Examples

```
library(officer)

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)

  # Set the unit in which the width and height arguments are expressed
  doc <- body_add_img(
    x = doc,
    src = img.file,
    height = 2.69,
    width = 3.53,
    unit = "cm"
  )
}

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

---

body_add_list *Add a list of items into a 'Word' document*

Description

Add a bullet or numbered list produced by `block_list_items()` into an rdocx object.

Usage

```
body_add_list(x, items, pos = "after")
```

Arguments

x	an rdocx object
items	a `block_list_items()` 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_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()`, `body_append_start_context()`, `body_import_docx()`

Examples

```
library(officer)

items <- block_list_items(
  list_item(fpar(ftext("Item 1", fp_text(color = "red"))), level = 1),
  list_item(fpar("Sub-item"), level = 2),
  list_item(fpar("Item 2"), level = 1),
  list_type = "bullet"
)

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

---

body_add_par *Add paragraphs of text in a 'Word' document*

Description

Add a paragraph of text into an rdocx object

Usage

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

Arguments

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

See Also

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

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 *Add plot in a 'Word' document*

Description

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

Usage

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

Arguments

x	an rdocx object
value	plot instructions, see plot_instr().
width, height	plot size in units expressed by the unit argument. Defaults to a width of 6 and a height of 5 "in"ches.
res	resolution of the png image in ppi
style	paragraph style
pos	where to add the new element relative to the cursor, one of "after", "before", "on".
unit	One of the following units in which the width and height arguments are expressed: "in", "cm" or "mm".
...	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_list(), body_add_par(), body_add_table(), body_add_toc(), body_append_start_context(), body_import_docx()

Examples

```
doc <- read_docx()

if (capabilities(what = "png")) {
  p <- plot_instr(
      code = {
        barplot(1:5, col = 2:6)
      }
    )

  doc <- body_add_plot(doc, value = p, style = "centered")

  # Set the unit in which the width and height arguments are expressed
  doc <- body_add_plot(doc, value = p, style = "centered", unit = "cm")
}

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

---

body_add_table *Add table in a 'Word' document*

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 alignement, argument length must match with columns length, values must be "l" (left), "r" (right) or "c" (center).
align_table	table alignment within document, value must be "left", "center" or "right"
stylenames	columns styles defined by table_stylenames()
first_row	Specifies that the first column conditional formatting should be applied. Details for this and other conditional formatting options can be found at http://officeopenxml.com/WPtblLook.php
first_column	Specifies that the first column conditional formatting should be applied.
last_row	Specifies that the first column conditional formatting should be applied.
last_column	Specifies that the first column conditional formatting should be applied.
no_hband	Specifies that the first column conditional formatting should be applied.
no_vband	Specifies that the first column conditional formatting should be applied.

See Also

Other functions for adding content: body_add_blocks(), body_add_break(), body_add_caption(), body_add_docx(), body_add_fpar(), body_add_gg(), body_add_img(), body_add_list(), body_add_par(), body_add_plot(), body_add_toc(), body_append_start_context(), body_import_docx()

Examples

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

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

---

body_add_toc *Add table of content in a 'Word' document*

Description

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

Usage

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

Arguments

x	an rdocx object
level	max title level of the table
pos	where to add the new element relative to the cursor, one of "after", "before", "on".
style	optional. style in the document that will be used to build entries of the TOC.
separator	unused, no effect

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_list(), body_add_par(), body_add_plot(), body_add_table(), body_append_start_context(), body_import_docx()

Examples

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

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

---

body_append_start_context

Fast Append context to a Word document

Description

This function is used to append content to a Word document in a fast way.

It does not use the XML tree of the document neither the cursor that is responsible for increasing the performance of Word document generation when looping over a large number of elements.

This function must be used with the write_elements_to_context() and body_append_stop_context() functions:

1. body_append_start_context() creates a context and returns a list with the context and the file connection.
2. write_elements_to_context() writes the elements to the context file connection.
3. body_append_stop_context() closes the file connection and replaces the XML in the document with the new XML.

Usage

```
body_append_start_context(x, additional_ns = character())

write_elements_to_context(context, ...)

body_append_stop_context(context)
```

Arguments

x	an rdocx object
additional_ns	a named character vector of additional XML namespaces to be added to the root node of the document. The names of the vector are the namespace prefixes and the values are the namespace URIs. This argument is useful when the elements to be added to the document require additional namespaces that are not already present in the document and not part of the xml generated by to_wml(). Simple users are not expected to use this argument. It is mainly intended for developers of officer extensions.
context	the context object created by body_append_start_context().
...	elements to be written to the context. These can be paragraphs, tables, images, etc. The elements should have an associated to_wml() method that converts them to WML format.

Value

body_append_start_context() returns a list representing the context that contains:

- doc: the original document object
- file_con: the file connection to the context
- file_path: the path to the context file
- final_str: the final XML string to be appended to the document later when calling body_append_stop_context().

This object should not be modified by the user but instead passed to write_elements_to_context() and body_append_stop_context().

write_elements_to_context() returns the context object.

body_append_stop_context() returns the rdocx object with the cursor position set to the end of the document.

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_list(), body_add_par(), body_add_plot(), body_add_table(), body_add_toc(), body_import_docx()

Examples

```
library(officer)

doc <- read_docx()
doc <- body_add_par(doc, value = "blah blah blah", style = "Normal")

z <- body_append_start_context(doc)

for (i in seq_len(50)) {
  write_elements_to_context(
    context = z,
```

```
    fpar(
      "Hello World, ",
      i,
      fp_p = fp_par(word_style = "heading 1")
    ),
    fpar(run_pagebreak())
  )
}
doc <- body_append_stop_context(z)


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

---

body_bookmark *Add bookmark in a 'Word' document*

Description

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

Usage

```
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_comment *Add comment in a 'Word' document*

Description

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

Usage

```
body_comment(x, cmt = ftext(""), author = "", date = "", initials = "")
```

Arguments

x	an rdocx object
cmt	a set of blocks to be used as comment content returned by function block_list().
author	comment author.
date	comment date
initials	comment initials

Examples

```
doc <- read_docx()
doc <- body_add_par(doc, "Paragraph")
doc <- body_comment(doc, block_list("This is a comment."))
docx_file <- print(doc, target = tempfile(fileext = ".docx"))
docx_comments(read_docx(docx_file))
```

body_end_block_section *Add any section*

Description

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

Usage

```
body_end_block_section(x, value)
```

Arguments

x	an rdocx object
value	a block_section object

Section model in Word

A `block_section` added with `body_end_block_section()` applies to the content that **precedes** the call: it closes the section that holds the previous paragraphs / tables and inherits any layout (orientation, columns, margins, headers / footers) defined by the `block_section`. The function name reflects this: it marks the *end* of a section.

Typical pattern: add the content, then close it with the section that should layout it.

```
doc <- read_docx() |>
  body_add_par("This paragraph is in landscape orientation.") |>
  body_end_block_section(block_section(prop_section(
    page_size = page_size(orient = "landscape")
  )))
```

The default section of the document (defined by the template or by `body_set_default_section()`) closes any content added after the last `body_end_block_section()` call.

The RTF output uses the opposite model: `rtf_add(block_section(...))` applies to the content that *follows* the call. See `rtf_add()`.

Illustrations

See Also

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

Examples

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

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

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

doc_1 <- body_end_block_section(doc_1, block_section(ps))

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

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

body_end_section_columns *Add multi columns section*

Description

A section with multiple columns is added to the document.

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

Usage

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

Arguments

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

See Also

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

Examples

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

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

`body_end_section_columns_landscape` *Add a landscape multi columns section*

Description

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

Usage

```
body_end_section_columns_landscape(
  x,
  widths = c(2.5, 2.5),
  space = 0.25,
  sep = FALSE,
  w = 16838/1440,
  h = 11906/1440
)
```

Arguments

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

See Also

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

Examples

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

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

body_end_section_continuous *Add continuous section*

Description

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

Usage

```
body_end_section_continuous(x)
```

Arguments

x	an rdocx object

See Also

Other functions for Word sections: body_end_block_section(), body_end_section_columns(), body_end_section_columns_landscape(), 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)
str2 <- paste(str2, collapse = " ")

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

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

body_end_section_landscape

Add landscape section

Description

A section with landscape orientation is added to the document.

Usage

```
body_end_section_landscape(x, w = 16838/1440, h = 11906/1440)
```

Arguments

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

See Also

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

Examples

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

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

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

body_end_section_portrait

Add portrait section

Description

A section with portrait orientation is added to the document.

Usage

```
body_end_section_portrait(x, w = 16838/1440, h = 11906/1440)
```

Arguments

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

See Also

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

Examples

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

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

---

body_import_docx *Import an external docx in a 'Word' document*

Description

Import body content and footnotes of a Word document into an rdocx object.

The function is similar to body_add_docx() but instead of adding the content as an external object, the document is read and all its content is appended to the target document.

Usage

```
body_import_docx(
  x,
  src,
  par_style_mapping = list(),
  run_style_mapping = list(),
  tbl_style_mapping = list(),
  prepend_chunks_on_styles = list()
)
```

Arguments

x	an rdocx object
src	path to docx file to import

par_style_mapping, run_style_mapping, tbl_style_mapping	Named lists describing how to remap styles from the source document (src) to styles available in the target document x. For each list entry, the name of the element is the target style (in x), and the value is a character vector of style names from the source document that should be replaced by this target style.
	• par_style_mapping: applies to paragraph styles.
	• run_style_mapping: applies to character (run) styles.
	• tbl_style_mapping: applies to table styles.
	Examples:
	`par_style_mapping = list(`
	`  "Normal"    = c("List Paragraph", "Body Text"),`
	`  "heading 1" = "Heading 1"`
	`)`
	`run_style_mapping = list(`
	`  "Emphasis"  = c("Emphasis", "Italic")`
	`)`
	`tbl_style_mapping = list(`
	`  "Normal Table" = c("Light Shading")`
	`)`
	Use styles_info() to inspect available styles and verify their names.

prepend_chunks_on_styles	A named list of run chunks to prepend to runs with specific styles. The names of the list are paragraph style names and the values run chunks to prepend. The first motivation for this argument is to allow prepending of runs in paragraphs with a defined style, for example to add a run_autonum() with all image captions.

Details

The following operations are performed when importing a document:

- Numberings are copied from the source document to the target document.
- Styles are not copied. If styles in the source document do not exist in the target document, the style specified in the par_style_mapping, run_style_mapping and tbl_style_mapping arguments will be used instead. If no mapping is provided, the default style will be used and a warning is emitted.

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_list(), body_add_par(), body_add_plot(), body_add_table(), body_add_toc(), body_append_start_context()

Examples

```
library(officer)

# example file from the package
file_input <- system.file(
  package = "officer",
  "doc_examples/example.docx"
)

# create a new rdocx document
x <- read_docx()

# import content from file_input
x <- body_import_docx(
  x = x,
  src = file_input,
  # style mapping for paragraphs and tables
  par_style_mapping = list(
    "Normal" = c("List Paragraph")
  ),
  tbl_style_mapping = list(
    "Normal Table" = "Light Shading"
  )
)

# Create temporary file
tf <- tempfile(fileext = ".docx")
# write to file
print(x, target = tf)
```

---

body_remove *Remove an element in a 'Word' document*

Description

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

Usage

```
body_remove(x)
```

Arguments

x	an rdocx object

Examples

```
library(officer)

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

str2 <- "Drop that text"

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

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

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

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

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

---

body_replace_all_text *Replace text anywhere in the document*

Description

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

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

Note that by default, grepl/gsub will use `fixed=FALSE`, which means that `old_value` and `new_value` will be interepreted 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 outputing 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

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

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

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

Arguments

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

header_replace_all_text

Replacements will be performed in each header of all sections.

Replacements will be performed in each footer of all sections.

Author(s)

Frank Hangler, <frank@plotandscatter.com>

See Also

grepl(), regex(), docx_show_chunk()

Examples

```
library(officer)

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_gg_at_bkm

Add plots at bookmark location in a 'Word' document

Description

Use these functions if you want to replace a paragraph containing a bookmark with a 'ggplot' or a base plot.

Usage

```
body_replace_gg_at_bkm(
  x,
  bookmark,
  value,
  width = 6,
  height = 5,
  res = 300,
  style = "Normal",
  scale = 1,
  keep = FALSE,
  ...
)

body_replace_plot_at_bkm(
  x,
  bookmark,
  value,
  width = 6,
  height = 5,
  res = 300,
  style = "Normal",
  keep = FALSE,
  ...
)
```

Arguments

x	an rdocx object
bookmark	bookmark id
value	a ggplot object for body_replace_gg_at_bkm() or a set plot instructions body_replace_plot_at_bkm(), see plot_instr().
width, height	plot size in units expressed by the unit argument. Defaults to a width of 6 and a height of 5 "in"ches.
res	resolution of the png image in ppi
style	paragraph style
scale	Multiplicative scaling factor, same as in ggsave
keep	Should the bookmark be preserved? Defaults to FALSE, i.e.the bookmark will be lost as a side effect.
...	Arguments to be passed to png function.

Examples

```
library(officer)

if (require("ggplot2")) {
```

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

  doc <- read_docx()
  doc <- body_add_par(doc, "insert_plot_here")
  doc <- body_bookmark(doc, "plot")
  doc <- body_replace_gg_at_bkm(doc, bookmark = "plot", value = gg_plot)
  print(doc, target = tempfile(fileext = ".docx"))
}
doc <- read_docx()
doc <- body_add_par(doc, "insert_plot_here")
doc <- body_bookmark(doc, "plot")
if (capabilities(what = "png")) {
  doc <- body_replace_plot_at_bkm(
    doc,
    bookmark = "plot",
    value = plot_instr(
      code = {
        barplot(1:5, col = 2:6)
      }
    )
  )
}
print(doc, target = tempfile(fileext = ".docx"))
```

`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

```
library(officer)

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


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

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

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

---

body_set_default_section

Define Default Section

Description

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

Usage

```
body_set_default_section(x, value)
```

Arguments

x	an rdocx object
value	a prop_section object

Illustrations

See Also

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

Examples

```
# Example 1: Setting page layout properties ----
# This example demonstrates how to configure the default section
# properties for page orientation, type, and margins

# Define custom section properties
# - Landscape orientation for wide tables
# - Continuous section (no page break)
# - Custom margins: top=1.5", bottom=0.75", left/right=2"
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)
)

# Create a new document
doc_1 <- read_docx()

# Add a wide table that benefits from landscape orientation
doc_1 <- body_add_par(doc_1, "Motor Trend Car Road Tests", style = "heading 1")
doc_1 <- body_add_table(doc_1, value = mtcars[1:10, ], style = "table_template")

# Add some text content
doc_1 <- body_add_par(doc_1, "Sample Text Content", style = "heading 2")
doc_1 <- body_add_par(doc_1, value = paste(rep(letters, 40), collapse = " "))

# Apply the section properties to the entire document
# This must be called at the end, after all content is added
doc_1 <- body_set_default_section(doc_1, default_sect_properties)

