Package 'ReporteRs'

March 28, 2018

Type Package

Title Microsoft Word and PowerPoint Documents Generation

Version 0.8.10

Description Create 'Microsoft Word' document (>=2007) and 'Microsoft PowerPoint' document (>=2007) from R. There are several features to let you format and present R outputs ; e.g. Editable Vector Graphics, functions for complex tables reporting, reuse of corporate template document. You can use the package as a tool for fast reporting and as a tool for reporting automation. The package does not require any installation of Microsoft product to be able to write Microsoft files.

License GPL-3

Depends R (>= 3.0), ReporteRsjars (>= 0.0.2)

Imports grDevices, rJava, utils, rvg (>= 0.1.2), xml2, gdtools, png, R.utils, knitr, shiny, htmltools

Suggests ggplot2, jpeg, bmp, testthat, magrittr, rmarkdown

SystemRequirements java (>= 6)

URL https://davidgohel.github.io/ReporteRs

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

RoxygenNote 6.0.1.9000

NeedsCompilation no

Author David Gohel [aut, cre]

Maintainer ORPHANED

X-CRAN-Original-Maintainer David Gohel <david.gohel@lysis-consultants.fr>

X-CRAN-Comment Orphaned on 2018-03-28 as maintainer address bounced.

Repository CRAN

Date/Publication 2018-03-28 11:28:08 UTC

R topics documented:

`ReporteRs-package` *ReporteRs: a package to create document from R*

Description

ReporteRs lets you create Microsoft Word, Microsoft PowerPoint and html documents.

Details

To get an r document object:

- docx Create a Microsoft Word document object
- pptx Create a Microsoft PowerPoint document object

The following functions can be used whatever the output format is (docx, pptx).

- addTitle Add a title
- addFlexTable Add a table (new)
- addPlot Add plots
- addImage Add external images
- addParagraph Add paragraphs of text
- addRScript Add an r script
- writeDoc Write the document into a file or a directory

ReporteRs comes with an object of class pot to let you handle text output and format. You can associate a text with formats (font size, font color, etc.), with an hyperlink or with a Footnote as a reference note.

ReporteRs comes also with an object of class FlexTable that let you design and format tabular outputs.

Default values:

With ReporteRs, some options can be used to reduce usage of some parameters:

- "ReporteRs-default-font" Default font family to use (default to "Helvetica"). This will be used as default values for argument fontname of addPlot and argument font.family of pot.
- "ReporteRs-fontsize" Default font size to use (default to 11). This will be used as default values for argument pointsize of addPlot and argument font.size of pot.
- "ReporteRs-list-definition" see list.settings.
- "ReporteRs-locale.language" language encoding (for html objects). Default to "en".
- "ReporteRs-locale.region" region encoding (for html objects). Default to "US".

Note

Examples are in a donttest section as they are using font that may be not available on the host machine. Default font is Helvetica, it can be modified with option ReporteRs-default-font. To run an example with 'Arial' default font, run first

```
options("ReporteRs-default-font" = "Arial")
```

Author(s)

David Gohel <david.gohel@lysis-consultants.fr>

Examples

```
options("ReporteRs-fontsize"=10, "ReporteRs-default-font"="Helvetica")
```

+.pot *pot concatenation*

Description

"+" function is to be used for concatenation of pot elements. Concatenation of 2 pot objects returns a pot (of length 2).

Usage

```
## S3 method for class 'pot'
e1 + e2
```

Arguments

e1	a pot object or a character (vector of length 1).
e2	a pot object or a character (vector of length 1).

Details

at least one of the two objects must be a pot object. If one of the 2 parameters is a simple string, it is converted as a pot object with no associated format ; therefore, document default document style will be used (see addParagraph).

See Also

addParagraph

Examples

```
pot("My tailor", textProperties(color="red") ) + " is " + pot("rich"
, textProperties(font.weight="bold") )
```

add.pot *add a paraggraph to an existing set of paragraphs of text*

Description

add a paraggraph to an existing set of paragraphs of text (set_of_paragraphs object).

Usage

```
add.pot(x, value)
```

Arguments

x	set_of_paragraphs object
value	pot object to add as a new paragraph

See Also

set_of_paragraphs, pot

Examples

```
pot1 = pot("My tailor", textProperties(color="red") ) + " is " + pot("rich"
, textProperties(font.weight="bold") )
my.pars = set_of_paragraphs( pot1 )
pot2 = pot("Cats", textProperties(color="red") ) + " and " + pot("Dogs"
, textProperties(color="blue") )
my.pars = add.pot( my.pars, pot2 )
```

---

addCodeBlock *Add code block into a document object*

Description

Add a code block into a document object

Usage

```
addCodeBlock(doc, file, text, ...)

## S3 method for class 'docx'
addCodeBlock(doc, file, text, par.properties = parProperties(),
  text.properties = textProperties(color = "#A7947D"), bookmark, ...)

## S3 method for class 'pptx'
addCodeBlock(doc, file, text, par.properties = parProperties(),
  text.properties = textProperties(color = "#A7947D"), append = FALSE, ...)
```

Arguments

doc	document object
file	script file. Not used if text is provided.
text	character vector. The text to parse. Not used if file is provided.
...	further arguments passed to other methods
par.properties	code block paragraph properties. An object of class parProperties
text.properties	code block text properties. An object of class textProperties

bookmark	Only for docx. A character value ; id of the Word bookmark to replace by the script. optional.
append	Only for pptx. boolean default to FALSE. If TRUE, paragraphs will be appened in the current shape instead of beeing sent into a new shape. Paragraphs can only be appended on shape containing paragraphs (i.e. you can not add paragraphs after a FlexTable).

Value

a document object

See Also

docx, pptx

Examples

```
cb <- "ls -a\nwhich -a ls"

options( "ReporteRs-fontsize" = 11 )


# docx example ---------
doc = docx( )
doc <- addCodeBlock( doc, text = cb )
writeDoc( doc, file = "ex_codeblock.docx" )



# pptx example ---------
doc = pptx( )
doc = addSlide( doc, slide.layout = "Title and Content" )
doc <- addCodeBlock( doc, text = cb )
writeDoc( doc, file = "ex_codeblock.pptx" )
```

---

addColumnBreak *Add a column break into a section*

Description

Add a column break into a section

Usage

```
addColumnBreak(doc, ...)

## S3 method for class 'docx'
addColumnBreak(doc, ...)
```

Arguments

doc	document object
...	further arguments passed to other methods

Details

addColumnBreak only works with docx documents.

Value

a document object

See Also

docx, addSection

Examples

```
doc.filename = "add_col_break.docx"
doc = docx( )
doc = addSection(doc, ncol = 2, columns.only = TRUE )
doc = addParagraph( doc = doc, "Text 1.", "Normal" )
doc = addColumnBreak(doc )
doc = addParagraph( doc = doc, "Text 2.", "Normal" )
writeDoc( doc, file = doc.filename )
```

---

addDate *Insert a date*

Description

Insert a date into a document object

Usage

```
addDate(doc, ...)

## S3 method for class 'pptx'
addDate(doc, value, str.format = "%Y-%m-%d", ...)
```

Arguments

doc	document object
...	further arguments passed to other methods
value	character value to add into the date shape of the current slide. optionnal. If missing current date will be used.
str.format	character value to use to format current date (if value is missing).

Details

addDate only works for pptx documents.

Value

a document object

See Also

pptx, addFooter, addPageNumber

Examples

```
doc <- pptx()

doc <- addSlide( doc, slide.layout = "Title and Content" )
## add a date on the current slide
doc = addDate( doc )

doc <- addSlide( doc, slide.layout = "Title and Content" )
## add a page number on the current slide but not
## the default text (slide number)
doc = addDate( doc, "Dummy date" )
```

---

addDocument *Add an external document into a document object*

Description

Add an external document into a document object

Usage

```
addDocument(doc, filename, ...)

## S3 method for class 'docx'
addDocument(doc, filename, ...)
```

Arguments

doc	document object
filename	"character" value, complete filename of the external file
...	further arguments passed to other methods

Details

ReporteRs does only copy the document as an external file. Headers and footers are also imported and displayed. This function is not to be used to merge documents.

Value

a document object

See Also

docx

Examples

```
doc.filename <- "addDocument_example.docx"
# set default font size to 10
options( "ReporteRs-fontsize" = 10 )

doc2embed <- docx( )
img.file <- file.path( Sys.getenv("R_HOME"),
                       "doc", "html", "logo.jpg" )
if( file.exists(img.file) && requireNamespace("jpeg", quietly = TRUE) ){
  dims <- attr( jpeg::readJPEG(img.file), "dim" )

  doc2embed <- addImage(doc2embed, img.file,
                        width = dims[2]/72, height = dims[1]/72)
  writeDoc( doc2embed, file = "external_file.docx" )

  doc <- docx( )
  doc <- addDocument( doc, filename = "external_file.docx" )
  writeDoc( doc, file = doc.filename )
}
```

---

addFlexTable *Insert a FlexTable into a document object*

Description

Insert a FlexTable into a document object

FlexTable can be manipulated so that almost any formatting can be specified. See FlexTable for more details.

Usage

```
addFlexTable(doc, flextable, ...)

## S3 method for class 'docx'
addFlexTable(doc, flextable,
  par.properties = parProperties(text.align = "left"), bookmark, ...)

## S3 method for class 'pptx'
addFlexTable(doc, flextable, offx, offy, width, height, ...)
```

Arguments

doc	document object
flextable	the FlexTable object
...	further arguments passed to other methods
par.properties	paragraph formatting properties of the paragraph that contains the table. An object of class parProperties. Only alignment will be used, if you'd like to add space around a table, specify padding on preceding and or following paragraph.
bookmark	a character vector specifying bookmark id (where to put the table). If provided, table will be add after paragraph that contains the bookmark. If not provided, table will be added at the end of the document.
offx	optional, x position of the shape (top left position of the bounding box) in inches. See details.
offy	optional, y position of the shape (top left position of the bounding box) in inches. See details.
width	optional, width of the shape in inches. See details.
height	optional, height of the shape in inches. See details.

Details

When document is a pptx object, two positioning methods are available.

If arguments offx, offy, width, height are missing, position and dimensions will be defined by the width and height of the next available shape of the slide. This dimensions can be defined in the layout of the PowerPoint template used to create the pptx object.

If arguments offx, offy, width, height are provided, they become position and dimensions of the new shape.

Value

a document object

See Also

FlexTable, docx , pptx

Examples

```
options( "ReporteRs-fontsize" = 11 )

ft_obj <- vanilla.table(mtcars)


# docx example -----
doc = docx( )
doc = addFlexTable( doc, flextable = ft_obj )
writeDoc( doc, file = "add_ft_ex.docx" )



# pptx example -----
doc = pptx( )
doc = addSlide( doc, slide.layout = "Title and Content" )
doc = addFlexTable( doc, flextable = ft_obj )
writeDoc( doc, file = "add_ft_ex.pptx" )
```

---

addFooter *Insert a footer into a document object*

Description

Insert a footer into a document object

Usage

```
addFooter(doc, ...)
```

Arguments

doc	document object
...	further arguments passed to other methods

Details

addFooter only works for pptx documents.

Value

a document object

See Also

pptx

addFooter.pptx *Insert a footer shape into a document pptx object*

Description

Insert a footer shape into the current slide of a pptx object.

Usage

```
## S3 method for class 'pptx'
addFooter(doc, value, ...)
```

Arguments

doc	pptx object
value	character value to add into the footer shape of the current slide.
...	further arguments, not used.

Value

a document object

See Also

pptx, addDate.pptx , addPageNumber.pptx

Examples

```
doc = pptx( )
doc = addSlide( doc, slide.layout = "Title and Content" )
doc = addFooter( doc, "Hi!" )
writeDoc( doc, file = "ex_footer.pptx" )
```

addFooterRow *add footer in a FlexTable*

Description

add a footer row in a FlexTable

Usage

```
addFooterRow(x, value, colspan, text.properties, par.properties,
  cell.properties)
```

Arguments

x	a FlexTable object
value	FlexRow object to insert as a footer row or a character vector specifying labels to use as columns labels.
colspan	integer vector. Optional. Applies only when argument value is a character vector. Vector specifying the number of columns to span for each corresponding value (in values).
text.properties	Optional. textProperties to apply to each cell. Used only if values are not missing.
par.properties	Optional. parProperties to apply to each cell. Used only if values are not missing.
cell.properties	Optional. cellProperties to apply to each cell. Used only if values are not missing.