# Save the document
print(doc_1, target = tempfile(fileext = ".docx"))


# Example 2: Adding headers and footers ----
# This example shows how to create a document with:
```

```
# - A header containing the R logo
# - A footer with the current date and page numbers

# Get the path to R logo
img_path <- file.path(R.home("doc"), "html", "logo.jpg")

# Create header content with the R logo
# The logo is positioned on the right side with specific dimensions
header_content <- block_list(
  fpar(
    external_img(src = img_path, height = 0.5, width = 0.5),
    fp_p = fp_par(text.align = "right")
  )
)

# Create footer content with date and page numbers
# Format: "Document generated on: [Date] | Page [X]"
footer_content <- block_list(
  fpar(
    "Document generated on: ",
    run_word_field(field = "Date \\@ \"MMMM d, yyyy\""),
    " | Page ",
    run_word_field(field = "PAGE"),
    fp_p = fp_par(text.align = "center")
  )
)

# Define section properties that include header and footer
# The header and footer will appear on all pages
sect_with_hf <- prop_section(
  page_size = page_size(orient = "portrait", width = 8.3, height = 11.7),
  page_margins = page_mar(
    bottom = 1,
    top = 1,
    right = 1,
    left = 1,
    header = 0.5,
    footer = 0.5
  ),
  type = "continuous",
  header_default = header_content,
  footer_default = footer_content
)

# Create a new document with content
doc_2 <- read_docx()

# Add a title page
doc_2 <- body_add_par(doc_2, "Annual Report 2024", style = "heading 1")
doc_2 <- body_add_par(
  doc_2,
  "Company Performance Analysis",
  style = "heading 2"
```

```
)

# Add some sections with content
doc_2 <- body_add_par(doc_2, "Executive Summary", style = "heading 2")
doc_2 <- body_add_par(
  doc_2,
  "This report provides a comprehensive analysis of company performance metrics."
)

# Add a table
doc_2 <- body_add_par(doc_2, "Key Metrics", style = "heading 2")
summary_data <- data.frame(
  Metric = c("Revenue", "Profit", "Growth"),
  Q1 = c(1.2, 0.3, 12),
  Q2 = c(1.5, 0.4, 15),
  Q3 = c(1.8, 0.5, 18),
  Q4 = c(2.1, 0.6, 20)
)
doc_2 <- body_add_table(doc_2, value = summary_data, style = "table_template")

# Add a plot
doc_2 <- body_add_par(doc_2, "Revenue Trend", style = "heading 2")
revenue_plot <- plot_instr({
  quarters <- paste0("Q", 1:4)
  revenue <- c(1.2, 1.5, 1.8, 2.1)
  barplot(
    revenue,
    names.arg = quarters,
    col = "#4472C4",
    border = NA,
    main = "Quarterly Revenue (Millions)",
    ylab = "Revenue ($M)",
    xlab = "Quarter"
  )
})
doc_2 <- body_add_plot(doc_2, revenue_plot, width = 5, height = 4)

# Apply section properties with header and footer
# The header (with R logo) and footer (with date and page number)
# will appear on all pages
doc_2 <- body_set_default_section(doc_2, sect_with_hf)

# Save the document
output_file <- tempfile(fileext = ".docx")
print(doc_2, target = output_file)
```

---

change_styles *Replace styles in a 'Word' Document*

Description

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

Usage

```
change_styles(x, mapstyles)
```

Arguments

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

Examples

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

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

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

---

color_scheme *Color scheme of a PowerPoint file*

Description

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

Usage

```
color_scheme(x)
```

Arguments

x	an rpptx object

See Also

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

Examples

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

---

cursor_begin *Set cursor in a 'Word' document*

Description

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

Usage

```
cursor_begin(x)

cursor_bookmark(x, id)

cursor_end(x)

cursor_reach_index(x, index)

cursor_reach(x, keyword, fixed = FALSE)

cursor_reach_test(x, keyword)

cursor_forward(x)

cursor_backward(x)
```

Arguments

x	a docx device
id	bookmark id
index	element index in the document
keyword	keyword to look for as a regular expression
fixed	logical. If TRUE, pattern is a string to be matched as is.

cursor_begin

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

cursor_bookmark

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

cursor_end

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

cursor_reach_index

Set the cursor at a specific index position in the document.

cursor_reach

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

cursor_reach_test

Test if an expression has a match in the document that contains text specified in argument `keyword`. The argument `keyword` is a regexpr pattern.

cursor_forward

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

cursor_backward

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

Examples

```
library(officer)

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

# replace all pars containing "to replace" ----
doc <- read_docx(path = template_file)
```

```
while (cursor_reach_test(doc, "to replace")) {
  doc <- cursor_reach(doc, "to replace")

  doc <- body_add_fpar(
    x = doc,
    pos = "on",
    value = fpar(
      "Here is a link: ",
      hyperlink_ftext(
        text = "yopyop",
        href = "https://cran.r-project.org/"
      )
    )
  )
}

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

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

# cursor_bookmark ----

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

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

`docx_bookmarks` *List Word bookmarks*

Description

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

Usage

```
docx_bookmarks(x)
```

Arguments

x	an `rdocx` object

See Also

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

Examples

```
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_comments *Get comments in a Word document as a data.frame*

Description

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

Usage

```
docx_comments(x)
```

Arguments

x	an rdocx object

Details

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

- "comment_id" - unique comment id
- "author" - name of the comment author
- "initials" - initials of the comment author
- "date" - timestamp of the comment
- "text" - a list column of characters containing the comment text. Elements can be vectors of length > 1 if a comment contains multiple paragraphs, blocks or runs or of length 0 if the comment is empty.

- "para_id" - a list column of characters containing the parent paragraph IDs. Elememts can be vectors of length > 1 if a comment spans multiple paragraphs or of length 0 if the comment has no parent paragraph.
- "commented_text" - a list column of characters containing the commented text. Elements can be vectors of length > 1 if a comment spans multiple paragraphs or runs or of length 0 if the commented text is empty.

Examples

```
library(officer)

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

a_par <- fpar(
  "This paragraph contains",
  run_comment(
    cmt = bl,
    run = ftext("a comment."),
    author = "Author Me",
    date = "2023-06-01"
  )
)

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

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

docx_comments(read_docx(docx_file))
```

---

docx_dim *'Word' page layout*

Description

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

Usage

```
docx_dim(x)
```

Arguments

x	an 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_embed_font *Embed Fonts in a Word Document*

Description

Copy TrueType or OpenType font files from the local system into a Word document. The font data is stored inside the .docx archive so that the document renders correctly when opened on a system where the font is not installed.

When only font_family is provided (without regular), the function looks up font file paths on the local system via gdtools::sys_fonts() and copies them into the document. A message shows the equivalent explicit call for reproducibility.

Usage

```
docx_embed_font(
  x,
  font_family,
  regular = NULL,
  bold = NULL,
  italic = NULL,
  bold_italic = NULL
)
```

Arguments

x	an rdocx object
font_family	font family name as it will appear in the document. When regular is not provided, this name is used to look up font files via gdtools::sys_fonts().
regular	path to the regular .ttf/.otf font file. If NULL, font files are detected automatically via gdtools.
bold	path to the bold .ttf/.otf font file (optional)
italic	path to the italic .ttf/.otf font file (optional)
bold_italic	path to the bold-italic .ttf/.otf font file (optional)

Value

the rdocx object with embedded fonts

Font licensing

Embedding a font in a document redistributes it. You must ensure that the font license permits embedding. Fonts under the SIL Open Font License (e.g. Liberation, Google Fonts) generally allow it. Many commercial fonts restrict or prohibit embedding. Check the license of the font before using this function.

See Also

gdtools::sys_fonts(), gdtools::register_gfont(), docx_set_settings()

Examples

```
# automatic detection (requires gdtools)
gdtools::register_liberationsans()
doc <- read_docx()
doc <- docx_embed_font(doc, font_family = "Liberation Sans")

# # explicit paths (no gdtools needed)
# doc <- docx_embed_font(
#   doc,
#   font_family = "My Font",
#   regular = "path/to/font-regular.ttf",
#   bold = "path/to/font-bold.ttf"
# )
```

---

docx_set_character_style

Add character style in a Word document

Description

The function lets you add or modify Word character styles.

Usage

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

Arguments

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

Examples

```
library(officer)
doc <- read_docx()

doc <- docx_set_character_style(
  doc,
  style_id = "newcharstyle",
  style_name = "label for char style",
  base_on = "Default Paragraph Font",
  fp_text_lite(
    shading.color = "red",
    color = "white")
)
paragraph <- fpar(
  run_wordtext("hello",
    style_id = "newcharstyle"))

doc <- body_add_fpar(doc, value = paragraph)
docx_file <- print(doc, target = tempfile(fileext = ".docx"))
docx_file
```

---

docx_set_paragraph_style

Add or replace paragraph style in a Word document

Description

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

Usage

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

Arguments

x	an rdocx object
style_id	a unique style identifier for Word.
style_name	a unique label associated with the style identifier. This label is the name of the style when Word edit the document.
base_on	the style name used as base style
fp_p	paragraph formatting properties, see fp_par().
fp_t	default text formatting properties. This is used as text formatting properties, see fp_text(). If NULL (default), the paragraph will used the default text formatting properties (defined by the base_on argument).

Examples

```
library(officer)

doc <- read_docx()

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

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

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

---

docx_set_settings *Set 'Microsoft Word' Document Settings*

Description

Set various settings of a 'Microsoft Word' document generated with 'officer'. Options include:

- zoom factor (default view in Word),
- default tab stop,
- hyphenation zone,
- decimal symbol,

- list separator (see details below),
- compatibility mode,
- even and odd headers management (see details below),
- and auto hyphenation activation.

Usage

```
docx_set_settings(
  x,
  zoom = 1,
  default_tab_stop = 0.5,
  hyphenation_zone = 0.25,
  decimal_symbol = ".",
  list_separator = ";",
  compatibility_mode = "15",
  even_and_odd_headers = FALSE,
  auto_hyphenation = FALSE,
  unit = "in"
)
```

Arguments

x	an rdocx object
zoom	zoom factor, default is 1 (100%)
default_tab_stop	default tab stop in inches, default is 0.5
hyphenation_zone	hyphenation zone in inches, default is 0.25
decimal_symbol	decimal symbol, default is "."
list_separator	list separator, default is ";". Sets the separator used by Word for lists (see details).
compatibility_mode	compatibility mode, default is "15"
even_and_odd_headers	whether to use different headers for even and odd pages, default is FALSE. Enables the "Different Odd and Even Pages" feature in 'Microsoft Word'.
auto_hyphenation	whether to enable auto hyphenation, default is FALSE.
unit	unit for default_tab_stop and hyphenation_zone, one of "in", "cm", "mm".

Details

- even_and_odd_headers: If TRUE, 'Microsoft Word' will use different headers for odd and even pages ("Different Odd & Even Pages" feature in Word). This is useful for professional documents or reports that require alternating page layouts.

- `list_separator`: Sets the character used by 'Microsoft Word' to separate items in lists (for example, when inserting tables or lists in Word). This parameter affects how 'Microsoft Word' handles data import/export (CSV, etc.) and can be adapted to language or local conventions (e.g., ";" for French, "," for English).

See Also

`read_docx()`

Examples

```
library(officer)

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

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

ps <- prop_section(
  header_default = header_default,
  footer_default = footer_default,
  header_first = header_first,
  footer_first = footer_first,
  header_even = header_even,
  footer_even = footer_even
)

x <- read_docx()

x <- docx_set_settings(
  x = x,
  zoom = 2,
  list_separator = ",",
  even_and_odd_headers = TRUE
)

for (i in 1:20) {
  x <- body_add_par(x, value = txt_lorem)
}
x <- body_set_default_section(
  x,
  value = ps
)
print(x, target = tempfile(fileext = ".docx"))
```

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()`

docx_summary *Get Word content in a data.frame*

Description

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

Usage

```
docx_summary(x, preserve = FALSE, remove_fields = FALSE, detailed = FALSE)
```

Arguments

x	an rdocx object
preserve	If FALSE (default), text in table cells is collapsed into a single line. If TRUE, line breaks in table cells are preserved as a "\n" character. This feature is adapted from `docxtractr::docx_extract_tbl()` published under a MIT licensed in the 'docxtractr' package by Bob Rudis.
remove_fields	if TRUE, prevent field codes from appearing in the returned data.frame.
detailed	Should run-level information be included in the dataframe? Defaults to FALSE. If TRUE, the dataframe contains detailed information about each run (text formatting, images, hyperlinks, etc.) instead of collapsing content at the paragraph level. When FALSE, run-level information such as images, hyperlinks, and text formatting is not available since data is aggregated at the paragraph level.

Value

A data.frame with the following columns depending on the value of `detailed`:

When `detailed = FALSE` (default), the data.frame contains:

- `doc_index`: Document element index (integer).
- `content_type`: Type of content: "paragraph" or "table cell" (character).
- `style_name`: Name of the paragraph style (character).
- `text`: Collapsed text content of the paragraph or cell (character).
- `table_index`: Index of the table (integer). `NA` for non-table content.
- `row_id`: Row position in table (integer). `NA` for non-table content.
- `cell_id`: Cell position in table row (integer). `NA` for non-table content.
- `is_header`: Whether the row is a table header (logical). `NA` for non-table content.
- `row_span`: Number of rows spanned by the cell (integer). `0` for merged cells. `NA` for non-table content.
- `col_span`: Number of columns spanned by the cell (character). `NA` for non-table content.
- `table_stylename`: Name of the table style (character). `NA` for non-table content.

When `detailed = TRUE`, the data.frame contains additional run-level information:

- `run_index`: Index of the run within the paragraph (integer).
- `run_content_index`: Index of content element within the run (integer).
- `run_content_text`: Text content of the run element (character).
- `image_path`: Path to embedded image stored in the temporary directory associated with the rdocx object (character). Images should be copied to a permanent location before closing the R session if needed.
- `field_code`: Field code content (character).
- `footnote_text`: Footnote text content (character).
- `link`: Hyperlink URL (character).
- `link_to_bookmark`: Internal bookmark anchor name for hyperlinks (character).
- `bookmark_start`: Names of the bookmarks starting on this paragraph (values are concatenated with '|').
- `character_stylename`: Name of the character/run style (character).
- `sz`: Font size in half-points (integer).
- `sz_cs`: Complex script font size in half-points (integer).
- `font_family_ascii`: Font family for ASCII characters (character).
- `font_family_eastasia`: Font family for East Asian characters (character).
- `font_family_hansi`: Font family for high ANSI characters (character).
- `font_family_cs`: Font family for complex script characters (character).
- `bold`: Whether the run is bold (logical).
- `italic`: Whether the run is italic (logical).

- underline: Whether the run is underlined (logical).
- color: Text color in hexadecimal format (character).
- shading: Shading pattern (character).
- shading_color: Shading foreground color (character).
- shading_fill: Shading background fill color (character).
- keep_with_next: Whether paragraph should stay with next (logical).
- align: Paragraph alignment (character).
- level: Numbering level (integer). NA if not a numbered list.
- num_id: Numbering definition ID (integer). NA if not a numbered list.

Note

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

Examples

```
library(officer)

example_docx <- system.file(
  package = "officer",
  "doc_examples/example.docx"
)
doc <- read_docx(example_docx)

docx_summary(doc)

docx_summary(doc, detailed = TRUE)
```

---

doc_properties *Read document properties*

Description

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

Usage

```
doc_properties(x)
```

Arguments

x	an rdocx or rpptx object

Value

a data.frame

See Also

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

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

Examples

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

---

empty_content *Empty block for 'PowerPoint'*

Description

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

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

Usage

```
empty_content()
```

See Also

ph_with(), body_add_blocks()

Examples

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

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

print(doc, target = fileout )
```

`external_img` *External image*

Description

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

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

Usage

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

Arguments

`src`	image file path
`width, height`	size of the image file. It can be ignored if parameter `guess_size=TRUE`, see parameter `guess_size`.
`unit`	unit for width and height, one of "in", "cm", "mm".
`guess_size`	If package 'magick' is installed, this option can be used (set it to `TRUE`). The images will be read and width and height will be guessed.
`alt`	alternative text for images

usage

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

See Also

ph_with, body_add, fpar

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

Examples

```
# 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, "Title and Content")
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"))
```

---

floating_external_img *Floating external image*

Description

Wraps an image in an object that can be embedded as a floating image in a 'Word' document. Unlike external_img(), which creates inline images, this function creates floating images that can be positioned anywhere on the page and allow text wrapping around them.

Usage

```
floating_external_img(
  src,
  width = 0.5,
  height = 0.2,
  pos_x = 0,
  pos_y = 0,
  pos_h_from = "margin",
  pos_v_from = "margin",
  wrap_type = "square",
  wrap_side = "bothSides",
  wrap_dist_top = 0,
```

```
  wrap_dist_bottom = 0,
  wrap_dist_left = 0.125,
  wrap_dist_right = 0.125,
  unit = "in",
  guess_size = FALSE,
  alt = ""
)
```

Arguments

src	image file path
width, height	size of the image file. It can be ignored if parameter guess_size=TRUE, see parameter guess_size.
pos_x, pos_y	horizontal and vertical position of the image relative to the anchor point
pos_h_from	horizontal positioning reference point, one of "margin", "page", "column", "character"
pos_v_from	vertical positioning reference point, one of "margin", "page", "paragraph", "line"
wrap_type	text wrapping type, one of "square", "topAndBottom", "through", "tight", "none"
wrap_side	which side text wraps around, one of "bothSides", "left", "right", "largest"
wrap_dist_top, wrap_dist_bottom, wrap_dist_left, wrap_dist_right	distance between image and text (in inches)
unit	unit for width, height, pos_x and pos_y, one of "in", "cm", "mm".
guess_size	If package 'magick' is installed, this option can be used (set it to TRUE). The images will be read and width and height will be guessed.
alt	alternative text for images

usage

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

See Also

external_img, body_add, fpar, rtf_doc, rtf_add

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

Examples

```
library(officer)
srcfile <- file.path(R.home("doc"), "html", "logo.jpg")
floatimg <- floating_external_img(
  src = srcfile, height = 1.06 / 2, width = 1.39 / 2,
  pos_x = 0, pos_y = 0,
```

```
  pos_h_from = "margin", pos_v_from = "margin"
)

text <- paste0(
  " is a floating image in a ",
  paste0(rep("very ", 30), collapse = ""),
  " long text!"
)

# docx example ----
x <- read_docx()
fp_t <- fp_text(font.size = 20, color = "red")
an_fpar <- fpar(floatimg, ftext(text, fp_t))
x <- body_add_fpar(x, an_fpar)
print(x, target = tempfile(fileext = ".docx"))

# rtf example ----
rtf_doc <- rtf_doc()
rtf_doc <- rtf_add(rtf_doc, an_fpar)
print(rtf_doc, target = tempfile(fileext = ".rtf"))
```

---

fpar *Formatted paragraph*

Description

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

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

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

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

Usage

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

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

Arguments

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

See Also

block_list(), body_add_fpar(), ph_with()

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

Examples

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

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

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

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

---

fp_border *Border properties object*

Description

create a border properties object.

Usage

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

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

Arguments

color	border color - single character value (e.g. "#000000" or "black")
style	border style - single character value : See Details for supported border styles.
width	border width - an integer value : 0>= value
object	fp_border object
...	further arguments - not used

Details

For Word output the following border styles are supported:

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

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

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

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

See Also

Other functions for defining formatting properties: `fp_cell()`, `fp_par()`, `fp_tab()`, `fp_tabs()`, `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",
  rowspan = 1,
  colspan = 1
)

## 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,
  border.right,
  vertical.align,
  margin = 0,
  margin.bottom,
  margin.top,
  margin.left,
  margin.right,
  background.color,
  text.direction,
  rowspan = 1,
  colspan = 1,
  ...
)
```

Arguments

`border`	shortcut for all borders.
`border.bottom`, `border.left`, `border.top`, `border.right`	`fp_border()` for borders.
`vertical.align`	cell content vertical alignment - a single character value, expected value is one of "center" or "top" or "bottom"
`margin`	shortcut for all margins.
`margin.bottom`, `margin.top`, `margin.left`, `margin.right`	cell margins - 0 or positive integer value.
`background.color`	cell background color - a single character value specifying a valid color (e.g. "#000000" or "black").

text.direction	cell text rotation - a single character value, expected value is one of "lrtb", "tbrl", "btlr".
rowspan	specify how many rows the cell is spanned over
colspan	specify how many columns the cell is spanned over
x, object	fp_cell object
type	output type - one of 'wml', 'pml', 'html', 'rtf'.
...	further arguments - not used

See Also

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

Examples

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

---

fp_par *Paragraph formatting properties*

Description

Create a fp_par object that describes paragraph formatting properties.

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

Usage

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

```
  hanging = NA,
  word_style = "Normal"
)

fp_par_lite(
  text.align = NA,
  padding = NA,
  line_spacing = NA,
  border = FALSE,
  padding.bottom = NA,
  padding.top = NA,
  padding.left = NA,
  padding.right = NA,
  border.bottom = FALSE,
  border.left = FALSE,
  border.top = FALSE,
  border.right = FALSE,
  shading.color = NA,
  keep_with_next = NA,
  tabs = FALSE,
  first_line = NA,
  hanging = NA,
  word_style = NA
)

## 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,
  keep_with_next,
  first_line,
  hanging,
  word_style,
  ...
```

)

Arguments

text.align	text alignment - a single character value, expected value is one of 'left', 'right', 'center', 'justify'.
padding	paragraph paddings - 0 or positive integer value. Argument padding overwrites arguments padding.bottom, padding.top, padding.left, padding.right.
line_spacing	line spacing, 1 is single line spacing, 2 is double line spacing.
border	shortcut for all borders.
padding.bottom, padding.top, padding.left, padding.right	paragraph paddings - 0 or positive integer value.
border.bottom, border.left, border.top, border.right	fp_border() for borders. overwrite other border properties.
shading.color	shading color - a single character value specifying a valid color (e.g. "#000000" or "black").
keep_with_next	a scalar logical. Specifies that the paragraph (or at least part of it) should be rendered on the same page as the next paragraph when possible.
tabs	NULL (default) for no tabulation marks setting or an object returned by fp_tabs(). Note this can only have effect with Word or RTF outputs.
first_line	first-line indent in points (positive moves the first line to the right). NA (default) leaves the first-line indent unset. Mutually exclusive with hanging – if both are provided, hanging wins.
hanging	hanging indent in points (positive moves the first line to the left relative to the following lines). NA (default) leaves the hanging indent unset.
word_style	Word paragraph style name
x, object	fp_par object
...	further arguments - not used

Value

a fp_par object

See Also

fpar

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

Examples

```
fp_par(text.align = "center", padding = 5)
fp_par(padding.left = 40, hanging = 20)
obj <- fp_par(text.align = "center", padding = 1)
update(obj, padding.bottom = 5)
```

fp_tab *Tabulation mark properties object*

Description

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

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

Usage

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

Arguments

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

See Also

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

Examples

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

fp_tabs *Tabs properties object*

Description

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

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

Usage

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

Arguments

`...`	fp_tab objects

See Also

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

Examples

```
z <- fp_tabs(
  fp_tab(pos = 0.4, style = "decimal"),
  fp_tab(pos = 1, style = "decimal")
)
fpar(
  run_tab(), ftext("88."),
  run_tab(), ftext("987.45"),
  fp_p = fp_par(
    tabs = z
  )
)
```

---

fp_text *Text formatting properties*

Description

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

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

Usage

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

fp_text_lite(
  color = NA,
  font.size = NA,
```

```
  font.family = NA,
  cs.family = NA,
  eastasia.family = NA,
  hansi.family = NA,
  bold = NA,
  italic = NA,
  underlined = NA,
  strike = NA,
  vertical.align = "baseline",
  shading.color = NA
)

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

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

## S3 method for class 'fp_text'
update(
  object,
  color,
  font.size,
  bold,
  italic,
  underlined,
  strike,
  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
strike	is strikethrough
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', 'rtf'.
...	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(), fp_tab(), fp_tabs()

Examples

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

---

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

```
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(), floating_external_img(), hyperlink_ftext(), run_autonum(), run_bookmark(), run_columnbreak(), run_comment(), run_footnote(), run_footnoteref(), run_linebreak(), run_pagebreak(), run_reference(), run_tab(), run_word_field(), run_wordtext()

Examples

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

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

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

---

hyperlink_ftext *Formatted chunk of text with hyperlink*

Description

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

Usage

```
hyperlink_ftext(text, prop = NULL, href)
```

Arguments

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

usage

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

See Also

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

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_dedupe_ph_labels

Detect and handle duplicate placeholder labels

Description

PowerPoint does not enforce unique placeholder labels in a layout. Selecting a placeholder via its label using ph_location_label will throw an error, if the label is not unique. layout_dedupe_ph_labels helps to detect, rename, or delete duplicate placholder labels.

Usage

```
layout_dedupe_ph_labels(x, action = "detect", print_info = FALSE)
```

Arguments

x	An rpptx object.
action	Action to perform on duplicate placeholder labels. One of: • detect (default) = show info on dupes only, make no changes

- rename = create unique labels. Labels are renamed by appending a sequential number separated by dot to duplicate labels. For example, c("title", "title") becomes c("title.1", "title.2").
- delete = only keep one of the placeholders with a duplicate label

print_info — Print action information (e.g. renamed placeholders) to console? Default is FALSE. Always TRUE for action detect.

Value

A rpptx object (with modified placeholder labels).

Examples

```
x <- read_pptx()
layout_dedupe_ph_labels(x)

file <- system.file("doc_examples", "ph_dupes.pptx", package = "officer")
x <- read_pptx(file)
layout_dedupe_ph_labels(x)
layout_dedupe_ph_labels(x, "rename", print_info = TRUE)
```

---

layout_default *Default layout for new slides*

Description

Set or remove the default layout used when calling add_slide().

Usage

```
layout_default(x, layout = NULL, master = NULL, as_list = FALSE)
```

Arguments

x	An rpptx object.
layout	Layout name. If NULL (default), removes the default layout.
master	Name of master. Only required if layout name is not unique across masters.
as_list	If TRUE, return a list with layout and master instead of the rpptx object.

Value

The rpptx object.

See Also

add_slide()

Examples

```
# set and remove the default layout
x <- read_pptx()
layout_default(x) # no defaults
x <- layout_default(x, "Title and Content") # set default
layout_default(x)
x <- add_slide(x) # new slide with default layout
x <- layout_default(x, NULL) # remove default
layout_default(x) # no defaults

# use when repeatedly adding slides with same layout
x <- read_pptx()
x <- layout_default(x, "Title and Content")
x <- add_slide(x, title = "1. Slide", body = "Some content")
x <- add_slide(x, title = "2. Slide", body = "Some more content")
x <- add_slide(x, title = "3. Slide", body = "Even more content")
```

---

layout_properties *Slide layout properties*

Description

Detailed information about the placeholders on the slide layouts (label, position, etc.). See *Value* section below for more info.

Usage

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

Arguments

x	an rpptx object
layout	slide layout name. If NULL, returns all layouts.
master	master layout name where layout is located. If NULL, returns all masters.

Value

Returns a data frame with one row per placeholder and the following columns:

- master_name: Name of master (a .pptx file may have more than one)
- name: Name of layout
- type: Placeholder type
- type_idx: Running index for phs of the same type. Ordering by ph position (top -> bottom, left -> right)
- id: A unique placeholder id (assigned by PowerPoint automatically, starts at 2, potentially non-consecutive)

- ph_label: Placeholder label (can be set by the user in PowerPoint)
- ph: Placholder XML fragment (usually not needed)
- offx,offy: placeholder's distance from left and top edge (in inch)
- cx,cy: width and height of placeholder (in inch)
- rotation: rotation in degrees
- fld_id is generally stored as a hexadecimal or GUID value
- fld_type: a unique identifier for a particular field

See Also

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

Examples

```
library(officer)

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_rename_ph_labels

Change ph labels in a layout

Description

There are two versions of the function. The first takes a set of key-value pairs to rename the ph labels. The second uses a right hand side (rhs) assignment to specify the new ph labels. See section *Details*.

NB: You can also rename ph labels directly in PowerPoint. Open the master template view (Alt + F10) and go to Home > Arrange > Selection Pane.

Usage

```
layout_rename_ph_labels(x, layout, master = NULL, ..., .dots = NULL)

layout_rename_ph_labels(x, layout, master = NULL, id = NULL) <- value
```

Arguments

x	An rpptx object.
layout	Layout name or index. Index is the row index of layout_summary().
master	Name of master. Only required if the layout name is not unique across masters.
...	Comma separated list of key-value pairs to rename phs. Either reference a ph via its label ("old label" = "new label") or its unique id ("id" = "new label").
.dots	Provide a named list or vector of key-value pairs to rename phs (list("old label" = "new label").
id	Unique placeholder id (see column id in layout_properties() or plot_layout_properties()).
value	Not relevant for user. A pure technical necessity for rhs assignments.

Details

- Note the difference between the terms id and index. Both can be found in the output of layout_properties(). The unique ph id is found in column id. The index refers to the index of the data frame row.
- In a right hand side (rhs) label assignment (<- new_labels), there are two ways to optionally specify a subset of phs to rename. In both cases, the length of the rhs vector (the new labels) must match the length of the id or index:
  1. use the id argument to specify ph ids to rename: layout_rename_ph_labels(..., id = 2:3) <- new_labels
  2. use an index in squared brackets: layout_rename_ph_labels(...)[1:2] <- new_labels

Value

Vector of renamed ph labels.