See Also

FlexTable, addHeaderRow , alterFlexTable

Examples

```
# simple example ----
MyFTable <- FlexTable( data = iris[1:5,1:4] )

# add a footer row with 1 cell that spans four columns
MyFTable <- addFooterRow( MyFTable, value =
  c("a note in table footer"), colspan = 4 )

# another example ----

# create a FlexTable
MyFTable <- FlexTable( data = iris[1:5,1:4] )

# define a complex formatted text
mytext <- pot("*", format = textProperties(
  vertical.align="superscript", font.size = 9) ) +
  pot( " this text is superscripted", format = textProperties(font.size = 9) )

# create a FlexRow - container for 1 cell
footerRow <- FlexRow()
footerRow[1] <- FlexCell( mytext, colspan = 4 )

# finally, add the FlexRow to the FlexTable
MyFTable <- addFooterRow( MyFTable, footerRow )
```

addHeaderRow *add header in a FlexTable*

Description

add a header row in a FlexTable

Usage

```
addHeaderRow(x, value, colspan, text.properties, par.properties,
  cell.properties, first = FALSE)
```

Arguments

x	a FlexTable object
value	FlexRow object to insert as an header row or a character vector specifying labels to use as columns labels.
colspan	integer vector. Optional. Applies only when argument value is a character vector. Vector specifying the number of columns to span for each corresponding value (in values).
text.properties	Optional. textProperties to apply to each cell. Used only if values are not missing. Default is the value of argument header.text.props provided to funtion FlexTable when object has been created
par.properties	Optional. parProperties to apply to each cell. Used only if values are not missing. Default is the value of argument header.par.props provided to funtion FlexTable when object has been created
cell.properties	Optional. cellProperties to apply to each cell. Used only if values are not missing. Default is the value of argument header.cell.props provided to funtion FlexTable when object has been created
first	if TRUE, row will be inserted as first row

See Also

FlexTable, addFooterRow , alterFlexTable

Examples

```
# simple example ----

# set header.columns to FALSE so that default header row is not added in
# the FlexTable object
# We do only want the 4 first columns of the dataset
MyFTable <- FlexTable( data = iris[46:55, ], header.columns = FALSE )
```

```
# add an header row with 3 cells, the first one spans two columns,
# the second one spans two columns and the last one does not span
# multiple columns
MyFTable <- addHeaderRow( MyFTable,
  value = c("Sepal", "Petal", ""),
  colspan = c( 2, 2, 1) )

# add an header row with modified table columns labels
MyFTable <- addHeaderRow( MyFTable,
  value=c("Length", "Width", "Length", "Width", "Species") )

# how to change default formats ----

MyFTable <- FlexTable( data = iris[46:55, ], header.columns = FALSE,
  body.cell.props = cellProperties(border.color="#7895A2"))
# add an header row with table columns labels
MyFTable <- addHeaderRow( MyFTable,
  text.properties = textProperties(color = "#517281", font.weight="bold"),
  cell.properties = cellProperties(border.color="#7895A2"),
  value = c("Sepal Length", "Sepal Width",
    "Sepal Length", "Sepal Width", "Species") )
```

---

addImage *Add an external image into a document object*

Description

Add an external image into a document object

Usage

```
addImage(doc, filename, ...)

## S3 method for class 'docx'
addImage(doc, filename, bookmark,
  par.properties = parProperties(text.align = "center", padding = 5), width,
  height, ...)

## S3 method for class 'pptx'
addImage(doc, filename, offx, offy, width, height, ...)
```

Arguments

doc	document object
filename	"character" value, complete filename of the external image
...	further arguments passed to other methods

bookmark	a character value ; id of the Word bookmark to replace by the image. optional. if missing, image is added at the end of the document.
par.properties	paragraph formatting properties of the paragraph that contains images. An object of class parProperties. It has no effect if doc is a pptx object.
width	image width in inches
height	image height in inches
offx	optional, x position of the shape (top left position of the bounding box) in inches. See details.
offy	optional, y position of the shape (top left position of the bounding box) in inches See details.

Details

Arguments width and height can be defined with functions png::readPNG, jpeg::readJPEG or bmp::read.bmp.

When document object is a pptx, width and height are not mandatory. By default, image is added to the next free 'content' shape of the current slide. See slide.layouts.pptx to view the slide layout.

If arguments offx and offy are missing, position is defined as the position of the next available shape of the slide. This dimensions can be defined in the layout of the PowerPoint template used to create the pptx object.

Value

a document object

See Also

docx, pptx

Examples

```
# get rlogo
img.file <- file.path( Sys.getenv("R_HOME"), "doc", "html", "logo.jpg" )

# tests to use later
has_img <- file.exists( img.file )
has_jpeg <- requireNamespace("jpeg", quietly = TRUE)
has_wmf <- exists("win.metafile")
is_sunos <- tolower(Sys.info()[["sysname"]]) == "sunos"

# create a wmf file if possible
if( has_wmf ){
  win.metafile(filename = "image.wmf", width = 5, height = 5 )
  barplot( 1:6, col = 2:7)
  dev.off()
}
```

```
# Image example for MS Word -------

doc <- docx()

if( has_img && has_jpeg ){
  dims <- attr( jpeg::readJPEG(img.file), "dim" )
  doc <- addImage(doc, img.file, width = dims[2]/72,
    height = dims[1]/72)
}

if( has_wmf ){
  doc <- addImage(doc, "image.wmf", width = 5, height = 5 )
}

writeDoc( doc, file = "ex_add_image.docx" )


# Image example for MS PowerPoint -------

if( !is_sunos ){

doc <- pptx()

if( has_img && has_jpeg ){
  doc <- addSlide( doc, "Title and Content" )
  dims <- attr( jpeg::readJPEG(img.file), "dim" )
  doc <- addImage(doc, img.file, width = dims[2]/72,
    height = dims[1]/72)
}
if( has_wmf ){
  doc <- addSlide( doc, "Title and Content" )
  doc <- addImage(doc, "image.wmf", width = 5, height = 5 )
}

writeDoc( doc, file = "ex_add_image.pptx" )
}
```

---

addPageBreak *Add a page break into a document object*

Description

Add a page break into a document object

Usage

```
addPageBreak(doc, ...)
```

```
## S3 method for class 'docx'
addPageBreak(doc, ...)
```

Arguments

doc	document object
...	further arguments passed to other methods

Details

addPageBreak only works with docx documents.

See addPageBreak.docx for examples.

Value

a document object

See Also

docx

Examples

```
doc = docx( title = "title" )
doc = addPageBreak( doc )
```

---

addPageNumber *Insert a page number into a document object*

Description

Insert a page number into a document object

Usage

```
addPageNumber(doc, ...)
```

Arguments

doc	document object
...	further arguments passed to other methods

Details

addPageNumber only works with pptx documents.

See addPageNumber.pptx for examples.

Value

a document object

See Also

pptx, addPageNumber.pptx

---

addPageNumber.pptx *Insert a page number shape into a document pptx object*

Description

Insert a page number shape into the current slide of a pptx object.

Usage

```
## S3 method for class 'pptx'
addPageNumber(doc, value, ...)
```

Arguments

doc	pptx object
value	character value to add into the page number shape of the current slide. optionnal. If missing current slide number will be used.
...	further arguments, not used.

Value

a pptx document object

See Also

addPageNumber, addDate

Examples

```
doc.filename <- "add_page_number_example.pptx"
doc <- pptx( title = "title" )
doc <- addSlide( doc, slide.layout = "Title Slide" )
# add a page number on the current slide ---
doc <- addPageNumber( doc )
doc <- addSlide( doc, slide.layout = "Title and Content" )
# add a page number with free text ----
doc <- addPageNumber( doc, value = "Page number text")
writeDoc( doc, file = doc.filename )
```

---

addParagraph *Add a paragraph into a document object*

Description

Add a paragraph into a document object

Usage

```
addParagraph(doc, value, ...)

## S3 method for class 'docx'
addParagraph(doc, value, stylename, bookmark,
  par.properties = parProperties(), restart.numbering = FALSE, ...)

## S3 method for class 'pptx'
addParagraph(doc, value, offx, offy, width, height,
  par.properties, append = FALSE, restart.numbering = FALSE, ...)
```

Arguments

doc	document object
value	text to add to the document as paragraphs: an object of class pot or set_of_paragraphs or a character vector.
...	further arguments passed to other methods
stylename	value of the named style to apply to paragraphs in the docx document. Expected value is an existing stylename of the template document used to create the docx object. This argument can only be used when value is a character vector. see styles.docx.
bookmark	a character value ; id of the Word bookmark to replace by the table. optional.
par.properties	parProperties to apply to paragraphs. When used in Word documents, it will only be used if stylename if missing. When used in PowerPoint documents, it will be used if offx or offy is used.

restart.numbering
: boolean value. If TRUE, next numbered list counter will be set to 1.

offx
: optional, x position of the shape (top left position of the bounding box) in inches. See details.

offy
: optional, y position of the shape (top left position of the bounding box) in inches. See details.

width
: optional, width of the shape in inches. See details.

height
: optional, height of the shape in inches. See details.

append
: boolean default to FALSE. If TRUE, paragraphs will be appened in the current shape instead of beeing sent into a new shape. Paragraphs can only be appended on shape containing paragraphs (i.e. you can not add paragraphs after a FlexTable). Applies to only pptx objects.

Details

a paragraph is a set of text that ends with an end of line. Read pot to see how to get different font formats. If an end of line is required, a new paragraph is required.

When document is a pptx object, two positioning methods are available.

If arguments offx, offy, width, height are missing, position and dimensions will be defined by the width and height of the next available shape of the slide. This dimensions can be defined in the layout of the PowerPoint template used to create the pptx object.

If arguments offx, offy, width, height are provided, they become position and dimensions of the new shape.

Also, when document is a pptx object, shading and border settings of argument par.properties will have no effect.

Argument par.properties is only used when value is a pot or a set_of_paragraphs.

If character values are used to fill slides, parameter append will be ignored.

Value

a document object

See Also

docx, pptx, pot, textProperties, parProperties, list.settings

Examples

```
# define some text
sometext <- c( "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
  "In sit amet ipsum tellus. Vivamus dignissim arcu sit amet faucibus auctor.",
  "Quisque dictum tristique ligula.")

# "My tailor is rich" with formatting on some words
pot1 <- pot("My tailor", textProperties(color = "red" ) ) +
  " is " + pot("rich", textProperties(shading.color = "red",
     font.weight = "bold" ) )
```

```
# "Cats and dogs" with formatting on some words
pot2 = pot("Cats", textProperties(color = "red" ) ) +
  " and " +
  pot("dogs", textProperties( color = "blue" ),
      hyperlink = "http://www.wikipedia.org/" )

# create a set of paragraphs made of pot1 and pot2
my.pars <- set_of_paragraphs( pot1, pot2 )

# docx example ---------

doc.filename = "ex_paragraph.docx"
doc <- docx()
styles(doc)

doc <- addTitle( doc, "Title example 1", level = 1 )
# Add "Hello World" into the document doc
doc <- addParagraph(doc, "Hello Word", stylename = "Normal" )

doc <- addTitle( doc, "Title example 2", level = 1 )

# add sometext with stylename BulletList
doc <- addParagraph( doc, value = sometext, stylename="BulletList" )

doc <- addTitle( doc, "Title example 3", level = 1 )

doc <- addParagraph( doc, value = my.pars, par.properties = parCenter() )

writeDoc( doc, file = doc.filename )


# pptx example -------
doc.filename = "ex_paragraph.pptx"
doc <- pptx()
doc <- addSlide(doc, "Title and Content")
# Add "Hello World" into the document doc
doc <- addParagraph(doc, "Hello Word" )

doc <- addSlide(doc, "Title and Content")
doc <- addParagraph( doc, value = my.pars )
# Add my.pars into the document doc
doc <- addParagraph(doc, my.pars, offx = 3, offy = 3,
  width = 2, height = 0.5,
  par.properties=parProperties(text.align="center", padding=0) )
doc <- addSlide(doc, "Title and Content")
# Add my.pars into the document doc
doc <- addParagraph(doc, my.pars,
  par.properties=parProperties(text.align="center", padding=24) )

writeDoc( doc, file = doc.filename )
```

addParagraph.Footnote *Insert a paragraph into a Footnote object*

Description

Insert paragraph(s) of text into a Footnote. To create a Footnote made of several paragraphs with different parProperties, add sequentially paragraphs with their associated parProperties objects with this function.

Usage

```
## S3 method for class 'Footnote'
addParagraph(doc, value, par.properties = parProperties(),
  ...)
```

Arguments

doc	Footnote object where to add paragraphs.
value	text to add to the document as paragraphs: an object of class pot or set_of_paragraphs or a character vector.
par.properties	parProperties to apply to paragraphs.
...	further arguments, not used.

Value

an object of class Footnote.