Examples

```
x <- read_pptx()

# INFO -------------

# Returns layout's ph_labels by default in same order as layout_properties()
layout_rename_ph_labels(x, "Comparison")
layout_properties(x, "Comparison")$ph_label


# BASICS -----------
#
# HINT: run `plot_layout_properties(x, "Comparison")` to see how labels change

# rename using key-value pairs: 'old label' = 'new label' or 'id' = 'new label'
layout_rename_ph_labels(x, "Comparison", "Title 1" = "LABEL MATCHED") # label matching
layout_rename_ph_labels(x, "Comparison", "3" = "ID MATCHED") # id matching
layout_rename_ph_labels(
  x,
  "Comparison",
```

```
  "Date Placeholder 6" = "DATE",
  "8" = "FOOTER"
) # label, id

# rename using a named list and the .dots arg
renames <- list("Content Placeholder 3" = "CONTENT_1", "6" = "CONTENT_2")
layout_rename_ph_labels(x, "Comparison", .dots = renames)

# rename via rhs assignment and optional index (not id!)
layout_rename_ph_labels(x, "Comparison") <- LETTERS[1:8]
layout_rename_ph_labels(x, "Comparison")[1:3] <- paste("CHANGED", 1:3)

# rename via rhs assignment and ph id (not index)
layout_rename_ph_labels(x, "Comparison", id = c(2, 4)) <- paste("ID =", c(2, 4))


# MORE ------------

# make all labels lower case
labels <- layout_rename_ph_labels(x, "Comparison")
layout_rename_ph_labels(x, "Comparison") <- tolower(labels)

# rename all labels to type [type_idx]
lp <- layout_properties(x, "Comparison")
layout_rename_ph_labels(x, "Comparison") <- paste0(
  lp$type,
  " [",
  lp$type_idx,
  "]"
)

# rename duplicated placeholders (see also `layout_dedupe_ph_labels()`)
file <- system.file("doc_examples", "ph_dupes.pptx", package = "officer")
x <- read_pptx(file)
lp <- layout_properties(x, "2-dupes")
idx <- which(lp$ph_label == "Content 7") # exists twice
layout_rename_ph_labels(x, "2-dupes")[idx] <- paste("DUPLICATE", seq_along(idx))

# warning: in case of duped labels only the first occurrence is renamed
x <- read_pptx(file)
layout_rename_ph_labels(x, "2-dupes", "Content 7" = "new label")
```

`layout_summary` *Presentation layouts summary*

Description

Get information 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 information: annotate_base(), color_scheme(), doc_properties(), 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)
```

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

length.rpptx *Number of slides*

Description

Function length will return the number of slides.

Usage

```
## S3 method for class 'rpptx'
length(x)
```

Arguments

x	an rpptx object

See Also

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

Examples

```
my_pres <- read_pptx()
my_pres <- add_slide(my_pres, "Title and Content")
my_pres <- add_slide(my_pres, "Title and Content")
length(my_pres)
```

list_item *Create a list item*

Description

Wrap an fpar() with a hierarchy level for use inside block_list_items().

Usage

```
list_item(x, level = 1L)
```

Arguments

x	an fpar() object or a character string (automatically converted to fpar())
level	hierarchy level, integer starting at 1

See Also

block_list_items()

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

---

media_extract *Extract media from a document object*

Description

Extract files from a rpptx object.

Usage

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

Arguments

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

Examples

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

---

move_slide *Move a slide*

Description

Move a slide in a pptx presentation.

Usage

```
move_slide(x, index = NULL, to)
```

Arguments

x	an rpptx object
index	slide index or a vector of slide indices to remove, default to current slide position.
to	new slide index.

Note

cursor is set on the last slide.

See Also

read_pptx()

Other functions to manipulate slides: add_slide(), on_slide(), remove_slide(), set_notes()

Examples

```
library(officer)

x <- read_pptx()
x <- add_slide(x, "Title and Content")
x <- ph_with(x, "Hello world 1", location = ph_location_type())
x <- add_slide(x, "Title and Content")
x <- ph_with(x, "Hello world 2", location = ph_location_type())
x <- move_slide(x, index = 1, to = 2)
```

---

notes_location_label *Location of a named placeholder for notes*

Description

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

Usage

```
notes_location_label(ph_label, ...)
```

Arguments

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

notes_location_type *Location of a placeholder for notes*

Description

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

Usage

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

Arguments

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

officer *Manipulate Microsoft Word and PowerPoint Documents with 'officer'*

Description

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

Examples of usage are:

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

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

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

Get Word content in a data.frame

While officer allows you to generate Word and PowerPoint documents, an important feature is also the ability to read the content of existing Word documents. Use `docx_summary()` to extract document content as a structured data.frame, making it easy to analyze and process Word files programmatically.

Copy an officer object

'officer' objects of class `rdocx` or `rpptx` use R6 classes with reference semantics. **Assignment does NOT create a copy**:

```
pptx1 <- read_pptx()
pptx2 <- pptx1  # pptx2 is a reference to pptx1, not a copy!
```

If you need independent documents (e.g., in loops), read the template each time:

```
for (i in 1:10) {
  doc <- read_docx("template.docx")  # Read fresh each iteration
  # ... modify doc ...
  print(doc, target = paste0("output_", i, ".docx"))
}
```

Author(s)

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

Authors:

- David Gohel <david.gohel@ardata.fr>
- Stefan Moog <moogs@gmx.de>
- Mark Heckmann <heckmann.mark@gmail.com> (ORCID)

Other contributors:

- ArData [copyright holder]
- Frank Hangler <frank@plotandscatter.com> (function body_replace_all_text) [contributor]
- Liz Sander <lsander@civisanalytics.com> (several documentation fixes) [contributor]
- Anton Victorson <anton@victorson.se> (fixes xml structures) [contributor]
- Jon Calder <jonmcalder@gmail.com> (update vignettes) [contributor]
- John Harrold <john.m.harrold@gmail.com> (function annotate_base) [contributor]
- John Muschelli <muschellij2@gmail.com> (google doc compatibility) [contributor]
- Bill Denney <wdenney@humanpredictions.com> (ORCID) (function as.matrix.rpptx) [contributor]
- Nikolai Beck <beck.nikolai@gmail.com> (set speaker notes for .pptx documents) [contributor]
- Greg Leleu <gregoire.leleu@gmail.com> (fields functionality in ppt) [contributor]
- Majid Eismann [contributor]
- Wahiduzzaman Khan (vectorization of remove_slide) [contributor]
- Hongyuan Jia <hongyuanjia@cqust.edu.cn> (ORCID) [contributor]
- Michael Stackhouse <mike.stackhouse@atorusresearch.com> [contributor]

See Also

The user documentation: https://ardata-fr.github.io/officeverse/ and manuals https://davidgohel.github.io/officer/

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 to manipulate slides: add_slide(), move_slide(), remove_slide(), set_notes()

Examples

```
library(officer)

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

open_file *Opens a file locally*

Description

Opening a file locally requires a compatible application to be installed (e.g., MS Office or LibreOffice for .pptx or .docx files).

Usage

```
open_file(path)
```

Arguments

path	File path.

Details

NB: Function is a small wrapper around `utils::browseURL()` to have a more suitable function name.

Examples

```
x <- read_pptx()
x <- add_slide(x, "Title Slide", ctrTitle = "My Title")
file <- print(x, tempfile(fileext = ".pptx"))
## Not run:
open_file(file)
## End(Not run)
```

page_mar *Page margins object*

Description

Define margins for each page of a section.

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

Usage

```
page_mar(
  bottom = 1417/1440,
  top = 1417/1440,
  right = 1417/1440,
  left = 1417/1440,
  header = 708/1440,
  footer = 708/1440,
  gutter = 0/1440
)
```

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_mar()
```

---

page_size *Page size object*

Description

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

Usage

```
page_size(
  width = 11906/1440,
  height = 16838/1440,
  orient = "portrait",
  unit = "in"
)
```

Arguments

width, height	page width, page height, default to A4 format If NULL the value will be ignored and Word will use the default value.
orient	page orientation, either 'landscape', either 'portrait'.
unit	unit for width and height, one of "in", "cm", "mm".

See Also

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

Examples

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

---

phs_with *Fill multiple placeholders using key value syntax*

Description

A sibling of ph_with that fills mutiple placeholders at once. Placeholder locations are specfied using the short form syntax. The location and corresponding object are passed as key value pairs (phs_with("short form location" = object)). Under the hood, ph_with is called for each pair. Note that phs_with does not cover all options from the ph_location_* family and is also less customization. It is a covenience wrapper for the most common use cases. The implemented short forms are listed in section "Short forms".

Usage

```
phs_with(x, ..., .dots = NULL, .slide_idx = NULL)
```

Arguments

x	A rpptx object.
...	Key-value pairs of the form "short form location" = object. If the short form is an integer or a string with blanks, you must wrap it in quotes or backticks.
.dots	List of key-value pairs "short form location" = object. Alternative to ....
.slide_idx	Numeric indexes of slides to process. NULL (default) processes the current slide only. Use keyword all for all slides.

Short forms

The following short forms are implemented and can be used as the parameter in the function call. The corresponding function from the `ph_location_*` family (called under the hood) is displayed on the right.

Short form	Description	Location function
`"left"`	Keyword string	`ph_location_left()`
`"right"`	Keyword string	`ph_location_right()`
`"fullsize"`	Keyword string	`ph_location_fullsize()`
`"body [1]"`	String: type + index in brackets (1 if omitted)	`ph_location_type("body", 1)`
`"my_label"`	Any string not matching a keyword or type	`ph_location_label("my_label")`
1	Length 1 integer	`ph_location_id(1)`

See Also

`ph_with()`, `add_slide()`

Examples

```
library(officer)

# use key-value format to fill phs
x <- read_pptx()
x <- add_slide(x, "Two Content")
x <- phs_with(
  x,
  `Title 1` = "A title", # ph label
  dt = Sys.Date(), # ph type
  `body[2]` = "Body 2", # ph type + type index
  left = "Left side", # ph keyword
  `6` = "Footer" # ph index
)

# reuse ph content via the .dots arg
x <- read_pptx()
my_ph_list <- list(`6` = "Footer", dt = Sys.Date())
x <- add_slide(x, "Two Content")
x <- phs_with(
  x,
  `Title 1` = "Title A",
  `body[2]` = "Body A",
  .dots = my_ph_list
)
x <- add_slide(x, "Two Content")
x <- phs_with(
  x,
  `Title 1` = "Title B",
  `body[2]` = "Body B",
  .dots = my_ph_list
)
```

```
# use the .slide_idx arg to select which slide(s) to process
x <- read_pptx()
x <- add_slide(x, "Two Content")
x <- add_slide(x, "Two Content")
x <- phs_with(x, `6` = "Footer", dt = Sys.Date(), .slide_idx = 1:2)

# run to open temp pptx file locally
# \dontrun{
# print(x, preview = TRUE)
# }
```

---

ph_hyperlink *Hyperlink a placeholder*

Description

Add hyperlink to a placeholder in the current slide.

Usage

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

Arguments

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

See Also

`ph_with()`

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

Examples

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

print(doc, target = fileout)
```

---

ph_location *Location for a placeholder from scratch*

Description

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

Usage

```
ph_location(
  left = 1,
  top = 1,
  width = 4,
  height = 3,
  newlabel = "",
  bg = NULL,
  rotation = NULL,
  ln = NULL,
  geom = 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
`ln`	a `sp_line()` object specifying the outline style.
`geom`	shape geometry, see http://www.datypic.com/sc/ooxml/t-a_ST_ShapeType.html
`...`	unused arguments

Details

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

left left coordinate of the bounding box

top top coordinate of the bounding box

width width of the bounding box

height height of the bounding box

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

See Also

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

Examples

```
library(officer)

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

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

ph_location_fullsize *Location of a full size element*

Description

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

Usage

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

Arguments

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

See Also

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

Examples

```
library(officer)

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

ph_location_id *Location of a placeholder based on its id*

Description

Each placeholder has an id (a low integer value). The ids are unique across a single layout. The function uses the placeholder's id to reference it. Different from a ph label, the id is auto-assigned by PowerPoint and cannot be modified by the user. Use layout_properties() (column id) and plot_layout_properties() (upper right corner, in green) to find a placeholder's id.

Usage

```
ph_location_id(id, newlabel = NULL, ...)
```

Arguments

id	placeholder id.
newlabel	a new label to associate with the placeholder.
...	not used.

Details

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

left left coordinate of the bounding box

top top coordinate of the bounding box

width width of the bounding box

height height of the bounding box

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

See Also

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

Examples

```
library(officer)

doc <- read_pptx()
doc <- add_slide(doc, "Comparison")
plot_layout_properties(doc, "Comparison")

doc <- ph_with(doc, "The Title", location = ph_location_id(id = 2)) # title
doc <- ph_with(doc, "Left Header", location = ph_location_id(id = 3)) # left header
doc <- ph_with(doc, "Left Content", location = ph_location_id(id = 4)) # left content
doc <- ph_with(doc, "The Footer", location = ph_location_id(id = 8)) # footer

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

## file.show(file) # may not work on your system
```

ph_location_label *Location of a named placeholder*

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(), ph_location_fullsize(), ph_location_id(), ph_location_left(), ph_location_right(), ph_location_template(), ph_location_type()

Examples

```
library(officer)

# ph_location_label demo ----

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

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

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

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

`ph_location_left` *Location of a left body element*

Description

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

Usage

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

Arguments

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

See Also

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

Examples

```
library(officer)

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

---

ph_location_right *Location of a right body element*

Description

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

Usage

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

Arguments

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

See Also

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

Examples

```
library(officer)

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

ph_location_template *Location for a placeholder based on a template*

Description

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

Usage

```
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', 'ftr', 'sldNum'. It will be used as a template placeholder.
`id`	index of the placeholder template. If two body placeholder, there can be two different index: 1 and 2 for the first and second body placeholders defined in the layout.
`...`	unused arguments

Details

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

left left coordinate of the bounding box

top top coordinate of the bounding box

width width of the bounding box

height height of the bounding box

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

See Also

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

Examples

```
library(officer)

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

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

Arguments

type	placeholder type to look for in the slide layout, one of 'body', 'title', 'ctrTitle', 'subTitle', 'dt', 'ftr', 'sldNum'.
type_idx	Type index of the placeholder. If there is more than one placeholder of a type (e.g., body), the type index can be supplied to uniquely identify a ph. The index is a running number starting at 1. It is assigned by placeholder position (top -> bottom, left -> right). See plot_layout_properties() for details. If idx argument is used, position_right and position_top are ignored.

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	(**DEPRECATED, use** type_idx **instead**) 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

Details

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

left left coordinate of the bounding box

top top coordinate of the bounding box

width width of the bounding box

height height of the bounding box

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

See Also

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

Examples

```
library(officer)

# 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, "Title and Content")
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")
doc <- ph_with(x = doc, "Un sous titre", location = loc_subtitle)
doc <- ph_with(x = doc, "Un titre", location = loc_ctrtitle)

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

`ph_remove` *Remove a shape*

Description

Remove a shape in a slide.

Usage

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

Arguments

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

See Also

`ph_with()`

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

Examples

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

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

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

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

print(doc, target = fileout)
```

---

ph_slidelink *Slide link to a placeholder*

Description

Add slide link to a placeholder in the current slide.

Usage

```
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()`. If used, `type` and `id` are ignored.
slide_index	slide index to reach

See Also

`ph_with()`

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

Examples

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

print(doc, target = fileout)
```

---

ph_with *Add objects on the current slide*

Description

add an 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

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

ph_with.character(x, value, location, ...)

ph_with.numeric(x, value, location, format_fun = format, ...)

ph_with.factor(x, value, location, ...)

ph_with.logical(x, value, location, format_fun = format, ...)

ph_with.Date(x, value, location, date_format = NULL, ...)

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

ph_with.unordered_list(x, value, location, ...)

ph_with.block_list_items(x, value, location, ...)

ph_with.data.frame(
  x,
  value,
  location,
  header = TRUE,
  tcf = table_conditional_formatting(),
  alignment = NULL,
  ...
)

ph_with.gg(x, value, location, res = 300, alt_text = "", scale = 1, ...)

ph_with.plot_instr(x, value, location, res = 300, ...)

ph_with.external_img(x, value, location, use_loc_size = TRUE, ...)

ph_with.fpar(x, value, location, ...)

ph_with.empty_content(x, value, location, ...)

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

Arguments

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

location	a placeholder location object or a location short form. 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"). In ph_with(), several location short forms can be used, as listed in section "Short forms".
...	further arguments passed to or from other methods. When adding a ggplot object or plot_instr, these arguments will be used by the png function.
format_fun	format function for non character vectors
date_format	A format string for dates (default "%Y-%m-%d"). See format arg in strftime() for details. Set a global default via options(officer.date_format = ...).
level_list	The list of levels for hierarchy structure as integer values. If used the object is formated 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. Defaults to "". If "" or NULL an alt text added with ggplot2::labs(alt = ...) will be used if any.
scale	Multiplicative scaling factor, same as in ggsave
use_loc_size	if set to FALSE, external_img width and height will be used.

Functions

- ph_with.character(): add a character vector to a new shape on the current slide, values will be added as paragraphs.
- ph_with.numeric(): add a numeric vector to a new shape on the current slide, values will be be first formatted then added as paragraphs.
- ph_with.factor(): add a factor vector to a new shape on the current slide, values will be be converted as character and then added as paragraphs.
- ph_with.Date(): add a Date object vector to a new shape on the current slide, values will be be first converted to character.
- ph_with.block_list(): add a block_list() made of fpar() to a new shape on the current slide.
- ph_with.unordered_list(): add a unordered_list() made of fpar() to a new shape on the current slide.
- ph_with.block_list_items(): add a block_list_items() (bullet or numbered list) to a new shape on the current slide.
- ph_with.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.
- ph_with.gg(): add a ggplot object to a new shape on the current slide. Use package 'rvg' for more advanced graphical features.

- ph_with.plot_instr(): add an R plot to a new shape on the current slide. Use package 'rvg' for more advanced graphical features.
- ph_with.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.
- ph_with.fpar(): add an fpar() to a new shape on the current slide as a single paragraph in a block_list().
- ph_with.empty_content(): add an empty_content() to a new shape on the current slide.
- ph_with.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.

Short forms

The location argument of ph_with() either expects a location object as returned by the ph_location_* functions or a corresponding location *short form* (string or numeric):

Location function	Short form	Description
ph_location_left()	"left"	Keyword string
ph_location_right()	"right"	Keyword string
ph_location_fullsize()	"fullsize"	Keyword string
ph_location_type("body", 1)	"body [1]"	String: type + index in brackets (1 if omitted)
ph_location_label("my_label")	"my_label"	Any string not matching a keyword or type
ph_location_id(1)	1	Length 1 integer
ph_location(0, 0, 4, 5)	c(0,0,4,5)	Length 4 numeric, optionally named, c(top=0, left=0, ...)

Illustrations

See Also

Specify placeholder locations with ph_location_type, ph_location, ph_location_label, ph_location_left, ph_location_right, ph_location_fullsize, ph_location_template. phs_with is a sibling of ph_with that fills multiple placeholders at once. Use add_slide to add new slides.

Examples

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

doc_1 <- read_pptx()
sz <- slide_size(doc_1)

# add text and a table ----
doc_1 <- add_slide(doc_1, layout = "Two Content", master = "Office Theme")
```

```
doc_1 <- ph_with(
  x = doc_1,
  value = c("Table cars"),
  location = ph_location_type(type = "title")
)
doc_1 <- ph_with(
  x = doc_1,
  value = names(cars),
  location = ph_location_left()
)
doc_1 <- ph_with(
  x = doc_1,
  value = cars,
  location = ph_location_right()
)
doc_1 <- ph_with(
  x = doc_1,
  value = Sys.Date(),
  location = ph_location_type("dt")
)

# 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, "Title and Content")
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, "Title and Content")
  gg_plot <- ggplot(data = iris) +
    geom_point(
      mapping = aes(Sepal.Length, Petal.Length),
      size = 3
    ) +
    theme_minimal()
  doc_1 <- ph_with(
```

```
    x = doc_1,
    value = gg_plot,
    location = ph_location_type(type = "body"),
    bg = "transparent"
  )
  doc_1 <- ph_with(
    x = doc_1,
    value = "graphic title",
    location = ph_location_type(type = "title")
  )
}

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

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

# add a block_list ----
dummy_text <- readLines(system.file(
  package = "officer",
  "doc_examples/text.txt"
))
fp_1 <- fp_text(bold = TRUE, color = "pink", font.size = 0)
```

```
fp_2 <- fp_text(bold = TRUE, font.size = 0)
fp_3 <- fp_text(italic = TRUE, color = "red", font.size = 0)
bl <- block_list(
  fpar(ftext("hello world", fp_1)),
  fpar(
    ftext("hello", fp_2),
    ftext("hello", fp_3)
  ),
  dummy_text
)
doc_1 <- add_slide(doc_1, "Title and Content")
doc_1 <- ph_with(
  x = doc_1,
  value = bl,
  location = ph_location_type(type = "body")
)


# fpar ------
fpt <- fp_text(
  bold = TRUE,
  font.family = "Bradley Hand",
  font.size = 150,
  color = "#F5595B"
)
hw <- fpar(
  ftext("hello ", fpt),
  hyperlink_ftext(
    href = "https://cran.r-project.org/index.html",
    text = "cran",
    prop = fpt
  )
)
doc_1 <- add_slide(doc_1, "Title and Content")
doc_1 <- ph_with(
  x = doc_1,
  value = hw,
  location = ph_location_type(type = "body")
)
# unordered_list ----
ul <- unordered_list(
  level_list = c(1, 2, 2, 3, 3, 1),
  str_list = c("Level1", "Level2", "Level2", "Level3", "Level3", "Level1"),
  style = fp_text(color = "red", font.size = 0)
)
doc_1 <- add_slide(doc_1, "Title and Content")
doc_1 <- ph_with(
  x = doc_1,
  value = ul,
  location = ph_location_type()
)

print(doc_1, target = fileout)
```

```
# Example using short-form locations ----
x <- read_pptx()
x <- add_slide(x, "Title Slide")
x <- ph_with(x, "A title", "Title 1") # label
x <- ph_with(x, "A subtitle", 3) # id
x <- ph_with(x, "A left text", "left") # keyword
x <- ph_with(x, Sys.Date(), "dt[1]") # type + index
x <- ph_with(x, "More content", c(5, .5, 5, 2)) # numeric vector (left, top, width, heigh)
# \dontrun{
# print(x, preview = TRUE) # opens file locally
# }
```

`plot_instr` *Wrap plot instructions for png plotting in Powerpoint or Word*

Description

A simple wrapper to capture plot instructions that will be executed and copied in a document. It produces an object of class 'plot_instr' with a corresponding method `ph_with()` and `body_add_plot()`.

The function enable usage of any R plot with argument code. Wrap your code between curly bracket if more than a single expression.

Usage

```
plot_instr(code)
```

Arguments

code	plotting instructions

See Also

`ph_with()`, `body_add_plot()`

Other block functions for reporting: `block_caption()`, `block_gg()`, `block_list()`, `block_list_items()`, `block_pour_docx()`, `block_section()`, `block_table()`, `block_toc()`, `fpar()`, `list_item()`, `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, "Title and Content")
doc <- ph_with(
  doc, anyplot,
  location = ph_location_fullsize(),
  bg = "#00000066", pointsize = 12)
print(doc, target = tempfile(fileext = ".pptx"))
```

`plot_layout_properties` *Slide layout properties plot*

Description

Plot slide layout properties into corresponding placeholders. This can be useful to help visualize placeholders locations and identifiers. *All* information in the plot stems from the `layout_properties()` output. See *Details* section for more info.

Usage

```
plot_layout_properties(
  x,
  layout = NULL,
  master = NULL,
  slide_idx = NULL,
  labels = TRUE,
  title = TRUE,
  type = TRUE,
  id = TRUE,
  cex = c(labels = 0.5, type = 0.5, id = 0.5),
  legend = FALSE
)
```

Arguments

`x`	an `rpptx` object
`layout`	slide layout name or numeric index (row index from `layout_summary()`. If `NULL` (default), it plots the current slide's layout or the default layout (if set and there are not slides yet).
`master`	master layout name where `layout` is located. Can be omitted if layout is unambiguous.
`slide_idx`	Numeric slide index (default `NULL`) to specify which slide's layout should be plotted.
`labels`	if `TRUE` (default), adds placeholder labels (centered in *red*).

title	if TRUE (default), adds a title with the layout and master name (latter in square brackets) at the top.
type	if TRUE (default), adds the placeholder type and its index (in square brackets) in the upper left corner (in *blue*).
id	if TRUE (default), adds the placeholder's unique id (see column id from layout_properties()) in the upper right corner (in *green*).
cex	List or vector to specify font size for labels, type, and id. Default is c(labels = 0.5, type = 0.5, id = 0.5). See graphics::text() for details on how cex works. Matching by position and partial name matching is supported. A single numeric value will apply to all three parameters.
legend	Add a legend to the plot (default FALSE).

Details

The plot contains all relevant information to reference a placeholder via the ph_location_* function family:

- label: ph label (red, center) to be used in ph_location_label(). *NB*: The label can be assigned by the user in PowerPoint.
- type[idx]: ph type + type index in brackets (blue, upper left) to be used in ph_location_type(). *NB*: The index is consecutive and is sorted by ph position (top -> bottom, left -> right).
- id: ph id (green, upper right) to be used in ph_location_id() (forthcoming). *NB*: The id is set by PowerPoint automatically and lack a meaningful order.

See Also

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

Examples

```
x <- read_pptx()

# select layout explicitly
plot_layout_properties(x = x, layout = "Title Slide", master = "Office Theme")
plot_layout_properties(x = x, layout = "Title Slide") # no master needed if layout name unique
plot_layout_properties(x = x, layout = 1) # use layout index instead of name

# plot default layout if one is set
x <- layout_default(x, "Title and Content")
plot_layout_properties(x)

# plot current slide's layout (default if no layout is passed)
x <- add_slide(x, "Title Slide")
plot_layout_properties(x)

# specify which slide's layout to plot by index
plot_layout_properties(x, slide_idx = 1)
```

```
# change appearance: what to show, font size, legend etc.
plot_layout_properties(
  x,
  layout = "Two Content",
  title = FALSE,
  type = FALSE,
  id = FALSE
)
plot_layout_properties(x, layout = 4, cex = c(labels = .8, id = .7, type = .7))
plot_layout_properties(x, 1, legend = TRUE)
```

`pptx_summary` *PowerPoint content in a data.frame*

Description

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

Usage

```
pptx_summary(x, preserve = FALSE)
```

Arguments

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

Examples

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

print.rdocx *Write a 'Word' File*

Description

print.rdocx() is the essential output function for creating Word files with officer. It takes an rdocx object (created with read_docx() and populated with content) and writes it to disk as a .docx file.

This function performs all necessary post-processing operations before writing the file.

The function is typically called at the end of your document creation workflow, after all content has been added with body_add_*() functions.

Usage

```
## S3 method for class 'rdocx'
print(
  x,
  target = NULL,
  copy_header_refs = FALSE,
  copy_footer_refs = FALSE,
  preview = FALSE,
  ...
)
```

Arguments

x	an rdocx object created with read_docx()
target	path to the .docx file to write. The file will be created or overwritten if it already exists. If NULL and preview = FALSE, the function returns NULL without writing a file.
copy_header_refs, copy_footer_refs	logical, default is FALSE. If TRUE, copy the references to the header and footer in each section of the body of the document. This parameter is experimental and may change in a future version.
preview	Save x to a temporary file and open it (default FALSE). When TRUE, the document is saved to a temporary location and opened with the system's default application for .docx files, useful for quick previewing during development.
...	unused

Value

The full path to the created .docx file (invisibly). This allows chaining operations or capturing the output path for further use.

See Also

Create a 'Word' document object with `read_docx()`, add content with functions `body_add_par()`, `body_add_plot()`, `body_add_table()`, change settings with `docx_set_settings()`, set properties with `set_doc_properties()`, read 'Word' styles with `styles_info()`.

Examples

```
library(officer)

# This example demonstrates how to create
# an small document -----

## Create a new Word document
doc <- read_docx()
doc <- body_add_par(doc, "hello world")
## Save the document
output_file <- print(doc, target = tempfile(fileext = ".docx"))

# preview mode: save to temp file and open locally ----
## Not run:
# print(doc, preview = TRUE)
```

`print.rpptx` *Write a 'PowerPoint' file.*

Description

Create a 'PowerPoint' file from an `rpptx` object (created by `read_pptx()`).

Usage

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

Arguments

`x`	an `rpptx` object.
`target`	path to the .pptx file to write. If `target` is `NULL` (default), the `rpptx` object is printed to the console.
`preview`	Save `x` to a temporary file and open it (default `FALSE`).
`...`	unused.

Value

If preview is `TRUE`, returns the temp file path invisibly.

See Also

read_pptx()

Examples

```
# write an rpptx object to a .pptx file ----
file <- tempfile(fileext = ".pptx")
x <- read_pptx() # empty presentation, has no slides yet
print(x, target = file)

# preview mode: save to temp file and open locally ----
## Not run:
print(x, preview = TRUE)

## End(Not run)
```

---

print.rtf *Write an 'RTF' File*

Description

Write the RTF object and its content to a file.

Usage

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

Arguments

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

See Also

rtf_doc()

Examples

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

prop_section *Section properties*

Description

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

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

Important: When creating multiple sections in a document, it is strongly recommended to use the **same** `page_margins` **object** for all sections to avoid unwanted page breaks. Changing page margins between sections can cause Word to insert automatic page breaks, even when using `type = "continuous"`. To ensure consistent behavior, create a single `page_mar()` object and reuse it across all `prop_section()` calls. See the examples in `body_end_block_section()` which demonstrate this best practice.