See Also

Footnote, parProperties, pot , set_of_paragraphs

addPlot *Add a plot into a document object*

Description

Add a plot into a document object

Usage

```
addPlot(doc, fun, pointsize = 12, vector.graphic = FALSE, ...)

## S3 method for class 'docx'
addPlot(doc, fun, pointsize = getOption("ReporteRs-fontsize"),
  vector.graphic = FALSE, width = 6, height = 6,
  fontname_serif = "Times New Roman", fontname_sans = "Calibri",
  fontname_mono = "Courier New", fontname_symbol = "Symbol",
  editable = TRUE, bookmark, par.properties = parProperties(text.align =
  "center", padding = 5), bg = "transparent", ...)

## S3 method for class 'pptx'
addPlot(doc, fun, pointsize = 11, vector.graphic = TRUE,
  fontname_serif = "Times New Roman", fontname_sans = "Calibri",
  fontname_mono = "Courier New", fontname_symbol = "Symbol",
  editable = TRUE, offx, offy, width, height, bg = "transparent", ...)
```

Arguments

doc	document object
fun	plot function. The function will be executed to produce graphics. For grid or lattice or ggplot object, the function should just be print and an extra argument x should specify the object to plot. For traditionnal plots, the function should contain plot instructions. See examples.
pointsize	the default pointsize of plotted text in pixels, default to 12 pixels.
vector.graphic	logical scalar, if TRUE, vector graphics are produced instead, PNG images if FALSE.
...	further arguments passed to or from other methods. See details.
width	plot width in inches (default value is 6).
height	plot height in inches (default value is 6).
fontname_serif, fontname_sans, fontname_mono, fontname_symbol	font names for font faces. Use fonts available on operating system.
editable	logical value - if TRUE vector graphics elements (points, text, etc.) are editable.
bookmark	id of the Word bookmark to replace by the plot. optional. bookmark is a character vector specifying bookmark id to replace by the plot(s). If provided, plot(s) will replace the paragraph that contains the bookmark. If not provided, plot(s) will be added at the end of the document.
par.properties	paragraph formatting properties of the paragraph that contains plot(s). An object of class parProperties
bg	the initial background colour.
offx, offy	optional. x and y position of the shape (left top position of the bounding box) in inches. See details.

Details

Plot parameters are specified with the ... argument. However, the most convenient usage is to wrap the plot code into a function whose parameters will be specified as '...'.

If you want to add ggplot2 or lattice plot, use `print` function.

`vector.graphic`: DrawingML instructions will be produced for `docx` and `pptx` objects. Don't use vector graphics if document is a docx and MS Word version used to open the document is 2007.

When document is a `pptx` object, two positioning methods are available.

If arguments offx, offy, width, height are missing, position and dimensions will be defined by the width and height of the next available shape of the slide. This dimensions can be defined in the layout of the PowerPoint template used to create the `pptx` object.

If arguments offx, offy, width, height are provided, they become position and dimensions of the new shape.

Value

a document object

See Also

docx, pptx

Examples

```
is_sunos <- tolower(Sys.info()[["sysname"]]) == "sunos"

options( "ReporteRs-fontsize" = 11 )


# plot example for docx -----

doc = docx( )
doc = addPlot( doc, fun = function() barplot( 1:6, col = 2:7),
  vector.graphic = TRUE, width = 5, height = 7,
  par.properties = parProperties(text.align = "center")
  )
writeDoc( doc, file = "ex_plot.docx" )



# plot example for pptx -----

doc = pptx( )
doc = addSlide( doc, slide.layout = "Title and Content" )

doc = addPlot( doc, fun = function() barplot( 1:6, col = 2:7),
  vector.graphic = TRUE, width = 5, height = 4 )
if( !is_sunos ){
  doc = addPlot( doc,
```

```
    fun = function() barplot( 1:6, col = 2:7),
    vector.graphic = FALSE,
    offx = 7, offy = 0,
    width = 3, height = 2
    )
}

writeDoc( doc, file = "ex_plot.pptx" )
```

---

addRScript *Add R script into a document object*

Description

Add R script into a document object

Usage

```
addRScript(doc, rscript, file, text, ...)

## S3 method for class 'docx'
addRScript(doc, rscript, file, text, bookmark,
  par.properties = parProperties(), ...)

## S3 method for class 'pptx'
addRScript(doc, rscript, file, text, append = FALSE, ...)
```

Arguments

doc	document object
rscript	an object of class RScript. Not used if file or text is provided.
file	R script file. Not used if text or rscript is provided.
text	character vector. The text to parse. Not used if file or rscript is provided.
...	further arguments passed to other methods
bookmark	a character value ; id of the Word bookmark to replace by the script. optional.
par.properties	paragraph formatting properties of the paragraphs that contain rscript. An object of class parProperties
append	boolean default to FALSE. If TRUE, paragraphs will be appened in the current shape instead of beeing sent into a new shape. Paragraphs can only be appended on shape containing paragraphs (i.e. you can not add paragraphs after a FlexTable).

Details

You have to one of the following argument: file or text or rscript.

Value

a document object

See Also

docx, pptx

Examples

```
# docx example -----------
doc.filename = "ex_rscript.docx"
doc <- docx()
doc <- addRScript(doc, text = "x = rnorm(100)" )
writeDoc( doc, file = doc.filename )


# pptx example -----------
doc.filename = "ex_rscript.pptx"
doc <- pptx()
doc <- addSlide(doc, "Title and Content")
doc <- addRScript(doc, text = "x = rnorm(100)" )
writeDoc( doc, file = doc.filename )
```

---

addSection *Add a section into a document object*

Description

Add a section into a document object

Usage

```
addSection(doc, ...)

## S3 method for class 'docx'
addSection(doc, landscape = FALSE, ncol = 1,
  space_between = 0.3, columns.only = FALSE, ...)
```

Arguments

doc	document object
...	further arguments passed to other methods
landscape	logical value. Specify TRUE to get a section with horizontal page.
ncol	integer number to specify how many columns the section should contains.
space_between	width in inches of the space between columns of the section.
columns.only	logical value, if set to TRUE, no break page will (continuous section).

Details

addSection only works with docx documents.

It lets you change document orientation and split new content along 2 or more columns. The function requires you to add a section before and after the item(s) that you want to be on a landscape and/or multicolumns mode page.

Value

a document object

See Also

docx

Examples

```
doc.filename = "addSection.docx"
doc <- docx()
doc = addSection(doc, landscape = TRUE, ncol = 2 )
doc = addPlot( doc = doc, fun = function() {
barplot( 1:8, col = 1:8 )
}, width = 3, height = 3, pointsize = 5)

doc = addColumnBreak(doc )
doc = addFlexTable(doc, FlexTable( iris[1:10,] ) )

doc = addSection(doc, ncol = 2 )
doc = addParagraph( doc = doc, "Text 1.", "Normal" )
doc = addColumnBreak(doc )
doc = addParagraph( doc = doc, "Text 2.", "Normal" )


doc = addSection(doc, ncol = 2, columns.only = TRUE )
doc = addFlexTable(doc, FlexTable(iris[1:10,] ) )
doc = addColumnBreak(doc )
doc = addParagraph( doc = doc, "Text 3.", "Normal" )


doc = addSection( doc )
doc = addFlexTable(doc, FlexTable(mtcars, add.rownames = TRUE) )
doc = addParagraph( doc = doc, "Text 4.", "Normal" )
writeDoc(doc, doc.filename)
```

addSlide *Add a slide into a document object*

Description

Add a slide into a document object

Usage

```
addSlide(doc, ...)

## S3 method for class 'pptx'
addSlide(doc, slide.layout, bookmark, ...)
```

Arguments

doc	document object
...	further arguments passed to other methods
slide.layout	layout name of the slide to create. See slide.layouts.pptx
bookmark	"integer" page number to specify where slide has to be replaced with a new empty one.

Details

addSlide only works with pptx documents. See addSlide.pptx for examples.

This function is a key function ; if no slide has been added into the document object no content (tables, plots, images, text) can be added.

If creating a slide of type "Title and Content", only one content can be added because there is only one content shape in the layout. If creating a slide of type "Two Content", two content can be added because there are 2 content shapes in the layout.

Content shapes are boxes with dotted borders that hold content in its place on a slide layout. If you need a new layout, create it in PowerPoint :

On the View tab, in the Presentation Views group, click Slide Master.

Function slide.layouts returns available layout names of the template used when pptx object has been created. It is important to know that when using addParagraph.pptx, paragraph and defaut font formats will be defined by the properties of the shape of the slide.layout where content will be added. For example, if you set the shape formatting properties to a 'no bullet', paragraphs of text won't have any bullet.

Also when using addPlot, plot dimensions will be the shape dimensions. It means that if you want to change plot dimensions , this has to be done in the PowerPoint template used when creating the pptx object.

Value

a document object

See Also

pptx, slide.layouts

Examples

```
doc.filename = "addSlide_example.pptx"
doc <- pptx()
doc <- addSlide(doc, "Title and Content")
doc <- addTitle(doc, "Title example")
writeDoc( doc, file = doc.filename )



# demo slide replacement --------

# define 2 FlexTables
ft1 = vanilla.table( mtcars[1:6,] , add.rownames = TRUE )
ft2 = vanilla.table( iris[1:10,], add.rownames = TRUE )

# create an doc to be used as template later
mydoc = pptx( )
mydoc = addSlide( mydoc, slide.layout = "Title and Content")
mydoc = addTitle( mydoc, "a table")
mydoc = addFlexTable( mydoc, ft1 )
mydoc = addSlide( mydoc, slide.layout = "Title and Content")
mydoc = addTitle( mydoc, "some text")
mydoc = addParagraph( mydoc, "text example" )
writeDoc( mydoc, "template_example.pptx" )

# use file pp_template_example.pptx as template
# and replace slide 1
mydoc = pptx(template = "template_example.pptx" )
mydoc = addSlide( mydoc, slide.layout = "Title and Content", bookmark = 1)
mydoc = addTitle( mydoc, "a new table")
mydoc = addFlexTable( mydoc, ft2 )
writeDoc( mydoc, "slide_replacement.pptx" )
```

---

addSubtitle *Add a subtitle shape into a document object*

Description

Add a subtitle shape into a document object

Add a addSubtitle shape into a pptx object.

Usage

```
addSubtitle(doc, ...)

## S3 method for class 'pptx'
addSubtitle(doc, value, ...)
```

Arguments

doc	document object
...	further arguments passed to other methods
value	"character" value to use as subtitle text

Details

Subtitle shape only exist in slide of type 'Title Slide'.

Value

a document object

See Also

pptx

Examples

```
doc.filename = "addSubtitle_example.pptx"
doc <- pptx()
doc <- addSlide( doc, slide.layout = "Title Slide" )
#set the main title
doc <- addTitle( doc, "Presentation title" )
#set the sub-title
doc <- addSubtitle( doc , "This document is generated with ReporteRs.")
writeDoc( doc, file = doc.filename )
```

---

addTitle *Add a title*

Description

Add a title into a document object

Usage

```
addTitle(doc, value, ...)

## S3 method for class 'docx'
addTitle(doc, value, level = 1, ...)

## S3 method for class 'pptx'
addTitle(doc, value, ...)
```

Arguments

doc	document object
value	"character" value to use as title text
...	further arguments passed to or from other methods..
level	"integer" positive value to use as heading level. 1 for title1, 2 for title2, etc. Default to 1.

Details

Function addTitle when used with docx object needs to know which style correspond to which title level (1 ; 1.1 ; 1.1.1 ; etc.). When a template is read, ReporteRs tries to guess what are the available styles (english, french, chinese, etc.). If styles for titles has not been detected you will see the following error when addTitle is being called:

```
You must defined title styles via map_title first.
```

As the error message points out, you have to call the function ‘map_title‘ to indicate which available styles are meant to be used as title styles.

To add a title into a pptx object you only have to specify the text to use as title. There is no level concept.

Value

a document object

See Also

map_title

Examples

```
# Title example for MS Word -------

doc.filename = "ex_add_title.docx"
doc <- docx()
doc <- addTitle( doc, "Title example 1", level = 1 )
doc <- addTitle( doc, "Title example 2", level = 1 )
writeDoc( doc, file = doc.filename )
```

```
# Title example for PowerPoint -------
doc.filename = "ex_add_title.pptx"
doc <- pptx()
doc <- addSlide(doc, "Title and Content")
doc <- addTitle(doc, "Title example")
writeDoc( doc, file = doc.filename )
```

---

addTOC *Add a table of contents into a document object*

Description

Add a table of contents into a document object

Usage

```
addTOC(doc, ...)

## S3 method for class 'docx'
addTOC(doc, stylename, level_max = 3, ...)
```

Arguments

doc	document object
...	further arguments passed to other methods
stylename	optional. Stylename in the document that will be used to build entries of the TOC.
level_max	max title level to show in the TOC (defaut to 3, from title1 to title3).

Details

When working with docx object:

If stylename is not used, a classical table of content will be produced.
If stylename is used, a custom table of contents will be produced, pointing to entries that have been formated with stylename. For example, this can be used to produce a toc with only plots.