Usage

```
prop_section(
  page_size = NULL,
  page_margins = NULL,
  type = "continuous",
  section_columns = NULL,
  header_default = NULL,
  header_even = NULL,
  header_first = NULL,
  footer_default = NULL,
  footer_even = NULL,
  footer_first = NULL
)
```

Arguments

`page_size`	page dimensions, an object generated with function page_size.
`page_margins`	page margins, an object generated with function page_mar. It is recommended to use the same margins object across all sections to prevent unintended page breaks.
`type`	Section type. It defines how the contents of the section will be placed relative to the previous section. Available types are "continuous" (begins the section on the next paragraph), "evenPage" (begins on the next even-numbered page), "nextColumn" (begins on the next column on the page), "nextPage" (begins on the following page), "oddPage" (begins on the next odd-numbered page).
`section_columns`	section columns, an object generated with function section_columns. Use NULL (default value) for no content.

header_default	content as a block_list() for the default page header. Use NULL (default value) for no content.
header_even	content as a block_list() for the even page header. Use NULL (default value) for no content.
header_first	content as a block_list() for the first page header. Use NULL (default value) for no content.
footer_default	content as a block_list() for the default page footer. Use NULL (default value) for no content.
footer_even	content as a block_list() for the even page footer. Use NULL (default value) for no content.
footer_first	content as a block_list() for the default page footer. Use NULL (default value) for no content.

Illustrations

See Also

block_section

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

Examples

```
library(officer)

# Example 1: Mixing different section layouts ----
# This example demonstrates how to create a document with multiple sections,
# each with different page orientations and column layouts

# Define a landscape section with single column
# This is useful for wide tables or charts
landscape_one_column <- block_section(
  prop_section(
    page_size = page_size(orient = "landscape"),
    type = "continuous"
  )
)

# Define a landscape section with two columns
# Useful for text-heavy content in landscape mode (e.g., newsletters)
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))
  )
)
```

```
# Create a new document
doc_1 <- read_docx()

# Section 1: Landscape single column for wide table ----
# Add a title for the first section
doc_1 <- body_add_par(doc_1, "Wide Table Section", style = "heading 1")
doc_1 <- body_add_par(
  doc_1,
  "This table is displayed in landscape orientation to accommodate all columns."
)

# Add a wide table with multiple columns
doc_1 <- body_add_table(doc_1, value = mtcars[1:10, ], style = "table_template")

# End the landscape single-column section
doc_1 <- body_end_block_section(doc_1, value = landscape_one_column)

# Section 2: Landscape two columns for text content ----
# Add a title for the two-column section
doc_1 <- body_add_par(doc_1, "Two-Column Text Section", style = "heading 1")
doc_1 <- body_add_par(
  doc_1,
  "The following text flows across two columns in landscape orientation."
)

# Add text content that will flow across two columns
doc_1 <- body_add_par(doc_1, value = paste(rep(letters, 50), collapse = " "))

# End the landscape two-column section
doc_1 <- body_end_block_section(doc_1, value = landscape_two_columns)

# Section 3: Return to portrait orientation ----
# After ending the previous sections, we're back to portrait (default)
doc_1 <- body_add_par(doc_1, "Portrait Table Section", style = "heading 1")
doc_1 <- body_add_par(
  doc_1,
  "This section returns to portrait orientation with a taller table."
)

# Add a longer table in portrait orientation
doc_1 <- body_add_table(doc_1, value = mtcars[1:25, ], style = "table_template")

# Save the document
output_file_1 <- tempfile(fileext = ".docx")
print(doc_1, target = output_file_1)


# Example 2: Different headers and footers (first, even, odd pages) ----
# This example demonstrates the complete header/footer system with:
# - Different header/footer for the first page
# - Different header/footer for even pages (left-side pages in duplex printing)
# - Default header/footer for odd pages (right-side pages)
```

```
# Create sample text to generate multiple pages
lorem_text <- paste(
  rep("Purus lectus eros metus turpis mattis platea praesent sed. ", 50),
  collapse = ""
)

# Define content for FIRST page header
# Typically used for title pages or cover pages
header_first <- block_list(
  fpar(
    ftext(
      "First Page Header - Title Page",
      fp_text_lite(bold = TRUE, color = "#4472C4", font.size = 14)
    ),
    fp_p = fp_par(
      text.align = "center",
      padding.bottom = 12,
      border.bottom = fp_border(color = "#4472C4", width = 2)
    )
  )
)

# Define content for EVEN pages header (left-side pages when printed)
# In duplex printing, this appears on the left side
header_even <- block_list(
  fpar(
    ftext("Chapter Title", fp_text_lite(italic = TRUE, font.size = 10)),
    fp_p = fp_par(text.align = "left")
  )
)

# Define content for DEFAULT pages header (odd pages/right-side)
# In duplex printing, this appears on the right side
header_default <- block_list(
  fpar(
    ftext("Document Title", fp_text_lite(italic = TRUE, font.size = 10)),
    fp_p = fp_par(text.align = "right")
  )
)

# Define content for FIRST page footer
footer_first <- block_list(
  fpar(
    ftext(
      "Company Name - Confidential",
      fp_text_lite(font.size = 9, color = "#666666")
    ),
    fp_p = fp_par(text.align = "center")
  )
)

# Define content for EVEN pages footer (includes page number on left)
footer_even <- block_list(
```

```
  fpar(
    run_word_field(field = "PAGE", prop = fp_text_lite(font.size = 9)),
    ftext(" | Document Name", fp_text_lite(font.size = 9)),
    fp_p = fp_par(
      text.align = "left",
      padding.top = 6,
      border.top = fp_border(color = "#CCCCCC", width = 1)
    )
  )
)

# Define content for DEFAULT pages footer (includes page number on right)
footer_default <- block_list(
  fpar(
    ftext("Document Name | ", fp_text_lite(font.size = 9)),
    run_word_field(field = "PAGE", prop = fp_text_lite(font.size = 9)),
    fp_p = fp_par(
      text.align = "right",
      padding.top = 6,
      border.top = fp_border(color = "#CCCCCC", width = 1)
    )
  )
)

# Create section properties with all header/footer variants
# When all three are defined (first, even, default), Word will use:
# - header_first/footer_first for page 1
# - header_even/footer_even for pages 2, 4, 6, etc.
# - header_default/footer_default for pages 3, 5, 7, etc.
section_with_all_hf <- prop_section(
  header_default = header_default,
  footer_default = footer_default,
  header_first = header_first,
  footer_first = footer_first,
  header_even = header_even,
  footer_even = footer_even
)

# Create a new document
doc_2 <- read_docx()

# Add enough content to create multiple pages
# This will demonstrate how the different headers/footers appear
for (i in 1:20) {
  doc_2 <- body_add_par(doc_2, paste0("Paragraph ", i, ": ", lorem_text))
}

# Apply the section properties with all header/footer configurations
doc_2 <- body_set_default_section(doc_2, value = section_with_all_hf)

# Save the document
# Open this document and scroll through pages to see:
# - Page 1: Special first page header/footer
```

```
# - Page 2: Even page header/footer with page number on left
# - Page 3: Odd page (default) header/footer with page number on right
# - And so on...
output_file_2 <- tempfile(fileext = ".docx")
print(doc_2, target = output_file_2)
```

`prop_table` *Table properties*

Description

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

Usage

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

Arguments

`style`	table style to be used to format table
`layout`	layout defined by `table_layout()`,
`width`	table width in the document defined by `table_width()`
`stylenames`	columns styles defined by `table_stylenames()`
`colwidths`	column widths defined by `table_colwidths()`
`tcf`	conditional formatting settings defined by `table_conditional_formatting()`
`align`	table alignment (one of left, center or right)
`word_title`	alternative text for Word table (used as title of the table)
`word_description`	alternative text for Word table (used as description of the table)

See Also

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

Examples

```
prop_table()
to_wml(prop_table())
```

`read_docx` *Create a 'Word' document object*

Description

`read_docx()` is the starting point for creating Word documents from R. It creates an R object representing a Word document that can be manipulated programmatically. When called without arguments, it creates an empty document based on a default template. When provided with a path, it reads an existing Word document (`.docx`) or template (`.dotx`) file.

Once created, you can:

- Add content from R: Insert text, formatted paragraphs (`fpar()`), tables (`body_add_table()`), plots (`body_add_plot()`), images, page breaks, table of contents, and more using the `body_add_*()` family of functions.
- Read and inspect content: Use `docx_summary()` to extract and analyze the document's content, structure, and formatting as a data frame.
- Write to file: Save the document to disk using `print(x, target = "path/to/file.docx")`.

Usage

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

Arguments

`path`	path to the docx file to use as base document. `dotx` file are supported.

Value

an object of class `rdocx`.

styles

The template file (specified via `path` or the default template) determines the available paragraph styles, character styles, and table styles in your document. These styles control the appearance of headings, body text, tables, and other elements.

When you use functions like `body_add_par(style = "heading 2")`, the style name must exist in the template. You can:

- Use `styles_info()` to list all available styles in your document
- Create a custom template in Word with your organization's styles and branding
- Use the default template for standard documents

The document layout (page size, margins, headers and footer content, orientation) also comes from the template and can be modified using `body_set_default_section()` or by adding section breaks with `body_end_section_continuous()` and related functions.

Illustrations

See Also

Save a 'Word' document object to a file with `print.rdocx()`, add content with functions `body_add_par()`, `body_add_plot()`, `body_add_table()`, change settings with `docx_set_settings()`, set properties with `set_doc_properties()`, read 'Word' styles with `styles_info()`.

Examples

```
library(officer)

# This example demonstrates how to create
# an empty document -----

## Create a new Word document
doc <- read_docx()
## Save the document
output_file <- print(doc, target = tempfile(fileext = ".docx"))

# This example demonstrates how to create a document
# with text, formatted paragraphs, tables, and plots ----
# organized in sections

# Create a new Word document
doc <- read_docx()

# Add main title
doc <- body_add_par(doc, "Annual Sales Report", style = "heading 1")

# Add introduction with formatted text using fpar
intro_text <- fpar(
  "This report presents the ",
  ftext(
    "quarterly sales analysis",
    fp_text_lite(bold = TRUE, color = "#C32900")
  ),
  " for the fiscal year. The following sections provide detailed insights ",
  "into our performance metrics and trends."
)
doc <- body_add_fpar(doc, intro_text)

## Section 1: Sales Data ----
doc <- body_add_par(doc, "Sales Performance", style = "heading 2")

# Add descriptive text
doc <- body_add_par(
  doc,
  "The table below summarizes sales data across different product categories:"
)
```

```
# Create and add a data frame as a table
sales_data <- data.frame(
  Quarter = c("Q1", "Q2", "Q3", "Q4"),
  Revenue = c(125000, 142000, 156000, 178000),
  Units = c(1250, 1420, 1560, 1780),
  Growth = c("5%", "13.6%", "9.9%", "14.1%"),
  stringsAsFactors = FALSE
)
doc <- body_add_table(doc, value = sales_data, style = "table_template")

# Add commentary with multiple formatted text elements
comment <- fpar(
  "Key finding: ",
  ftext(
    "Q4 showed the strongest performance",
    fp_text_lite(bold = TRUE, font.size = 11)
  ),
  " with a ",
  ftext("14.1% growth rate", fp_text_lite(color = "#006699", bold = TRUE)),
  " compared to the previous quarter."
)
doc <- body_add_fpar(doc, comment)

## Section 2: Visualizations ----
doc <- body_add_par(doc, "Revenue Trends", style = "heading 2")

# Add explanatory text
doc <- body_add_par(
  doc,
  "Figure 1 illustrates the quarterly revenue progression throughout the year."
)

# Create a plot showing revenue trends
revenue_plot <- plot_instr({
  quarters <- c("Q1", "Q2", "Q3", "Q4")
  revenue <- c(125000, 142000, 156000, 178000)
  barplot(
    revenue,
    names.arg = quarters,
    col = "#4472C4",
    border = NA,
    main = "Quarterly Revenue",
    ylab = "Revenue ($)",
    xlab = "Quarter",
    ylim = c(0, 200000)
  )
  grid(nx = NA, ny = NULL, col = "gray90", lty = 1)
})
doc <- body_add_plot(doc, revenue_plot, width = 6, height = 4)

# Add another section with a different plot
doc <- body_add_par(doc, "Sales Distribution Analysis", style = "heading 2")
```

```
# Add context for the second plot
analysis_intro <- fpar(
  "The distribution analysis below shows the ",
  ftext("variability in daily sales", fp_text_lite(italic = TRUE)),
  " across all quarters. This helps identify patterns and outliers in our sales data."
)
doc <- body_add_fpar(doc, analysis_intro)

# Create a density plot
distribution_plot <- plot_instr({
  # Simulate daily sales data
  set.seed(123)
  daily_sales <- c(
    rnorm(90, mean = 1400, sd = 200), # Q1-Q3
    rnorm(30, mean = 2000, sd = 250) # Q4 (higher mean)
  )
  plot(
    density(daily_sales),
    main = "Distribution of Daily Sales",
    xlab = "Daily Sales (Units)",
    ylab = "Density",
    col = "#C32900",
    lwd = 2
  )
  polygon(density(daily_sales), col = rgb(0.76, 0.16, 0, 0.2), border = NA)
  abline(v = mean(daily_sales), col = "#006699", lwd = 2, lty = 2)
  legend("topright", legend = "Mean", col = "#006699", lty = 2, lwd = 2)
})
doc <- body_add_plot(doc, distribution_plot, width = 6, height = 4)

# Add concluding remarks
doc <- body_add_par(doc, "Conclusion", style = "heading 2")
conclusion <- fpar(
  "The analysis demonstrates ",
  ftext("consistent growth", fp_text_lite(bold = TRUE, color = "#006699")),
  " throughout the year, with particularly strong performance in Q4. ",
  "This trend suggests effective market strategies and increasing customer demand."
)
doc <- body_add_fpar(doc, conclusion)

# Save the document
comprehensive_file <- print(doc, target = tempfile(fileext = ".docx"))
# Using a custom template ----
# This example shows how to start from an existing template
# instead of creating a blank document

# Get the path to a landscape template included in the package
template <- system.file(package = "officer", "doc_examples", "landscape.docx")

# Create a document based on the template
# The document will inherit the template's styles and page settings
doc_2 <- read_docx(path = template)
```

```
# Add a section with a table
doc_2 <- body_add_par(doc_2, "Motor Trend Car Data", style = "heading 2")
doc_2 <- body_add_table(doc_2, value = head(mtcars))

# Add a section with a plot
doc_2 <- body_add_par(doc_2, "Sales Distribution", style = "heading 2")
doc_2 <- body_add_plot(doc_2, distribution_plot)

# Save the document
docx_file_output <- print(doc_2, target = tempfile(fileext = ".docx"))
```

`read_pptx` *Create a 'PowerPoint' document object*

Description

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

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

Usage

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

Arguments

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

master layouts and slide layouts

`read_pptx()` uses a PowerPoint file as the initial document. This is the original PowerPoint document where all slide layouts, placeholders for shapes and styles come from. Major points to be aware of are:

- Slide layouts are relative to a master layout. A document can contain one or more master layouts; a master layout can contain one or more slide layouts.
- A slide layout inherits design properties from its master layout but some properties can be overwritten.
- Designs and formatting properties of layouts and shapes (placeholders in a layout) are defined within the initial document. There is no R function to modify these values - they must be defined in the initial document.

See Also

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

Examples

```
read_pptx()
```

read_xlsx *Create an 'Excel' document object*

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 *Remove slide(s)*

Description

Remove one or more slides from a pptx presentation.

Usage

```
remove_slide(x, index = NULL, rm_images = FALSE)
```

Arguments

x	an rpptx object
index	slide index or a vector of slide indices to remove, default to current slide position.
rm_images	unused anymore.

Note

cursor is set on the last slide.

See Also

read_pptx(), ph_with(), ph_remove()

Other functions to manipulate slides: add_slide(), move_slide(), on_slide(), set_notes()

Examples

```
library(officer)

x <- read_pptx()
x <- add_slide(x, "Title and Content")
x <- remove_slide(x)

# Remove multiple slides at once
x <- read_pptx()
x <- add_slide(x, "Title and Content")
x <- add_slide(
  x,
  layout = "Two Content",
  `Title 1` = "A title",
  dt = "Jan. 26, 2025",
  `body[2]` = "Body 2",
  left = "Left side",
  `6` = "Footer"
)
x <- add_slide(
  x,
  layout = "Two Content",
  `Title 1` = "A title",
  dt = "Jan. 26, 2025",
  `body[2]` = "Body 2",
  left = "Left side",
  `6` = "Footer"
)
x <- add_slide(x, "Title and Content")
x <- remove_slide(x, index = c(2, 4))
pptx_file <- print(x, target = tempfile(fileext = ".pptx"))
pptx_file
```

rtf_add *Add content into an RTF document*

Description

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

Usage

```
rtf_add(x, value, ...)

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

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

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

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

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

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

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

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

## S3 method for class 'plot_instr'
rtf_add(
```

```
  x,
  value,
  width = 6,
  height = 5,
  res = 300,
  scale = 1,
  ppr = fp_par(text.align = "center"),
  ...
)
```

Arguments

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

Methods (by class)

- rtf_add(block_section): add a new section definition
- rtf_add(character): add characters as new paragraphs
- rtf_add(factor): add a factor vector as new paragraphs
- rtf_add(double): add a double vector as new paragraphs
- rtf_add(fpar): add an fpar()
- rtf_add(block_list): add an block_list()
- rtf_add(block_toc): add a block_toc() (table of contents). Word populates the TOC at open time from paragraphs styled with the built-in heading styles (which carry an outline level). LibreOffice will not render the TOC automatically.
- rtf_add(gg): add a ggplot2
- rtf_add(plot_instr): add a plot_instr() object

Section model in RTF

A `block_section` added with `rtf_add()` applies to the content that **follows** the call: it opens a new section whose layout (orientation, columns, margins, headers / footers) is inherited by every paragraph, table or graphic added afterwards, until the next `block_section` (or the end of the document).

Typical pattern: declare the section, then add the content.

```
doc <- rtf_doc() |>
  rtf_add(block_section(prop_section(
    page_size = page_size(orient = "landscape")
  ))) |>
  rtf_add(fpar("This paragraph is in landscape orientation."))
```

The first section of the document is configured via the `def_sec` argument of `rtf_doc()` (a prop_section object, not a `block_section`). It applies to every element added before the first `rtf_add(block_section(...))` call.

The Word output uses the opposite model: `body_end_block_section()` applies to the content that *precedes* the call. See `body_end_block_section()`.

Examples

```
## Simple RTF example ----

library(officer)

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

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

doc <- rtf_add(
  x = doc,
  value = fpar(
    ftext("how are you?", prop = def_text),
    fp_p = fp_par(text.align = "center")
  )
)

a_paragraph <- fpar(
  ftext("Here is a date: ", prop = def_text),
  run_word_field(field = "Date \\@ \"MMMM d yyyy\""),
  fp_p = center_par
)
doc <- rtf_add(
  x = doc,
  value = block_list(
    a_paragraph,
    a_paragraph,
```

```
    a_paragraph
  )
)

if (require("ggplot2")) {
  gg <- gg_plot <- ggplot(data = iris) +
    geom_point(mapping = aes(Sepal.Length, Petal.Length))
  doc <- rtf_add(doc, gg, width = 3, height = 4, ppr = center_par)
}
anyplot <- plot_instr(code = {
  barplot(1:5, col = 2:6)
})
doc <- rtf_add(doc, anyplot, width = 5, height = 4, ppr = center_par)

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


## RTF example with sections ----

library(officer)

quick_section_header <- function(label) {
  block_list(
    fpar(
      ftext(label, fp_text_lite(bold = TRUE, color = "#006699"))
    )
  )
}

quick_section_footer <- function(label) {
  block_list(
    fpar(
      "Page ",
      run_word_field(field = "PAGE  \\* MERGEFORMAT")
    )
  )
}

quick_hello_world <- function(doc) {
  rtf_add(
    doc,
    fpar(
      ftext(
        "Hello World"
      ),
      fp_p = fp_par(text.align = "left")
    )
  )
}

three_cols_section <- block_section(
  prop_section(
    type = "continuous",
```

```
    section_columns = section_columns(widths = c(1.7, 1.7, 1.7), space = 0.25),
    header_default = quick_section_header("Three columns section"),
    footer_default = quick_section_footer("Three columns section")
  )
)

doc <- rtf_doc(
  def_sec = prop_section(
    header_default = quick_section_header("Default section"),
    footer_default = quick_section_footer("Default section")
  ),
  normal_par = fp_par(padding = 3)
)
doc <- rtf_add(doc, block_toc(level = 3))

doc <- rtf_add(doc, "Sections demo", style = "heading 1")
doc <- rtf_add(doc, "Default section", style = "heading 2")
doc <- quick_hello_world(doc)

if (require("ggplot2")) {
  gg_iris <- ggplot(iris, aes(Sepal.Length, Petal.Length, colour = Species)) +
    geom_point() +
    theme_minimal()
  doc <- rtf_add(doc, gg_iris, width = 4, height = 3)
}

doc <- rtf_add(doc, "Three columns", style = "heading 2")

doc <- rtf_add(doc, three_cols_section)

titles_list <- block_list(
  fpar(ftext("Left Title"), fp_p = fp_par(text.align = "left")),
  fpar(
    run_columnbreak(),
    ftext("Centered Title"),
    fp_p = fp_par(text.align = "center")
  ),
  fpar(
    run_columnbreak(),
    ftext("Right Title"),
    fp_p = fp_par(text.align = "right")
  )
)
doc <- rtf_add(doc, titles_list)
doc <- rtf_add(doc, block_section(prop_section()))
doc <- rtf_add(doc, fpar(run_linebreak()))
doc <- quick_hello_world(doc)

landscape_section <- block_section(prop_section(
  type = "nextPage",
  page_size = page_size(orient = "landscape"),
  header_default = quick_section_header("Landscape section"),
  footer_default = quick_section_footer("Landscape section")
```

```
))
doc <- rtf_add(doc, landscape_section)
doc <- rtf_add(doc, "Landscape orientation", style = "heading 2")
doc <- quick_hello_world(doc)

doc <- rtf_add(
  doc,
  block_section(
    prop_section(
      type = "nextPage",
      header_default = quick_section_header("Final section"),
      footer_default = quick_section_footer("Final section")
    )
  )
)
doc <- rtf_add(doc, "Back to portrait", style = "heading 2")
doc <- quick_hello_world(doc)
print(doc, target = tempfile(fileext = ".rtf"))
```

`rtf_doc` *Create an RTF document object*

Description

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

Usage

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

Arguments

`def_sec`	a prop_section object used to define the default section (page size, margins, header / footer, columns) applied to content added before any explicit `block_section()`.
`normal_par`	an object generated by `fp_par()`
`normal_chunk`	an object generated by `fp_text()`

Value

an object of class `rtf` representing an empty RTF document.

Built-in styles

`rtf_doc()` registers four paragraph styles: `"Normal"` (built from `normal_par` / `normal_chunk`) and `"heading 1"` / `"heading 2"` / `"heading 3"` (bold, dark blue shades, with outline levels so the paragraphs feed Word's navigation pane and TOC field). The heading names mirror Word's own convention so a script that adds paragraphs with `style = "heading 1"` works identically against Word output (`body_add_par(..., style = "heading 1")`) and RTF output (`rtf_add(..., style = "heading 1")`).

To override a built-in look (size, color, spacing, outline level), call `rtf_set_paragraph_style()` after `rtf_doc()` with the same `style_id`.

```
doc <- rtf_doc()
doc <- rtf_set_paragraph_style(
  doc,
  style_id = "heading 1",
  fp_p = fp_par(text.align = "center", padding = 12),
  fp_t = fp_text_lite(bold = TRUE, font.size = 24, color = "#000000"),
  outline_level = 1L
)
```

Use `rtf_styles_info()` to inspect the current style table.

See Also

`read_docx()`, `print.rtf()`, `rtf_add()`. See `?rtf_add` for a complete multi-section example exercising `def_sec`, headers / footers and several `block_section()` calls.

Examples

```
rtf_doc(normal_par = fp_par(padding = 3))
```

---

rtf_set_paragraph_style

Add or replace a paragraph style in an RTF document

Description

Add or replace a paragraph style in the document stylesheet so that subsequent paragraphs can reference it via the `style` argument of `rtf_add()`. The function mirrors `docx_set_paragraph_style()` for Word.

Usage

```
rtf_set_paragraph_style(
  x,
  style_id,
  style_name = style_id,
```

```
  base_on = "Normal",
  fp_p = fp_par(),
  fp_t = NULL,
  outline_level = NULL
)
```

Arguments

x	an rtf object created by rtf_doc().
style_id	user-facing identifier used as the key when a paragraph references the style (rtf_add(..., style = style_id)).
style_name	display label of the style as it appears in Word's style menu. Defaults to style_id.
base_on	the style_id of the parent style. Properties not defined in the new style are inherited from the parent. Defaults to "Normal".
fp_p	paragraph formatting properties, see fp_par().
fp_t	text formatting properties, see fp_text(). If NULL the paragraph inherits text properties from the parent style.
outline_level	integer between 1 and 9, or NULL. Sets the outline level (\\outlinelevel) so paragraphs using the style feed Word's navigation pane and TOC field. Level 1 is the topmost (used by Heading 1).

Value

the rtf object with the style added or updated.

See Also

docx_set_paragraph_style(), rtf_doc(), rtf_add()

Examples

```
doc <- rtf_doc()
doc <- rtf_set_paragraph_style(
  doc,
  style_id = "Callout",
  fp_p = fp_par(text.align = "center", padding = 6),
  fp_t = fp_text_lite(bold = TRUE, color = "#1F4E79")
)
doc <- rtf_add(doc, "Heads up", style = "Callout")
print(doc, target = tempfile(fileext = ".rtf"))
```

rtf_styles_info *Read paragraph styles defined on an RTF document*

Description

Return the data.frame of styles currently registered on the document. Useful for debugging or for checking which built-in styles are available before referencing them via `style = ...`.

Usage

```
rtf_styles_info(x)
```

Arguments

x	an rtf object created by `rtf_doc()`.

Value

a data.frame with one row per style.

See Also

`rtf_set_paragraph_style()`, `styles_info()`

Examples

```
rtf_styles_info(rtf_doc())
```

run_autonum *Auto number*

Description

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

Usage

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

```
  tnd = 0,
  tns = "-"
)
```

Arguments

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

usage

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

See Also

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

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

Examples

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

run_bookmark *Bookmark for 'Word'*

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(), floating_external_img(), ftext(), hyperlink_ftext(), run_autonum(), run_columnbreak(), run_comment(), run_footnote(), run_footnoteref(), run_linebreak(), run_pagebreak(), run_reference(), run_tab(), run_word_field(), run_wordtext()

Examples

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

run_columnbreak *Column break for 'Word'*

Description

Create a representation of a column break.

Usage

```
run_columnbreak()
```

usage

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

See Also

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

Examples

```
run_columnbreak()
```

---

run_comment *Comment for 'Word'*

Description

Add a comment on a run object.

Usage

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

Arguments

cmt	a set of blocks to be used as comment content returned by function `block_list()`. the "run functions for reporting".
run	a run object, made with a call to one of
author	comment author.
date	comment date
initials	comment initials
prop	formatting text properties returned by `fp_text_lite()` or `fp_text()`. It also can be NULL in which case, no formatting is defined (the default is applied).

See Also

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

Examples

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

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

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

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

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

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

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

---

run_footnote *Footnote for 'Word'*

Description

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

Usage

```
run_footnote(x, prop = NULL)
```

Arguments

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

See Also

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

Examples

```
library(officer)

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

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

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

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

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

---

run_footnoteref *Word footnote reference*

Description

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

Usage

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

Arguments

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

See Also

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

Examples

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

---

run_linebreak *Page break for 'Word'*

Description

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

Usage

```
run_linebreak()
```

usage

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

See Also

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

Examples

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

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

run_pagebreak *Page break for 'Word'*

Description

Object representing a page break for a Word document.

Usage

```
run_pagebreak()
```

usage

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

See Also

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

Examples

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

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

run_reference *Cross reference*

Description

Create a representation of a reference

Usage

```
run_reference(id, prop = NULL)
```

Arguments

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

usage

You can use this function in conjunction with 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(), floating_external_img(), ftext(), hyperlink_ftext(), run_autonum(), run_bookmark(), run_columnbreak(), run_comment(), run_footnote(), run_footnoteref(), run_linebreak(), run_pagebreak(), run_tab(), run_word_field(), run_wordtext()

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

Examples

```
run_reference("a_ref")
```

---

run_tab *Tab for 'Word'*

Description

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

Tabulation marks settings can be defined with fp_tabs() in paragraph settings defined with fp_par().

Usage

```
run_tab()
```

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(), floating_external_img(), ftext(), hyperlink_ftext(), run_autonum(), run_bookmark(), run_columnbreak(), run_comment(), run_footnote(), run_footnoteref(), run_linebreak(), run_pagebreak(), run_reference(), run_word_field(), run_wordtext()

Examples

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

---

run_wordtext *Word chunk of text with a style*

Description

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

Usage

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

Arguments

text	text value, a single character value
style_id	'Word' unique style identifier associated with the style to use.

See Also

ftext()

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

Examples

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

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

`run_word_field` *'Word' computed field*

Description

Create a 'Word' computed field.

Usage

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

Arguments

`field`	Value for a "Word Computed Field" as 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()`, `floating_external_img()`, `ftext()`, `hyperlink_ftext()`, `run_autonum()`, `run_bookmark()`, `run_columnbreak()`, `run_comment()`, `run_footnote()`, `run_footnoteref()`, `run_linebreak()`, `run_pagebreak()`, `run_reference()`, `run_tab()`, `run_wordtext()`

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

Examples

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

section_columns *Section columns*

Description

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

Usage

```
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

```
section_columns()
```

set_autonum_bookmark *Update bookmark of an autonumber run*

Description

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

Usage

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

Arguments

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

See Also

run_autonum()

Examples

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

`set_doc_properties` *Set document properties*

Description

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

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

Usage

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

Arguments

`x`	an rdocx or rpptx object
`title`, `subject`, `creator`, `description`	text fields
`created`	a date object
`hyperlink_base`	a string specifying the base URL for relative hyperlinks in the document (only for rdocx).
`...`	named arguments (names are field names), each element is a single character value specifying value associated with the corresponding field name. These pairs of *key-value* are added as custom properties. If a value is `NULL` or `NA`, the corresponding field is set to ” in the document properties.

values	a named list (names are field names), each element is a single character value specifying value associated with the corresponding field name. If values is provided, argument ... will be ignored.