Value

a document object

See Also

docx, styles, addTitle, addParagraph

Examples

```
# simple TOC -----

# Create a new document
doc = docx( title = "title" )
#leave the first page blank and add a page break
doc = addPageBreak(doc)
# add a TOC (to be refresh when document is opened)
# and add a page break
doc = addTOC(doc)
doc = addPageBreak(doc)

# add titles that will be entries in the TOC
doc = addTitle( doc, "My first title", level = 1 )
doc = addTitle( doc, "My second title", level = 1 )

# Write the object in file "addTOC_example1.docx"
writeDoc( doc, "addTOC_example1.docx" )

# custom TOC -----

doc = docx( title = "title" )

#leave the first page blank and add a page break
doc = addPageBreak(doc)

doc = addTitle( doc, "Plots", level = 1 )
doc = addPlot( doc, fun = plot,
  x = rnorm( 100 ),y = rnorm (100 ),
  main = "base plot main title"
)

doc = addParagraph( doc, value="graph example 1", stylename = "rPlotLegend" )

if( requireNamespace("ggplot2", quietly = TRUE) ){

  myplot = ggplot2::qplot(Sepal.Length, Petal.Length,
    data = iris, color = Species,
    size = Petal.Width, alpha = I(0.7))
  doc = addPlot( doc = doc, fun = print,
    x = myplot #this argument MUST be named, print is expecting argument 'x'
  )
  doc = addParagraph( doc, value="graph example 2",
    stylename = "rPlotLegend" )

}

# Because we used "rPlotLegend" as legend in plot
# , addTOC will use this stylename to define
# entries in the generated TOC
doc = addTOC(doc, stylename = "rPlotLegend")
```

```
# Write the object in file "addTOC_example2.docx"
writeDoc( doc, "addTOC_example2.docx" )
```

`as.FlexTable` *R tables as FlexTables*

Description

Get a `FlexTable` object from an R object.

Usage

```
as.FlexTable(x, ...)
```

Arguments

`x`	object to get `FlexTable` from
`...`	further arguments passed to other methods

Value

a `FlexTable` object

See Also

`FlexTable`

`as.FlexTable.sessionInfo`
get FlexTable from a sessionInfo object

Description

Get a `FlexTable` object from a `sessionInfo` object.

Usage

```
## S3 method for class 'sessionInfo'
as.FlexTable(x, locale = TRUE, ...)
```

Arguments

`x`	`sessionInfo` object to get `FlexTable` from
`locale`	show locale information?
`...`	further arguments, not used.

Value

a FlexTable object

Examples

```
ft <- as.FlexTable( sessionInfo() )
```

---

as.html *get HTML code*

Description

Get HTML code in a character vector.

Usage

```
as.html(object, ...)
```

Arguments

object	object to get HTML from
...	further arguments passed to other methods

Details

See FlexTable for examples.

Value

a character value

See Also

FlexTable

as.html.FlexTable *get HTML code from a FlexTable*

Description

get HTML code from a FlexTable

Usage

```
## S3 method for class 'FlexTable'
as.html(object, ...)
```

Arguments

object	the FlexTable object
...	further arguments passed to other methods

Value

a character value

See Also

FlexTable

as.html.pot *get HTML code from a pot*

Description

get HTML code from a pot

Usage

```
## S3 method for class 'pot'
as.html(object, ...)
```

Arguments

object	the pot object
...	further arguments passed to other methods

Value

a character value

See Also

pot

Examples

```
my_pot = pot("My tailor", textProperties(color="red") ) + " is " + pot("rich"
, textProperties(font.weight="bold") )
as.html( my_pot )
```

---

borderProperties *border properties object*

Description

create a border properties object.

Usage

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

## S3 method for class 'borderProperties'
chprop(object, color, style, width, ...)

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

## S3 method for class 'borderProperties'
as.character(x, ...)
```

Arguments

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

Details

Get a modified version of a borderProperties with chprop.

See Also

alterFlexTable, setFlexTableBorders, shortcut_properties

Examples

```
borderProperties()
borderProperties(color="orange", style="solid", width=1)
borderProperties(color="gray", style="dotted", width=1)

# update borderProperties --------
x = borderProperties()
chprop(x, color="orange", style="dashed", width=1)
chprop(x, width=5)
```

`cellProperties` *Cell formatting properties*

Description

Create a `cellProperties` object that describes cell formatting properties. This objects are used by `FlexTable`.

Usage

```
cellProperties(padding, border.width, border.style, border.color, border.bottom,
  border.left, border.top, border.right, border.bottom.color = "black",
  border.bottom.style = "solid", border.bottom.width = 1,
  border.left.color = "black", border.left.style = "solid",
  border.left.width = 1, border.top.color = "black",
  border.top.style = "solid", border.top.width = 1,
  border.right.color = "black", border.right.style = "solid",
  border.right.width = 1, vertical.align = "middle", padding.bottom = 0,
  padding.top = 0, padding.left = 0, padding.right = 0,
  background.color = "transparent", text.direction = "lrtb")

## S3 method for class 'cellProperties'
chprop(object, border.bottom, border.left, border.top,
  border.right, padding, border.bottom.color, border.bottom.style,
  border.bottom.width, border.left.color, border.left.style, border.left.width,
  border.top.color, border.top.style, border.top.width, border.right.color,
  border.right.style, border.right.width, vertical.align, padding.bottom,
  padding.top, padding.left, padding.right, background.color, text.direction,
  ...)

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

Arguments

`padding`	cell padding - 0 or positive integer value. Argument `padding` overwrites arguments `padding.bottom`, `padding.top`, `padding.left`, `padding.right`.

`border.width` border width - 0 or positive integer value. Argument `border.width` overwrites arguments `border.bottom.width`, `border.top.width` , `border.left.width`, `border.right.width`.

`border.style` border style - a single character value, expected value is one of "none" , "solid", "dotted", "dashed". Argument `border.style` overwrites arguments `border.bottom.style`, `border.top.style`, `border.left.style` , `border.right.style`.

`border.color` border color - a single character value specifying a valid color (e.g. "#000000" or "black"). Argument `border.color` overwrites arguments `border.bottom.color` , `border.top.color`, `border.left.color`, `border.right.color`.

`border.bottom` `borderProperties` for bottom border. overwrite all border.bottom.* if specified.

`border.left` `borderProperties` for left border. overwrite all border.left.* if specified.

`border.top` `borderProperties` for top border. overwrite all border.top.* if specified.

`border.right` `borderProperties` for right border. overwrite all border.right.* if specified.

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

`border.bottom.style`
border bottom style - a single character value, expected value is one of "none", "solid", "dotted", "dashed".

`border.bottom.width`
border bottom width - 0 or positive integer value

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

`border.left.style`
border left style - a single character value, expected value is one of "none", "solid", "dotted", "dashed".

`border.left.width`
border left width - 0 or positive integer value

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

`border.top.style`
border top style - a single character value, expected value is one of "none", "solid", "dotted", "dashed".

`border.top.width`
border top width - 0 or positive integer value

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

`border.right.style`
border right style - a single character value, expected value is one of "none", "solid", "dotted", "dashed".

`border.right.width`
: border right width - 0 or positive integer value

`vertical.align`
: cell content vertical alignment - a single character value , expected value is one of "center" or "top" or "bottom"

`padding.bottom`
: cell bottom padding - 0 or positive integer value.

`padding.top`
: cell top padding - 0 or positive integer value.

`padding.left`
: cell left padding - 0 or positive integer value.

`padding.right`
: cell right padding - 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".

`object`
: `cellProperties` object to update

`...`
: further arguments - not used

`x`
: `cellProperties` object to print

Details

Default values are:

- `border.bottom.color` "black"
- `border.bottom.style` "solid"
- `border.bottom.width` 1
- `border.left.color` "black"
- `border.left.style` "solid"
- `border.left.width` 1
- `border.top.color` "black"
- `border.top.style` "solid"
- `border.top.width` 1
- `border.right.color` "black"
- `border.right.style` "solid"
- `border.right.width` 1
- `vertical.align` "middle"
- `padding.bottom` 1
- `padding.top` 1
- `padding.left` 1
- `padding.right` 1
- `background.color` "white"
- `text.direction` "lrtb"

Get a modified version of a `cellProperties` with `chprop`.

See Also

borderProperties, FlexTable, shortcut_properties

Examples

```
cellProp01 <- cellProperties( border.color = "gray", border.width = 2 )
cellProp02 <- cellProperties(border.left.width = 0, border.right.width = 0,
  border.bottom.width = 2, border.top.width = 0,
  padding.bottom = 2, padding.top = 2,
  padding.left = 2, padding.right = 2 )
cellProp <- cellProperties()

chprop( cellProp, border.bottom.color = "#8A949B" )
chprop( cellProp, border.color = "#8A949B" )
chprop( cellProp, border.right.width = 2 )
chprop( cellProp, border.width = 2 )
chprop( cellProp, padding = 5 )
chprop( cellProp,
  border.bottom = borderProperties( style = "dotted" ) )
chprop( cellProp, border.style = "dotted" )
```

---

chprop *Change a formatting properties object*

Description

Change a formatting properties object

Usage

```
chprop(object, ...)
```

Arguments

object	formatting properties object
...	further arguments passed to other methods

Details

See chprop.textProperties or chprop.parProperties or chprop.cellProperties for examples.

Value

a formatting properties object

See Also

cellProperties, textProperties , parProperties

chprop.FlexTable *format FlexTable*

Description

Format a FlexTable object.

Usage

```
## S3 method for class 'FlexTable'
chprop(object, value, i, j, to = "body", side = "top",
  ...)
```

Arguments

object	the FlexTable object
value	a formatting properties object (textProperties, parProperties, borderProperties, cellProperties)
i	vector (integer index, row.names values or boolean vector) for rows selection.
j	vector (integer index, col.names values or boolean vector) for columns selection.
to	specify on which part of the FlexTable to apply the value, must be one of the following values "body" (default) or "header" or "footer"
side	used only when value is a borderProperties, specify on which side to apply the properties. It must be one of "bottom", "top", "left", "right".
...	unused

Examples

```
my_ft <- vanilla.table( head( iris, n = 5 ) )
my_ft <- chprop( my_ft, textBoldItalic(), i = 1, to = "header" )
my_ft <- chprop( my_ft, parCenter(), j = 5 )
my_ft <- chprop( my_ft, borderSolid(color = "red"), i = 5, side = "bottom" )
```

CodeBlock *Code Block Object*

Description

Code Block Object. A code block object is a block of text treated as verbatim text in a document object.

Usage

```
CodeBlock(file, text, text.properties = textProperties(color = "#A7947D"),
  par.properties = parProperties(text.align = "left", shading.color =
  "#5FB0B8"))
```

Arguments

file	script file. Not used if text is provided.
text	character vector. The text to parse. Not used if file is provided.
text.properties	default textProperties object
par.properties	default parProperties object

See Also

addCodeBlock

Examples

```
cb_example <- CodeBlock( text = "ls -a\nwhich -a ls" )
```

---

colorProperties *color properties object*

Description

create a color properties object.

Usage

```
colorProperties(value = "black")

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

## S3 method for class 'colorProperties'
as.character(x, ...)
```

Arguments

value	valid color
x	a colorProperties object
...	unused

deleteBookmark *delete a bookmark into a docx object*

Description

delete a bookmark into a docx object

Usage

```
deleteBookmark(doc, bookmark)
```

Arguments

doc	Object of class docx
bookmark	a character vector specifying bookmark id to delete

See Also

docx

deleteBookmarkNextContent *delete first content after a bookmark into a docx object*

Description

delete first content after a bookmark into a docx object

Usage

```
deleteBookmarkNextContent(doc, bookmark)
```

Arguments

doc	Object of class docx
bookmark	a character vector specifying bookmark id to delete

See Also

docx

docx *Create Microsoft Word document object representation*

Description

Create a docx object

Usage

```
docx(title = "untitled", template, empty_template = FALSE,
  list.definition = getOption("ReporteRs-list-definition"))

## S3 method for class 'docx'
dim(x)

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

Arguments

title	"character" value: title of the document (in the doc properties).
template	"character" value, it represents the filename of the docx file used as a template.
empty_template	wether the content of the template should be clear or not.
list.definition	a list definition to specify how ordered and unordered lists have to be formated. See list.settings. Default to getOption("ReporteRs-list-definition").
x	a docx objet
...	unused

Details

Several methods can used to send R output into an object of class docx.

- addTitle add titles
- addParagraph add text
- addPlot add plots
- addFlexTable add tables. See FlexTable
- addImage add external images
- addTOC add table of content
- addPageBreak add page break
- addSection add section (for landscape orientation)
- addDocument add an external docx file into a docx object.

R outputs (tables, plots, paragraphs and images) can be inserted (and not added at the end) in a document if a bookmark exists in the template file.

Once object has content, user can write the docx into a ".docx" file, see writeDoc.

dim returns page width and height and page margins of a docx object.

print print informations about an object of class docx.

Value

an object of class docx.

Note

Word 2007-2013 (*.docx) file formats are the only supported files.
Document are manipulated in-memory ; a docx's document is not written to the disk unless the writeDoc method has been called on the object.

References

Wikipedia: Office Open XML
http://en.wikipedia.org/wiki/Office_Open_XML

See Also

pptx

Examples

```
# set default font size to 10
options( "ReporteRs-fontsize" = 10 )

# Create a new document
doc <- docx( title = "title" )

# display available styles
styles( doc )

# add title
doc <- addParagraph( doc, "Document title", stylename = "TitleDoc" )

# add a paragraph
doc <- addParagraph( doc , "This document is generated with ReporteRs."
  , stylename="Citationintense")

# add page break
doc <- addPageBreak( doc )

# add a title
doc <- addTitle( doc, "Table of contents", level =  1 )

################ TOC DEMO ################
```

```
# add a table of content
doc <- addTOC( doc )

# add page break and then tables of contents for produced plots and tables
doc <- addPageBreak( doc )
doc <- addTitle( doc, "List of graphics", level =  1 )
doc <- addTOC( doc, stylename = "rPlotLegend" )
doc <- addTitle( doc, "List of tables", level =  1 )
doc <- addTOC( doc, stylename = "rTableLegend" )

# add page break
doc <- addPageBreak( doc )

################ TEXT DEMO ################

# add a title
doc <- addTitle( doc, "Text demo", level =  1 )


sometext = c( "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
  , "In sit amet ipsum tellus. Vivamus dignissim arcu sit."
  , "Quisque dictum tristique ligula."
)
# add simple text with 'Normal' style
doc <- addParagraph( doc, value = sometext, stylename="Normal" )
# add simple text with 'BulletList' style
doc <- addParagraph( doc, value = sometext, stylename="BulletList" )

# Add "My tailor is rich" and "Cats and Dogs"
# format some of the pieces of text
pot1 = pot("My tailor", textProperties(color="red",
shading.color = "#CCCCCC" ) ) + " is " + pot("rich",
textProperties(font.weight="bold") )
pot2 = pot("Cats", textProperties(color="red" )) + " and " + pot("Dogs",
textProperties(color="blue" ) )
doc <- addParagraph(doc, set_of_paragraphs( pot1, pot2 ),
stylename = "Normal" )

doc <- addParagraph(doc, "Silentium tractibus per minimis ne excita
ut temptentur generalibus quam primordiis per clades post delictis
iuge exitium silentium per et.",
  par.properties = parProperties( padding.left = 25,
    padding.right = 25) )


doc <- addParagraph(doc,
  pot("Gallus necem refert singula modum quae est quae quorum leo.",
format = textItalic( ) ),
  par.properties = parProperties(list.style = "blockquote") )


ordered.list.level1 = parProperties(list.style = "ordered", level = 1 )
ordered.list.level2 = parProperties(list.style = "ordered", level = 2 )
```

```
doc <- addParagraph( doc, value = sometext,
par.properties = ordered.list.level1 )
doc <- addParagraph( doc, value = sometext,
par.properties = ordered.list.level2 )

################ PLOT DEMO ################
# load ggplot2
if( requireNamespace("ggplot2", quietly = TRUE) ){
  doc <- addTitle( doc, "Plot example", level = 1 )

  # create a ggplot2 plot
  myplot = ggplot2::qplot(Sepal.Length, Petal.Length, data = iris
    , color = Species, size = Petal.Width, alpha = I(0.7) )

  # Add myplot into object doc
  doc <- addPlot( doc = doc, fun = print, x = myplot )

  # Add a legend below the plot
  doc <- addParagraph( doc, value = "my first plot",
stylename = "rPlotLegend")
}

################ FLEXTABLE DEMO ################

doc <- addTitle( doc, "FlexTable example", level = 1 )

# Create a FlexTable with data.frame mtcars, display rownames
# use different formatting properties for header and body
MyFTable <- FlexTable( data = mtcars, add.rownames = TRUE,
header.cell.props = cellProperties( background.color = "#00557F" ),
header.text.props = textProperties( color = "white",
font.size = 11, font.weight = "bold" ),
body.text.props = textProperties( font.size = 10 )
)
# zebra stripes - alternate colored backgrounds on table rows
MyFTable <- setZebraStyle( MyFTable, odd = "#E1EEf4", even = "white" )

# applies a border grid on table
MyFTable <- setFlexTableBorders(MyFTable,
inner.vertical = borderProperties( color="#0070A8", style="solid" ),
inner.horizontal = borderNone(),
outer.vertical = borderProperties( color = "#006699",
style = "solid", width = 2 ),
outer.horizontal = borderProperties( color = "#006699",
style = "solid", width = 2 )
)


# add MyFTable into document
doc <- addFlexTable( doc, MyFTable )
doc <- addParagraph( doc, value = "my first table",
stylename = "rTableLegend")
```

```
# write the doc
writeDoc( doc, file = "document_example.docx")


# get docx page dimensions ------
doc = docx( title = "title" )
dim( doc )
```

FlexCell *Cell object for FlexTable*

Description

Create a representation of a cell that can be inserted in a FlexRow. For internal usage.

Usage

```
FlexCell(value, colspan = 1, par.properties = parProperties(),
  cell.properties = cellProperties())
```

Arguments

value	a content value - a value of type character or pot or set_of_paragraphs.
colspan	defines the number of columns the cell should span
par.properties	parProperties to apply to content
cell.properties	cellProperties to apply to content

See Also

addFlexTable, addHeaderRow, addFooterRow

Examples

```
FlexCell( value = "Hello" )
FlexCell( value = "Hello", colspan = 3)
FlexCell( "Column 1",
cell.properties = cellProperties(background.color="#527578")  )

# define a complex formatted text
mytext <- pot("Hello", format = textProperties(color = "blue") ) +
  " " +
  pot( "world", format = textProperties(font.size = 9) )
Fcell = FlexCell( mytext, colspan = 4 )
```

```
# define two paragraph and put them in a FlexCell
mytext1 <- pot("Hello", format = textProperties(color = "blue") )
mytext2 <- pot( "world", format = textProperties(font.size = 9) )
Fcell <- FlexCell( set_of_paragraphs( mytext1, mytext2 ) )
```

FlexRow *Row object for FlexTable*

Description

Create a representation of a row that can be inserted in a FlexTable. For internal usage.

Usage

```
FlexRow(values, colspan, text.properties = textProperties(),
  par.properties = parProperties(), cell.properties = cellProperties())
```

Arguments

values	Optional. a character vector to use as text content, the row will contain as many cells as there are in values.
colspan	integer Optional. vector specifying for each element the number of columns to span for each corresponding value (in values).
text.properties	Optional. textProperties to apply to each cell. Used only if values are not missing.
par.properties	Optional. parProperties to apply to each cell. Used only if values are not missing.
cell.properties	Optional. cellProperties to apply to each cell. Used only if values are not missing.

See Also

FlexTable, alterFlexRow , addHeaderRow, addFooterRow

Examples

```
cell_fp <- cellProperties(background.color="#527578")
# example with characters ----
headerRow <- FlexRow( c("col 1", "col 2"), cell.properties = cell_fp )
# example with FlexCell ----
cell_fp <- cellProperties(background.color="#527578")
headerRow <- FlexRow()
headerRow[1] <- FlexCell( "col 1", cell.properties = cell_fp )
headerRow[2] <- FlexCell( "col 2", cell.properties = cell_fp )
```

FlexTable *FlexTable creation*

Description

Create an object of class FlexTable.

FlexTable can be manipulated so that almost any formatting can be specified.

An API is available to let you manipulate (format, add text, merge cells, etc.) your FlexTable. A FlexTable is made of 3 parts: header, body and footer. To insert headers and footers rows with eventually merged cells, see addHeaderRow and addFooterRow.

Formating can be done on cells, paragraphs and text (borders, colors, fonts, etc.) , see alterFlexTable.

Usage

```
FlexTable(data, numrow, numcol, header.columns = TRUE, add.rownames = FALSE,
  body.cell.props = cellProperties(), body.par.props = parProperties(padding
  = 0), body.text.props = textProperties(),
  header.cell.props = cellProperties(),
  header.par.props = parProperties(padding = 0),
  header.text.props = textProperties(font.weight = "bold"))
```

Arguments

data	(a data.frame or matrix object) to add
numrow	number of row in the table body. Mandatory if data is missing.
numcol	number of col in the table body. Mandatory if data is missing.
header.columns	logical value - should the colnames be included in the table as table headers. If FALSE, no headers will be printed unless you use addHeaderRow.
add.rownames	logical value - should the row.names be included in the table.
body.cell.props	default cells formatting properties for table body
body.par.props	default paragraphs formatting properties for table body
body.text.props	default text formatting properties for table body
header.cell.props	default cells formatting properties for table headers
header.par.props	default paragraphs formatting properties for table headers
header.text.props	default text formatting properties for table headers

Details

The classical workflow would be to create a FlexTable, to add headers rows (see addHeaderRow) and eventually footers rows (see addFooterRow).

A FlexTable lets you add text in cells and modify cells, paragraphs and text properties. Text can be added with operator [<-. Text, paragraphs and cells properties can be also modified with operator [<-. (see alterFlexTable).

Below list of functions to use with FlexTable objects:

Text formatting

Apply a textProperties object to a subset of the FlexTable. Use the operator [<-. The textProperties object will be used to format all text from selected cells. See alterFlexTable.

Text adding

Add text with operator [<-. Text can be added just after the last text in the cell or as a new paragraph. Format can also be specified. Text can also be a pot object if the text format is complex.

Paragraph formatting

Apply a parProperties object to a subset of the FlexTable. Use the operator [<-. The parProperties object will be used to format all paragraphs from selected cells. See alterFlexTable.

Cell formatting

Apply a cellProperties object to a subset of the FlexTable. Use the operator [<-. The cellProperties object will be used to format selected cells. See alterFlexTable.

Borders

Apply borders scheme to a FlexTable with function setFlexTableBorders.

Set a border to a selection in a FlexTable with the operator [<- and an object of class borderProperties. Don't forget to specify argument side. See alterFlexTable.

Cell background colors

Applies background colors to cells. See setFlexTableBackgroundColors.

Alternate row colors (zebra striping) with function setZebraStyle.

Applies background colors to rows with function setRowsColors.

Applies background colors to columns with function setColumnsColors.

Cell merge

Span rows within columns with function spanFlexTableRows.

Span columns within rows with function spanFlexTableColumns.

Columns widths

Set columns widths with function setFlexTableWidths.

See Also

addHeaderRow, addFooterRow, setFlexTableWidths , alterFlexTable, setFlexTableBorders , spanFlexTableRows, spanFlexTableColumns , setRowsColors, setColumnsColors, setZebraStyle , setFlexTableBackgroundColors, pot , addFlexTable

Examples

```
# Create a FlexTable with data.frame mtcars, display rownames
# use different formatting properties for header and body
MyFTable <- FlexTable( data = mtcars, add.rownames = TRUE,
  header.cell.props = cellProperties( background.color = "#00557F" ),
  header.text.props = textProperties( color = "white",
    font.size = 11, font.weight = "bold" ),
  body.text.props = textProperties( font.size = 10 )
)
# zebra stripes - alternate colored backgrounds on table rows
MyFTable <- setZebraStyle( MyFTable, odd = "#E1EEf4", even = "white" )

# applies a border grid on table
MyFTable <- setFlexTableBorders(MyFTable,
  inner.vertical = borderProperties( color="#0070A8", style="solid" ),
  inner.horizontal = borderNone(),
  outer.vertical = borderProperties( color = "#006699",
style = "solid", width = 2 ),
  outer.horizontal = borderProperties( color = "#006699",
style = "solid", width = 2 )
)

# set default font size to 10
options("ReporteRs-fontsize" = 10)

# a summary of mtcars
dataset <- aggregate(mtcars[, c("disp", "mpg", "wt")],
  by = mtcars[, c("cyl", "gear", "carb")], FUN = mean)
dataset <- dataset[order(dataset$cyl, dataset$gear, dataset$carb),]


# set cell padding defaut to 2
baseCellProp <- cellProperties(padding = 2)

# Create a FlexTable with data.frame dataset
MyFTable <- FlexTable( data = dataset,
  body.cell.props = baseCellProp, header.cell.props = baseCellProp,
  header.par.props = parProperties(text.align = "right") )

# set columns widths (inch)
MyFTable <- setFlexTableWidths(MyFTable,
                               widths = c(0.5, 0.5, 0.5, 0.7, 0.7, 0.7))

# span successive identical cells within column 1, 2 and 3
MyFTable <- spanFlexTableRows(MyFTable, j = 1,
                              runs = as.character(dataset$cyl))
MyFTable <- spanFlexTableRows(MyFTable, j = 2,
                              runs = as.character(dataset$gear))
MyFTable <- spanFlexTableRows(MyFTable, j = 3,
                              runs = as.character(dataset$carb))
```

```
# overwrites some text formatting properties
MyFTable[dataset$wt < 3, 6] <- textProperties(color = "#003366")
MyFTable[dataset$mpg < 20, 5] <- textProperties(color = "#993300")

# overwrites some paragraph formatting properties
MyFTable[, 1:3] <- parProperties(text.align = "center")
MyFTable[, 4:6] <- parProperties(text.align = "right")