Note

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

See Also

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

Examples

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

---

set_notes *Set notes for current slide*

Description

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

Usage

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

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

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

Arguments

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

Methods (by class)

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

See Also

`print.rpptx()`, `read_pptx()`, `add_slide()`, `notes_location_label()`, `notes_location_type()`

Other functions to manipulate slides: `add_slide()`, `move_slide()`, `on_slide()`, `remove_slide()`

Examples

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

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

print(doc, target = fileout)
```

---

sheet_add_drawing *Add a drawing to an Excel sheet*

Description

Add a graphical element into a sheet of an xlsx workbook. This is a generic function dispatching on `value`. Supported inputs include:

- external_img(): an image file (PNG, JPEG, GIF, ...) copied into the workbook and placed on the sheet.
- gg: a ggplot object rendered to PNG via ragg::agg_png() and embedded via the external_img path. Extra arguments: res (default 300 ppi), alt_text (auto-detected via ggplot2::get_alt_text() when empty), scale (same semantics as ggplot2::ggsave()).

The drawing can be positioned in three ways, matching Excel's own "Format Picture" options:

- Fixed position (default): pass left / top and width / height in inches. The drawing stays put when rows or columns are inserted/resized.
- Move and size with cells (Excel's default): pass anchor = "B2:H20" (a cell range). The drawing spans those two cells and follows them.
- Move but don't size with cells: pass anchor = "B2" (a single cell). The drawing anchors to that cell, keeping the size set by width / height.

Additional methods for ms_chart and dml are provided by the extension packages 'mschart' and 'rvg'.

Use sheet_write_data() to write data into the sheet before or after adding a drawing.

Usage

```
sheet_add_drawing(x, value, sheet, ...)
```

Arguments

x	rxlsx object created by read_xlsx()
value	object to add (dispatched to the appropriate method)
sheet	sheet name (must already exist, see add_sheet())
...	method-specific arguments. Common across external_img and gg methods: left / top (fixed-position top-left in inches, default (1, 1)), width / height (size in inches), anchor (cell reference like "B2" or "B2:H20", see description), edit_as (one of "twoCell", "oneCell", "absolute"; how Excel treats the drawing when rows/columns are resized — only meaningful with a cell range anchor). The gg method also accepts res (ppi, default 300), alt_text (auto-detected via ggplot2::get_alt_text() if empty) and scale (passed to ragg::agg_png()).

Value

the rxlsx object (invisibly)

See Also

read_xlsx(), add_sheet(), sheet_write_data()

Examples

```
img <- system.file("extdata", "example.png", package = "officer")
if (nzchar(img) && file.exists(img)) {
  x <- read_xlsx()
  x <- add_sheet(x, label = "pics")
  x <- sheet_add_drawing(
    x, sheet = "pics",
    value = external_img(img, width = 2, height = 2)
  )
  print(x, target = tempfile(fileext = ".xlsx"))
}

if (requireNamespace("ggplot2", quietly = TRUE)) {
  gg <- ggplot2::ggplot(iris, ggplot2::aes(Sepal.Length, Sepal.Width)) +
    ggplot2::geom_point()
  x <- read_xlsx()
  x <- add_sheet(x, label = "plots")
  x <- sheet_add_drawing(x, value = gg, sheet = "plots",
                         width = 4, height = 3)
  print(x, target = tempfile(fileext = ".xlsx"))
}
```

---

sheet_names *Sheet names of an xlsx workbook*

Description

Return the sheet names of an `rxlsx` object, in the order they appear in the workbook.

Usage

```
sheet_names(x)
```

Arguments

x	an rxlsx object (created by `read_xlsx()`).

Value

A character vector of sheet names.

Examples

```
wb <- read_xlsx()
sheet_names(wb)

wb <- add_sheet(wb, label = "new sheet")
sheet_names(wb)
```

sheet_remove *Remove a sheet*

Description

Remove a sheet from an xlsx workbook, deleting the worksheet XML, its relationship file and the content-type override.

Usage

```
sheet_remove(x, sheet)
```

Arguments

x	rxlsx object
sheet	name of the sheet to remove

Value

the rxlsx object (invisibly)

Examples

```
wb <- read_xlsx()
wb <- add_sheet(wb, "kept")
default_name <- sheet_names(wb)[1]
wb <- sheet_remove(wb, sheet = default_name)
print(wb, target = tempfile(fileext = ".xlsx"))
```

sheet_select *Select sheet*

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") )
```

`sheet_write_data` *Write data to a sheet*

Description

Write a content into a sheet of an xlsx workbook. Multiple calls can write to different positions on the same sheet.

This is a generic function dispatching on `value`. Supported inputs:

- `data.frame`: written as a table (header in row `start_row`, data starting at `start_row + 1`).
- `character`: each element in its own cell. The `character` method accepts a `direction` argument (`"vertical"` – default – or `"horizontal"`) to stack elements in a column or a row.
- `fpar()`: richtext paragraph written into a single cell at `(start_row, start_col)`. Font, size, colour, bold, italic, underline, strikethrough and sub/superscript chunks are honoured.
- `block_list()`: one cell per `fpar` item. Also accepts `direction = "vertical"` (default) or `"horizontal"`.

Usage

```
sheet_write_data(x, value, sheet, start_row = 1L, start_col = 1L, ...)
```

Arguments

`x`	rxlsx object
`value`	a `data.frame`, `character` vector, `fpar()` or `block_list()`
`sheet`	sheet name (must already exist)
`start_row`	row index where the header / first cell will be written (default 1)
`start_col`	column index where the first column / first cell will be written (default 1)
`...`	method-specific arguments. In particular, `direction = "vertical"` (default) or `"horizontal"` is honoured by the `character` and `block_list` methods.

Examples

```
# sheet_write_data() is an S3 generic dispatching on `value`:
#   - data.frame
#   - character
#   - fpar
#   - block_list
```

```
library(officer)

wb <- read_xlsx()
# drop the template's default sheet so the workbook contains only `demo`
wb <- sheet_remove(wb, sheet = sheet_names(wb)[1])
wb <- add_sheet(wb, label = "demo")

# --- A1:C4  data.frame (tabular write) ----
wb <- sheet_write_data(
  wb,
  sheet = "demo",
  value = head(iris, 3)[, 1:3],
  start_row = 1,
  start_col = 1
)

# --- A6:A8  character, vertical default ----
wb <- sheet_write_data(
  wb,
  sheet = "demo",
  value = c("first", "second", "third"),
  start_row = 6,
  start_col = 1
)

# --- A10:C10  character, horizontal ----
wb <- sheet_write_data(
  wb,
  sheet = "demo",
  value = c("one", "two", "three"),
  direction = "horizontal",
  start_row = 10,
  start_col = 1
)

# --- A12  fpar with rich text in a single cell ----
wb <- sheet_write_data(
  wb,
  sheet = "demo",
  start_row = 12,
  start_col = 1,
  value = fpar(
    ftext("bold ", fp_text(bold = TRUE, color = "red", font.size = 12)),
    ftext("italic ", fp_text(italic = TRUE, color = "blue")),
    ftext("under ", fp_text(underlined = TRUE)),
    ftext("strike ", fp_text(strike = TRUE, color = "#888888")),
    ftext("H", fp_text()),
    ftext("2", fp_text(vertical.align = "subscript")),
    ftext("O  (m", fp_text()),
    ftext("2", fp_text(vertical.align = "superscript")),
    ftext(")", fp_text())
  )
)
```

```
# --- A14:A16  block_list of fpars, stacked vertically ----
wb <- sheet_write_data(
  wb,
  sheet = "demo",
  start_row = 14,
  start_col = 1,
  value = block_list(
    fpar(ftext(
      "header row",
      fp_text(bold = TRUE, font.size = 14, color = "#006699")
    )),
    fpar(ftext("middle row", fp_text(italic = TRUE, color = "#222222"))),
    fpar(ftext(
      "final row",
      fp_text(bold = TRUE, color = "darkgreen", font.size = 11)
    ))
  )
)

# --- B14:D14  block_list horizontal, 3 fpars in a row ----
wb <- sheet_write_data(
  wb,
  sheet = "demo",
  start_row = 14,
  start_col = 2,
  direction = "horizontal",
  value = block_list(
    fpar(ftext("left", fp_text(bold = TRUE))),
    fpar(ftext("middle", fp_text(color = "red"))),
    fpar(ftext("right", fp_text(italic = TRUE)))
  )
)

# --- Commentary column (E) via character ---
wb <- sheet_write_data(
  wb,
  sheet = "demo",
  start_row = 1,
  start_col = 5,
  value = c(
    "data.frame -> tabular (iris cols 1-3)",
    "",
    "",
    "character(3) vertical -> A6:A8",
    "",
    "",
    "",
    "character(3) horizontal -> A10:C10",
    "",
    "",
    "fpar with 9 ftext chunks in A12",
    "",
```

```
    "",
    "block_list(3) vertical -> A14:A16",
    "",
    "",
    "(B14:D14) block_list horizontal"
  )
)

out <- tempfile(fileext = ".xlsx")
print(wb, target = out)
```

---

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 information: `annotate_base()`, `color_scheme()`, `doc_properties()`, `layout_properties()`, `layout_summary()`, `length.rpptx()`, `plot_layout_properties()`, `slide_summary()`

Examples

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

`slide_summary` *Slide content in a data.frame*

Description

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

Usage

```
slide_summary(x, index = NULL)
```

Arguments

`x`	an rpptx object
`index`	slide index

Note

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

See Also

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

Examples

```
my_pres <- read_pptx()
my_pres <- add_slide(my_pres, "Title and Content")
my_pres <- ph_with(my_pres, format(Sys.Date()),
  location = ph_location_type(type="dt"))
my_pres <- add_slide(my_pres, "Title and Content")
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)
```

slide_visible<- *Get or set slide visibility*

Description

PPTX slides can be visible or hidden. This function gets or sets the visibility of slides.

Usage

```
slide_visible(x) <- value

slide_visible(x, hide = NULL, show = NULL)
```

Arguments

x	An rpptx object.
value	Boolean vector with slide visibilities.
hide, show	Indexes of slides to hide or show.

Value

Boolean vector with slide visibilities or rpptx object if changes are made to the object.

Examples

```
path <- system.file("doc_examples/example.pptx", package = "officer")
x <- read_pptx(path)

slide_visible(x) # get slide visibilities

x <- slide_visible(x, hide = 1:2) # hide slides 1 and 2
x <- slide_visible(x, show = 1:2) # make slides 1 and 2 visible
x <- slide_visible(x, show = 1:2, hide = 3)

slide_visible(x) <- FALSE # hide all slides
slide_visible(x) <- c(TRUE, FALSE, TRUE) # set each slide separately
slide_visible(x) <- c(TRUE, FALSE) # warns that rhs values are recycled

slide_visible(x)[2] <- TRUE # set 2nd slide to visible
slide_visible(x)[c(1, 3)] <- FALSE # 1st and 3rd slide
slide_visible(x)[c(1, 3)] <- c(FALSE, FALSE) # identical
```

sp_line *Line properties*

Description

Create a sp_line object that describes line properties.

Usage

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

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

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

## S3 method for class 'sp_line'
to_pml(x, add_ns = FALSE, ...)
```

Arguments

color	line color - a single character value specifying a valid color (e.g. "#000000" or "black").
lwd	line width (in point) - 0 or positive integer value.

lty	single character value specifying the line type. Expected value is one of the following : default 'solid' or 'dot' or 'dash' or 'lgDash' or 'dashDot' or 'lgDashDot' or 'lgDashDotDot' or 'sysDash' or 'sysDot' or 'sysDashDot' or 'sysDashDotDot'.
linecmpd	single character value specifying the compound line type. Expected value is one of the following : default 'sng' or 'dbl' or 'tri' or 'thinThick' or 'thickThin'
lineend	single character value specifying the line end style Expected value is one of the following : default 'rnd' or 'sq' or 'flat'
linejoin	single character value specifying the line join style Expected value is one of the following : default 'round' or 'bevel' or 'miter'
headend	a sp_lineend object specifying line head end style
tailend	a sp_lineend object specifying line tail end style
x, object	sp_line object
...	further arguments - not used
add_ns	unused; kept for compatibility with the to_pml() generic.

Value

a sp_line object

See Also

sp_lineend

Other functions for defining shape properties: sp_lineend()

Examples

```
library(officer)

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

---

sp_lineend *Line end properties*

Description

Create a sp_lineend object that describes line end properties.

Usage

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

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

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

Arguments

type	single character value specifying the line end type. Expected value is one of the following : default 'none' or 'triangle' or 'stealth' or 'diamond' or 'oval' or 'arrow'
width	single character value specifying the line end width Expected value is one of the following : default 'sm' or 'med' or 'lg'
length	single character value specifying the line end length Expected value is one of the following : default 'sm' or 'med' or 'lg'
x, object	sp_lineend object
...	further arguments - not used

Value

a sp_lineend object

See Also

sp_line

Other functions for defining shape properties: sp_line()

Examples

```
library(officer)

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

styles_info *Read 'Word' styles*

Description

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

Usage

```
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

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

table_colwidths *Column widths of a table*

Description

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

Usage

```
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

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

Arguments

`first_row`, `last_row`
: apply or remove formatting from the first or last row in the table.

`first_column`, `last_column`
: apply or remove formatting from the first or last column in the table.

`no_hband`, `no_vband`
: don't display odd and even rows or columns with alternating shading for ease of reading.

Note

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

See Also

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

Examples

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

table_layout *Algorithm for table layout*

Description

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

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

Usage

```
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)
)

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

---

table_width *Preferred width for a table*

Description

Define the preferred width for a table.

Usage

```
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.

Consider using block_list_items() instead, which supports rich text via fpar(), works in both Word and PowerPoint, and supports numbered lists.

Usage

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

Arguments

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

See Also

ph_with()

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

Examples

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

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

Index