Footnote1 <- Footnote()

par1 <- pot("About this reference", textBold())
par2 <- pot(
  "Omni ab coalitos pro malivolus obsecrans graviter cum perquisitor \
  perquisitor pericula saepeque inmunibus coalitos ut.",
  textItalic(font.size = 8)
)
Footnote1 <- addParagraph(Footnote1,
                          set_of_paragraphs(par1, par2),
                          parProperties(text.align = "justify"))

Footnote1 <- addParagraph(
  Footnote1,
  set_of_paragraphs("list item 1", "list item 2"),
  parProperties(text.align = "left", list.style = "ordered")
)

an_rscript <- RScript(text = "x = rnorm(10)")
Footnote1 <- addParagraph(Footnote1, an_rscript)

MyFTable[1, 1, newpar = TRUE] <- pot("a note",
                                     footnote = Footnote1,
                                     format = textBold(color = "gray"))

pot_link <- pot(" (link example)", textProperties(color = "cyan"),
                hyperlink = "http://www.wikipedia.org/")

MyFTable[1, 1, to = "header"] <- pot_link

# applies a border grid on table
MyFTable <- setFlexTableBorders( MyFTable, footer = TRUE,
  inner.vertical = borderProperties(color = "#666666"),
  inner.horizontal = borderProperties(color = "#666666"),
  outer.vertical = borderProperties(width = 2, color = "#666666"),
  outer.horizontal = borderProperties(width = 2, color = "#666666") )

ft <- vanilla.table(head(iris))
ft <- setFlexTableBackgroundColors( ft,
  i = 1:3, j = c("Petal.Length", "Species"), colors = "yellow" )
```

Footnote *Create a Footnote*

Description

A footnote is a a set of paragraphs placed at the bottom of a page if document object is a docx object.

If in a docx object, footnote will be flagged by a number immediately following the portion of the text the note is in reference to.

Usage

```
Footnote(index.text.properties = textProperties(vertical.align =
  "superscript"))
```

Arguments

index.text.properties
: textProperties to apply to note index symbol (only for docx object).

Value

an object of class Footnote.

See Also

docx, pot

Examples

```
## docx example
doc = docx( )

par1 = pot("About this reference", textBold( ) )
par2 = pot("Omni ab coalitos pro malivolus obsecrans graviter
cum perquisitor perquisitor pericula saepeque inmunibus coalitos ut.",
textItalic(font.size = 8) )

Footnote1 = Footnote( )
Footnote1 = addParagraph( Footnote1, set_of_paragraphs( par1, par2 ),
parProperties(text.align = "justify"))
Footnote1 = addParagraph( Footnote1,
set_of_paragraphs( "list item 1", "list item 2" ),
parProperties(text.align = "left", list.style = "ordered"))
an_rscript = RScript( par.properties = parProperties(shading.color = "gray90"),
text = "ls()
x = rnorm(10)" )
```

```
Footnote1 = addParagraph( Footnote1, an_rscript,
parProperties(text.align = "left"))


Footnote2 = Footnote(  )
Footnote2 = addParagraph( Footnote2, pot("This is another reference" ),
par.properties = parProperties(text.align = "center"))

doc = addTitle( doc, "Title example 1", level = 1 )

pot1 = "Hae duae provinciae " + pot("bello",
footnote = Footnote1 ) + " quondam piratico catervis mixtae
praedonum a Servilio pro consule missae sub
iugum factae sunt vectigales. et hae quidem regiones velut in prominenti
lingua positae ob orbe eoo monte Amano disparantur."

pot2 = pot("Latius iam disseminata licentia onerosus bonis Caesar
post haec adhibens modum orientis latera cuncta vexabat nec honoratis
nec urbium primatibus nec plebeiis." ) + pot(" Here is another note.",
  footnote = Footnote2)


# Add my.pars into the document doc
doc = addParagraph(doc, set_of_paragraphs( pot1, pot2 ) )

docx.file = "footnote.docx"

writeDoc( doc, file = docx.file )
```

`is.color` *color checking*

Description

Check if character string is a valid color representation

Usage

```
is.color(x)
```

Arguments

x	value(s) to be tested

Details

see http://stackoverflow.com/questions/13289009/check-if-character-string-is-a-valid-color-represent
13290832#13290832

See Also

pptx, docx

Examples

```
is.color( c(NA, "black", "blackk", "1", "#00", "#000000") )
```

`knit_print.FlexTable` *FlexTable custom printing function for knitr*

Description

FlexTable custom printing function for knitr

Usage

```
## S3 method for class 'FlexTable'
knit_print(x, ...)
```

Arguments

`x`	a `FlexTable` to be printed
`...`	further arguments, not used.

`light.table` *light FlexTable shortcut*

Description

get a simple FlexTable from a dataset

Usage

```
light.table(dataset, add.rownames = FALSE, text.direction = "lrtb")
```

Arguments

`dataset`	the data to use
`add.rownames`	logical value - should the row.names be included in the table.
`text.direction`	header cell text rotation - a single character value, expected value is one of "lrtb", "tbrl", "btlr".

See Also

FlexTable

Examples

```
ft <- light.table( iris)
```

`list.settings` *format ordered and unordered lists*

Description

Create a description used to format ordered and unordered lists in object documents.

Usage

```
list.settings(ol.left = seq(from = 0, by = 0.4, length.out = 9),
  ol.hanging = rep(0.4, 9), ol.format = rep("decimal", 9),
  ol.pattern = paste0("%", 1:9, "."), ul.left = seq(from = 0, by = 0.4,
  length.out = 9), ul.hanging = rep(0.4, 9), ul.format = c("disc", "circle",
  "square", "disc", "circle", "square", "disc", "circle", "square"))
```

Arguments

`ol.left`	left indent values (in inches) for each level of an ordered list. Length must be 9 as there are 9 elements to define (from level1 to level9).
`ol.hanging`	space values (in inches) between numbering label (argument `ol.format`) and content for each level of an ordered list. Length must be 9 as there are 9 elements to define (from level1 to level9).
`ol.format`	type of numbering for ordered levels, values can be 'decimal' or 'upperRoman' or 'lowerRoman' or 'upperLetter' or 'lowerLetter'. Length must be 9 as there are 9 elements to define (from level1 to level9).
`ol.pattern`	numbering pattern for ordered levels. A level numbering has the following syntax: `"%1"` (numbering of level1), `"%2"` (numbering of level2), ..., `"%9"` (numbering of level9).
`ul.left`	left indent values for each level of an unordered list. Length must be 9 as there are 9 elements to define (from level1 to level9). Length must be 9 as there are 9 elements to define (from level1 to level9).
`ul.hanging`	space values (in inches) between bullet symbol (argument `ul.format`) and content for each level of an unordered list. Length must be 9 as there are 9 elements to define (from level1 to level9).
`ul.format`	type of bullet for unordered levels, values can be 'disc' or 'circle' or 'square'. Length must be 9 as there are 9 elements to define (from level1 to level9).

Details

List settings are used to configure formatting of list in documents.

It can be set in R session options or as a parameter in `docx` or `pptx`.

See Also

addParagraph, ReporteRs

Examples

```
numbering.pattern = c( "%1.", "%1. %2.", "%1. %2. %3.",
  "%4.", "%5.", "%6.", "%7.", "%8.", "%9." )

ordered.formats = rep( c( "decimal", "upperRoman", "upperLetter"), 3 )

unordered.formats = rep( c( "square", "disc", "circle"), 3 )

left.indent = seq( from = 0, by = 0.5, length.out = 9)

options("ReporteRs-list-definition" = list(
  ol.left = left.indent,
  ol.hanging = rep( 0.4, 9 ),
  ol.format = ordered.formats,
  ol.pattern = numbering.pattern,
  ul.left = left.indent,
  ul.hanging = rep( 0.4, 9 ),
  ul.format = unordered.formats
  )
)


doc.filename = "ex_list.docx"
doc <- docx()

doc <- addTitle( doc, "List example", level = 1 )

# define some text
text1 = "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
text2 = "In sit amet ipsum tellus. Vivamus arcu sit amet faucibus auctor."
text3 = "Quisque dictum tristique ligula."

# define parProperties with list properties
ordered.list.level1 = parProperties(list.style = "ordered",
                                    level = 1 )
ordered.list.level2 = parProperties(list.style = "ordered",
                                    level = 2 )

# define parProperties with list properties
unordered.list.level1 = parProperties(list.style = "unordered",
                                      level = 1 )
unordered.list.level2 = parProperties(list.style = "unordered",
                                      level = 2 )

# add ordered list items
doc = addParagraph( doc, value = text1,
                    par.properties = ordered.list.level1 )
doc = addParagraph( doc, value = text2,
```

```
                    par.properties = ordered.list.level2 )

# add ordered list items without restart renumbering
doc = addParagraph( doc, value = c( text1, text2, text3),
                    par.properties = ordered.list.level1 )

# add ordered list items and restart renumbering
doc = addParagraph( doc, value = c( text1, text2, text3),
                    restart.numbering = TRUE, par.properties = ordered.list.level1 )

# add unordered list items
doc = addParagraph( doc, value = text1,
                    par.properties = unordered.list.level1 )
doc = addParagraph( doc, value = text2,
                    par.properties = unordered.list.level2 )

writeDoc( doc, file = doc.filename )
```

`list_bookmarks` *List Bookmarks from a Word Document*

Description

List all bookmarks available in a docx object.

Usage

```
list_bookmarks(x, body = TRUE, header = TRUE, footer = TRUE)
```

Arguments

x	a docx object
body	specifies to scan document body
header	specifies to scan document header
footer	specifies to scan document footer

Value

a character vector

See Also

docx, text_extract

Examples

```
doc <- docx( title = "My example", template = file.path(
  system.file(package = "ReporteRs"), "templates/bookmark_example.docx") )
list_bookmarks( doc )
```

`map_title` *map titles styles*

Description

Set manually headers'styles of a docx object

Usage

```
map_title(doc, stylenames)
```

Arguments

doc	docx object to be used with map_title.
stylenames	existing styles (character vector) where first element represents the style to use for title 1, second element represents the style to use for title 2, etc.

Details

Function `addTitle` need to know which styles are corresponding to which title level (1 ; 1.1 ; 1.1.1 ; etc.). When template is read, function `docx` try to guess what are theses styles. If he do not succeed, an error occured saying 'You must defined header styles via map_title first.'. In that case, run `styles(...)` to see what are available styles, then map_title to indicate which available styles are meant to be used as header styles.

See Also

`docx`,`styles.docx`,`addTitle.docx`

Examples

```
doc.filename = "addImage_example.docx"
doc <- docx()
doc = map_title(doc, stylenames = c("Titre1", "Titre2", "Titre3",
  "Titre4", "Titre5", "Titre6", "Titre7", "Titre8", "Titre9" ) )
```

office_web_viewer *Office Web Viewer*

Description

Produce an iframe linked to Office Web Viewer. It let's you display a Microsoft Office document in a Web browser.

Usage

```
office_web_viewer(url)
```

Arguments

url	file url

parProperties *Paragraph formatting properties*

Description

Create a parProperties object that describes paragraph formatting properties.

Usage

```
parProperties(text.align = "left", padding.bottom = 1, padding.top = 1,
  padding.left = 1, padding.right = 1, padding, list.style = "none",
  level = 1, border.bottom = borderNone(), border.left = borderNone(),
  border.top = borderNone(), border.right = borderNone(), shading.color)

## S3 method for class 'parProperties'
chprop(object, text.align, padding.bottom, padding.top,
  padding.left, padding.right, padding, list.style, level, border.bottom,
  border.left, border.top, border.right, shading.color, ...)

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

Arguments

text.align	text alignment - a single character value, expected value is one of 'left', 'right', 'center', 'justify'.
padding.bottom	paragraph bottom padding - 0 or positive integer value.
padding.top	paragraph top padding - 0 or positive integer value.
padding.left	paragraph left padding - 0 or positive integer value.

padding.right	paragraph right padding - 0 or positive integer value.
padding	paragraph padding - 0 or positive integer value. Argument padding overwrites arguments padding.bottom, padding.top, padding.left, padding.right.
list.style	list style - a single character value, expected value is one of 'none' (default), 'unordered', 'ordered', 'blockquote'. This will not have any effect if used in a FlexTable.
level	list level if argument list is not 'none'. This will not have any effect if used in a FlexTable.
border.bottom	borderProperties for bottom border. overwrite all border.bottom.* if specified.
border.left	borderProperties for left border. overwrite all border.left.* if specified.
border.top	borderProperties for top border. overwrite all border.top.* if specified.
border.right	borderProperties for right border. overwrite all border.right.* if specified.
shading.color	shading color - a single character value specifying a valid color (e.g. "#000000" or "black").
object	parProperties object to update
...	further arguments - not used
x	parProperties object to print

Details

parProperties is used to control paragraph properties. It is used when adding plots or when adding content in a FlexTable.

Default values are:

- text.align "left"
- padding.bottom 1
- padding.top 1
- padding.left 1
- padding.right 1
- list.style 'none'
- level 1

Get a modified version of a parProperties with chprop.

Value

a parProperties object

See Also

alterFlexTable, addParagraph, shortcut_properties

Examples

```
parProperties( text.align = "center", padding = 5)

parProperties( text.align = "center",
  padding.top = 5, padding.bottom = 0,
  padding.left = 2, padding.right = 0 )

parProperties( list.style = "ordered", level = 2)

parProperties( list.style = "unordered", level = 2)
parProp = parProperties()

chprop( parProp, text.align = "center" )
chprop( parProp, padding.left = 2 )
chprop( parProp, padding = 2 )

chprop( parProp, padding = 2, text.align = "center" )
```

---

pot *Piece of Text (formated text)*

Description

Create an object with a text to display and its formatting properties.

Usage

```
pot(value = "", format = textProperties(), hyperlink, footnote)

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

## S3 method for class 'pot'
as.character(x, ...)

## S3 method for class 'pot'
knit_print(x, ...)
```

Arguments

value	text value or a value that has a format method returning character value.
format	formatting properties (an object of class textProperties).
hyperlink	a valid url to use as hyperlink when clicking on value.
footnote	a Footnote object.
x	a pot object
...	further arguments, not used.

Details

a pot (piece of text) is a convenient way to define a paragraph of text where some text are not all formated the same.

A pot can be associated with an hyperlink.

A pot can be associated with a Footnote. Note that footnotes can not be inserted in a `pptx` object.

See Also

`addParagraph`, `Footnote` , `+.pot`, `pot_img`,

Examples

```
# "My tailor is rich" with formatting on some words
pot1 <- pot("My tailor", textProperties(color = "red" ) ) +
  " is " + pot("rich", textProperties(shading.color = "red",
     font.weight = "bold" ) )

# "Cats and dogs" with formatting on some words
pot2 = pot("Cats", textProperties(color = "red" ) ) +
  " and " +
  pot("dogs", textProperties( color = "blue" ),
      hyperlink = "http://www.wikipedia.org/" )
```

---

pot_img *Image to be concatenate with pot object*

Description

Create an pot object that handle images.

Usage

```
pot_img(filename, width, height)
```

Arguments

`filename`	`"character"` value, complete filename of the external image
`width`	image width in inches
`height`	image height in inches

pptx *Create Microsoft PowerPoint document object representation*

Description

Create a pptx object

Usage

```
pptx(title, template,
  list.definition = getOption("ReporteRs-list-definition"))

## S3 method for class 'pptx'
dim(x)

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

Arguments

title	"character" value: title of the document (in the doc properties).
template	"character" value, it represents the filename of the pptx file used as a template.
list.definition	a list definition to specify how ordered and unordered lists have to be formated. See list.settings. Default to getOption("ReporteRs-list-definition").
x	Object of class pptx
...	further arguments, not used.

Details

To send R output in a pptx document, a slide (see addSlide.pptx have to be added to the object first (because output is beeing written in slides).

Several methods can used to send R output into an object of class pptx.

- addTitle.pptx add titles
- addParagraph.pptx add text
- addPlot.pptx add plots
- addFlexTable.pptx add FlexTable
- addDate.pptx add a date (most often in the bottom left area of the slide)
- addFooter.pptx add a comment in the footer (most often in the bottom center area of the slide)
- addPageNumber.pptx add a page number (most often in the bottom right area of the slide)
- addImage.pptx add external images

Once object has content, user can write the pptx into a ".pptx" file, see writeDoc.

dim returns slide width and height, position and dimension of the next available shape in the current slide.

print print informations about an object of class pptx.

Value

an object of class pptx.

Note

Power Point 2007-2013 (*.pptx) file formats are the only supported files.

Document are manipulated in-memory ; a pptx's document is not written to the disk unless the writeDoc method has been called on the object.

References

Wikipedia: Office Open XML
http://en.wikipedia.org/wiki/Office_Open_XML

See Also

docx

Examples

```
# set default font size to 10
options( "ReporteRs-fontsize" = 11 )

# Word document to write
pptx.file = "presentation_example.pptx"

# Create a new document
doc = pptx( title = "title" )

# display layouts names
slide.layouts( doc )

# add a slide with layout "Title Slide"
doc = addSlide( doc, slide.layout = "Title Slide" )

#set the main title
doc = addTitle( doc, "Presentation title" )
#set the sub-title
doc = addSubtitle( doc , "This document is generated with ReporteRs.")


################ TEXT DEMO ################

# add a slide with layout "Title and Content" then add content
```

```
doc = addSlide( doc, slide.layout = "Two Content" )

# add a title
doc = addTitle( doc, "Text demo" )
sometext = c( "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
, "In sit amet ipsum tellus. Vivamus dignissim sit amet auctor."
, "Quisque dictum tristique ligula."
)

# add simple text
doc = addParagraph( doc, value = sometext )

# Add "My tailor is rich" and "Cats and Dogs"
# format some of the pieces of text
pot1 = pot("My tailor"
, textProperties(color="red" ) ) + " is " + pot("rich"
, textProperties(font.weight="bold") )
pot2 = pot("Cats"
, textProperties(color="red" )
) + " and " + pot("Dogs"
, textProperties(color="blue" ) )
doc = addParagraph(doc, set_of_paragraphs( pot1, pot2 ) )

################ PLOT DEMO ################
if( requireNamespace("ggplot2", quietly = TRUE) ){
  doc = addSlide( doc, slide.layout = "Title and Content" )
  doc = addTitle( doc, "Plot examples" )

  myplot = ggplot2::qplot(Sepal.Length, Petal.Length
    , data = iris, color = Species
    , size = Petal.Width, alpha = I(0.7)
)
  # Add titles and then 'myplot'
  doc = addPlot( doc, function( ) print( myplot ) )
}

################ FLEXTABLE DEMO ################
doc = addSlide( doc, slide.layout = "Title and Content" )
doc = addTitle( doc, "FlexTable example" )

# Create a FlexTable with data.frame mtcars, display rownames
# use different formatting properties for header and body
MyFTable = FlexTable( data = mtcars, add.rownames = TRUE,
header.cell.props = cellProperties( background.color = "#00557F" ),
header.text.props = textProperties( color = "white",
font.size = 11, font.weight = "bold" ),
body.text.props = textProperties( font.size = 10 )
)
# zebra stripes - alternate colored backgrounds on table rows
MyFTable = setZebraStyle( MyFTable, odd = "#E1EEf4", even = "white" )

# applies a border grid on table
MyFTable = setFlexTableBorders(MyFTable,
```

```
inner.vertical = borderProperties( color="#0070A8", style="solid" ),
inner.horizontal = borderNone(),
outer.vertical = borderProperties( color = "#006699",
  style = "solid", width = 2 ),
outer.horizontal = borderProperties( color = "#006699",
  style = "solid", width = 2 )
)

# add MyFTable into document
doc = addFlexTable( doc, MyFTable )

# write the doc
writeDoc( doc, file = pptx.file )


# get pptx page dimensions ------
doc = pptx( title = "title" )
doc = addSlide( doc, "Title and Content" )
dim(doc)
```

`print.FlexTable` *Print FlexTables*

Description

print a `FlexTable` object. If R session is interactive, the FlexTable is rendered in an HTML page and loaded into a WWW browser.

Usage

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

Arguments

`x`	a `FlexTable` object
`...`	further arguments, not used.

`print.Footnote` *print a Footnote*

Description

print a Footnote

Usage

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

Arguments

x	a Footnote object
...	further arguments, not used.

---

renderFlexTable *FlexTable output for shiny*

Description

Creates a reactive FlexTable that is suitable for assigning to a tableOutput slot.

Usage

```
renderFlexTable(expr, ..., env = parent.frame(), quoted = FALSE)
```

Arguments

expr	An expression that returns a FlexTable object.
...	not used
env	The environment in which to evaluate expr.
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable.

RScript *RScript object*

Description

Colored RScript object

Usage

```
RScript(file, text, comment.properties = textProperties(color = "#A7947D"),
  roxygencomment.properties = textProperties(color = "#5FB0B8"),
  symbol.properties = textProperties(color = "black"),
  operators.properties = textProperties(color = "black"),
  keyword.properties = textProperties(color = "#4A444D"),
  string.properties = textProperties(color = "#008B8B", font.style =
  "italic"), number.properties = textProperties(color = "blue"),
  functioncall.properties = textProperties(color = "blue"),
  argument.properties = textProperties(color = "#666666"),
  package.properties = textProperties(color = "green"),
  formalargs.properties = textProperties(color = "#424242"),
  eqformalargs.properties = textProperties(color = "#424242"),
  assignement.properties = textProperties(color = "black"),
  slot.properties = textProperties(color = "#F25774"),
  default.properties = textProperties(color = "black"),
  par.properties = parProperties())
```

Arguments

`file`	R script file. Not used if text is provided.
`text`	character vector. The text to parse. Not used if file is provided.
`comment.properties`	comment txtProperties object
`roxygencomment.properties`	roxygencomment txtProperties object
`symbol.properties`	symbol txtProperties object
`operators.properties`	operators txtProperties object
`keyword.properties`	keyword txtProperties object
`string.properties`	string txtProperties object
`number.properties`	number txtProperties object
`functioncall.properties`	functioncall txtProperties object
`argument.properties`	argument txtProperties object
`package.properties`	package txtProperties object
`formalargs.properties`	formalargs txtProperties object
`eqformalargs.properties`	eqformalargs txtProperties object

`assignement.properties`
: assignement txtProperties object

`slot.properties`
: slot txtProperties object

`default.properties`
: default txtProperties object

`par.properties`
: a parProperties object

See Also

addRScript

Examples

```
an_rscript = RScript( text = "ls()
x = rnorm(10)" )
```

`setColumnsColors` *applies background colors to columns of a FlexTable*

Description

applies background colors to columns of a FlexTable

Usage

```
setColumnsColors(object, j, colors)
```

Arguments

`object`
: a `FlexTable` object

`j`
: vector (integer index, col.names values or boolean vector) for columns selection.

`colors`
: background colors to apply (e.g. "#000000" or "black")

See Also

setRowsColors, FlexTable, setZebraStyle

Examples

```
MyFTable <- FlexTable( data = mtcars[1:10, ], add.rownames=TRUE )
MyFTable <- setColumnsColors( MyFTable, j=3:4, colors = "red" )
```

setFlexTableBackgroundColors *applies background colors to cells of a FlexTable*

Description

applies background colors to cells of a FlexTable

Usage

```
setFlexTableBackgroundColors(object, i, j, colors, to = "body")
```

Arguments

object	a FlexTable object
i	vector (integer index, row.names values or boolean vector) for rows selection.
j	vector (integer index, col.names values or boolean vector) for columns selection.
colors	background colors to apply (e.g. "#000000" or "black"). a character vector of colors with as many elements as defined by the selection.
to	specify on which part of the FlexTable to apply colors, must be one of the following values "body" (default) or "header" or "footer"

See Also

FlexTable, is.color

Examples

```
# example 1 ----
ft <- vanilla.table(head(iris))
ft <- setFlexTableBackgroundColors( ft,
  i = 1:3, j = c("Petal.Length", "Species"), colors = "yellow" )

# example 2 ----
data <- structure(c(1, -0.991, -0.993, -0.956, 0.939, -0.986, 0.708,
0.932, 0.827, 0.768, -0.799, -0.991, 1, 0.992, 0.972, -0.923,
0.966, -0.766, -0.963, -0.783, -0.735, 0.82, -0.993, 0.992, 1,
0.944, -0.955, 0.987, -0.692, -0.93, -0.843, -0.801, 0.757, -0.956,
0.972, 0.944, 1, -0.819, 0.897, -0.88, -0.985, -0.635, -0.565,
0.924, 0.939, -0.923, -0.955, -0.819, 1, -0.975, 0.472, 0.795,
0.944, 0.92, -0.565, -0.986, 0.966, 0.987, 0.897, -0.975, 1,
-0.59, -0.872, -0.903, -0.855, 0.697, 0.708, -0.766, -0.692,
-0.88, 0.472, -0.59, 1, 0.899, 0.204, 0.14, -0.949, 0.932, -0.963,
-0.93, -0.985, 0.795, -0.872, 0.899, 1, 0.594, 0.544, -0.912,
0.827, -0.783, -0.843, -0.635, 0.944, -0.903, 0.204, 0.594, 1,
0.979, -0.34, 0.768, -0.735, -0.801, -0.565, 0.92, -0.855, 0.14,
0.544, 0.979, 1, -0.236, -0.799, 0.82, 0.757, 0.924, -0.565,
```

```
0.697, -0.949, -0.912, -0.34, -0.236, 1), .Dim = c(11L, 11L),
.Dimnames = list(c("mpg", "cyl", "disp", "hp", "drat", "wt",
"qsec", "vs", "am", "gear", "carb"), c("mpg", "cyl", "disp",
"hp", "drat", "wt", "qsec", "vs", "am", "gear", "carb")))

mycolors <-
  structure(c("#1A9850", "#D73027", "#D73027", "#D73027", "#1A9850",
  "#D73027", "#66BD63", "#1A9850", "#1A9850", "#1A9850", "#D73027",
  "#D73027", "#1A9850", "#1A9850", "#1A9850", "#D73027", "#1A9850",
  "#D73027", "#D73027", "#D73027", "#F46D43", "#1A9850", "#D73027",
  "#1A9850", "#1A9850", "#1A9850", "#D73027", "#1A9850", "#F46D43",
  "#D73027", "#D73027", "#D73027", "#1A9850", "#D73027", "#1A9850",
  "#1A9850", "#1A9850", "#D73027", "#1A9850", "#D73027", "#D73027",
  "#F46D43", "#F46D43", "#1A9850", "#1A9850", "#D73027", "#D73027",
  "#D73027", "#1A9850", "#D73027", "#A6D96A", "#1A9850", "#1A9850",
  "#1A9850", "#F46D43", "#D73027", "#1A9850", "#1A9850", "#1A9850",
  "#D73027", "#1A9850", "#F46D43", "#D73027", "#D73027", "#D73027",
  "#66BD63", "#66BD63", "#D73027", "#F46D43", "#D73027", "#A6D96A",
  "#F46D43", "#1A9850", "#1A9850", "#D9EF8B", "#D9EF8B", "#D73027",
  "#1A9850", "#D73027", "#D73027", "#D73027", "#1A9850", "#D73027",
  "#1A9850", "#1A9850", "#66BD63", "#66BD63", "#D73027", "#1A9850",
  "#D73027", "#D73027", "#F46D43", "#1A9850", "#D73027", "#D9EF8B",
  "#66BD63", "#1A9850", "#1A9850", "#FDAE61", "#1A9850", "#F46D43",
  "#D73027", "#F46D43", "#1A9850", "#D73027", "#D9EF8B", "#66BD63",
  "#1A9850", "#1A9850", "#FEE08B", "#D73027", "#1A9850", "#1A9850",
  "#1A9850", "#F46D43", "#66BD63", "#D73027", "#D73027", "#FDAE61",
  "#FEE08B", "#1A9850"), .Dim = c(11L, 11L))

MyFTable <- FlexTable( data, add.rownames = TRUE )

# set computed colors
MyFTable <- setFlexTableBackgroundColors( MyFTable,
  j = dimnames(data)[[2]], colors = mycolors )

# cosmetics
MyFTable <- setFlexTableBackgroundColors( MyFTable,
  i = 1, colors = "gray", to = "header" )
MyFTable[1, , to = "header"] <- textBold(color="white")

MyFTable <- setFlexTableBackgroundColors( MyFTable,
  j = 1, colors = "gray" )
MyFTable[,1] <- textBold(color="white")

bp1 <- borderProperties( color = "white"  )
bp2 <- chprop( bp1, width = 2 )
MyFTable <- setFlexTableBorders( MyFTable,
  inner.vertical = bp1, inner.horizontal = bp1,
  outer.vertical = bp2, outer.horizontal = bp2 )
```

---

setFlexTableBorders *change grid lines of a FlexTable*

Description

apply borders scheme to a FlexTable. A border scheme is a set of 4 different borders: inner vectical and horizontal , outer vectical and horizontal.

Usage

```
setFlexTableBorders(object, inner.vertical = borderProperties(),
  inner.horizontal = borderProperties(),
  outer.vertical = borderProperties(),
  outer.horizontal = borderProperties(), body = TRUE, header = TRUE,
  footer = FALSE)
```

Arguments

object	a FlexTable object
inner.vertical	a borderProperties object
inner.horizontal	a borderProperties object
outer.vertical	a borderProperties object
outer.horizontal	a borderProperties object
body	a logical value (default to TRUE), specifies to apply scheme to table body
header	a logical value (default to TRUE), specifies to apply scheme to table header
footer	a logical value (default to FALSE), specifies to apply scheme to table footer

See Also

FlexTable

Examples

```
MyFTable <- FlexTable( data = mtcars[1:10, ], add.rownames=TRUE )
MyFTable <- setFlexTableBorders( MyFTable,
  inner.vertical = borderProperties( style = "dashed" ),
  inner.horizontal = borderProperties( style = "dashed" ),
  outer.vertical = borderProperties( width = 2 ),
  outer.horizontal = borderProperties( width = 2 ) )
```

setFlexTableWidths *set columns widths of a FlexTable*

Description

set columns widths of a FlexTable in inches.

Usage

```
setFlexTableWidths(object, widths)
```

Arguments

object	a FlexTable object
widths	a numeric vector specifying columns widths in inches.

See Also

FlexTable

Examples

```
MyFTable <- FlexTable( data = iris[1:10, ] )
MyFTable <- setFlexTableWidths( MyFTable, widths = c(1,1,1,1,3))
```

setRowsColors *applies background colors to rows of a FlexTable*

Description

applies background colors to rows of a FlexTable

Usage

```
setRowsColors(object, i, colors)
```

Arguments

object	a FlexTable object
i	vector (integer index, row.names values or boolean vector) for rows selection.
colors	background colors to apply (e.g. "#000000" or "black")

See Also

FlexTable, setColumnsColors, setZebraStyle

Examples

```
MyFTable <- FlexTable( data = mtcars[1:10, ], add.rownames=TRUE )
MyFTable <- setRowsColors( MyFTable, i=1:4, colors = "red" )
```

---

setZebraStyle *FlexTable rows zebra striping*

Description

applies background color to alternate rows (zebra striping). Set a color if row index is odd and another if row index is even.

Usage

```
setZebraStyle(object, odd, even)
```

Arguments

object	a FlexTable object
odd	background color applied to odd row indexes - single character value (e.g. "#000000" or "black")
even	background color applied to even row indexes - single character value (e.g. "#000000" or "black")

See Also

FlexTable

Examples

```
MyFTable <- FlexTable( data = iris[1:10, ] )

# Zebra striped table
MyFTable <- setZebraStyle( MyFTable, odd = "#8A949B", even = "#FAFAFA" )
```

set_of_paragraphs *Set of paragraphs of text*

Description

Create a container of paragraphs of text (pot objects).

Usage

```
set_of_paragraphs(...)
```

Arguments

...	pot objects, one per paragraph.

Details

each pot are representing a paragraph. A paragraph consists of one or more pieces of text and ends with an end of line. Objects of class set_of_paragraphs are to be used with addParagraph.

See Also

addParagraph, addParagraph.docx, addParagraph.pptx, pot

Examples

```
pot1 = pot("My tailor", textProperties(color="red") ) + " is " + pot("rich"
, textProperties(font.weight="bold") )
pot2 = pot("Cats", textProperties(color="red") ) + " and " + pot("Dogs"
, textProperties(color="blue") )
my.pars = set_of_paragraphs( pot1, pot2 )
```

slide.layouts *Get layout names of a document object*

Description

Get layout names that exist into a document

Usage

```
slide.layouts(doc, ...)
```

Arguments

doc	document object
...	further arguments passed to other methods

Details

`slide.layouts` only works with pptx documents. See `slide.layouts.pptx` for examples.

See Also

`pptx`, `slide.layouts.pptx`, `addSlide.pptx`

---

slide.layouts.pptx *Get layout names of a pptx document*

Description

Get layout names that exist into the template used when pptx has been created.

Usage

```
## S3 method for class 'pptx'
slide.layouts(doc, layout, ...)
```

Arguments

`doc`	Object of class `pptx` to extract layout names from.
`layout`	optional single string value, one of the layout names
`...`	further arguments, not used.

Details

Available names are layout names of the template document (e.g. Title and Content , Two Content, etc.). If layout is specified, the layout representation will be produced in a plot. This can be useful to check available shapes.

See Also

`pptx`, `addSlide.pptx`, `slide.layouts`

Examples

```
doc.filename = "addFlexTable_example.pptx"
doc = pptx( title = "title" )
layouts = slide.layouts(doc)
layouts

# loop over layout names to plot each slide style
for(i in layouts ){
  slide.layouts(doc, i )
  title(sub = i )
}
```

`spanFlexTableColumns` *Span columns within rows*

Description

Span columns within rows.

Usage

```
spanFlexTableColumns(object, i, from, to, runs)
```

Arguments

`object`	a `FlexTable` object
`i`	vector (integer index, row.names values or boolean vector) for rows selection.
`from`	index of the first column to span (its content will be the visible one).
`to`	index of the last column to span.
`runs`	a vector of size `numcol` of FlexTable. If provided, successive runs of equal values will indicate to merge corresponding columns.

Note

Overlappings of horizontally merged cells and vertically merged cells are forbidden.

See Also

`spanFlexTableRows`, `FlexTable`

Examples

```
mydata = iris[46:55, ]
MyFTable <- FlexTable( data = mydata )

# merge columns 2 to 4 in line 3
MyFTable <- spanFlexTableColumns( MyFTable, i = 2, from = 2, to = 4 )

# merge cells in line 4 when successive values of
# a given character vector are identical. Note
# the character vector length is the same
# than the number of columns of the FlexTable.
MyFTable <- spanFlexTableColumns( MyFTable, i = 4,
  runs = c( "a", "b", "b", "c", "d") )
```

spanFlexTableRows *Span rows within columns*

Description

Span rows within columns.

Usage

```
spanFlexTableRows(object, j, from, to, runs)
```

Arguments

object	a FlexTable object
j	vector (integer index, col.names values or boolean vector) for columns selection.
from	index of the first row to span (its content will be the visible one).
to	index of the last row to span.
runs	a vector of size numrow of FlexTable. If provided, successive runs of equal values will indicate to merge corresponding rows.

Note

Overlappings of horizontally merged cells and vertically merged cells are forbidden.

See Also

FlexTable, spanFlexTableColumns

Examples

```
mydata = iris[46:55, ]
MyFTable <- FlexTable( data = mydata )

# merge line 5 to 7 in column 1
MyFTable <- spanFlexTableRows( MyFTable, j = 3, from = 5, to = 7 )

# merge cells in column "Species" when successive values
# of Species are identical. Note
# the character vector length is the same
# than the number of lines of the FlexTable.
MyFTable <- spanFlexTableRows( MyFTable, j = "Species",
  runs = as.character( mydata$Species ) )
```

styles *Get styles names of a document object*

Description

Get styles names that exist into a document

Usage

```
styles(doc, ...)

## S3 method for class 'docx'
styles(doc, ...)
```

Arguments

doc	document object
...	further arguments passed to other methods

Details

With docx document, styles will be paragraph styles of the base document (e.g. Normal, Title1, etc.). Names of the returned character vector are labels associated with styles names.

See Also

docx, styles

Examples

```
doc = docx( title = "title" )
styles(doc)
```

textNormal *shortcuts for formatting properties*

Description

Shortcuts for textProperties, parProperties, borderProperties and cellProperties.

Usage

```
textNormal(...)

textBold(...)

textItalic(...)

textBoldItalic(...)

parRight(...)

parLeft(...)

parCenter(...)

parJustify(...)

borderDotted(...)

borderDashed(...)

borderNone(...)

borderSolid(...)

cellBorderNone(...)

cellBorderBottom(...)

cellBorderTop(...)

cellBorderTB(...)
```

Arguments

...	further arguments passed to original functions.

Examples

```
textNormal()
textBold()
textItalic()
textBoldItalic()
parRight()
parLeft()
parLeft()
parLeft()
borderDotted()
borderDashed()
```

```
borderNone()
borderSolid()
cellBorderNone()
cellBorderBottom()
cellBorderTop()
cellBorderTB()
```

`textProperties` *Text formatting properties*

Description

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

Usage

```
textProperties(color = "black", font.size = getOption("ReporteRs-fontsize"),
  font.weight = "normal", font.style = "normal", underlined = FALSE,
  font.family = getOption("ReporteRs-default-font"),
  vertical.align = "baseline", shading.color)

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

## S3 method for class 'textProperties'
as.character(x, ...)

## S3 method for class 'textProperties'
chprop(object, color, font.size, font.weight,
  font.style, underlined, font.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.
`font.weight`	single character value specifying font weight (expected value is `normal` or `bold`).
`font.style`	single character value specifying font style (expected value is `normal` or `italic`).
`underlined`	single logical value specifying if the font is underlined.
`font.family`	single character value specifying font name (it has to be an existing font in the OS).
`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	textProperties object to print
...	further arguments - not used
object	textProperties object to update

Details

Get a modified version of a textProperties with chprop.

Value

a textProperties object

See Also

pot, alterFlexTable, shortcut_properties

Examples

```
textProperties( font.size = 12 )
textProperties(color="red", font.weight = "bold",
  font.style = "italic", underlined = TRUE )
textProperties( shading.color = "red" )
print( textProperties (color="red", font.size = 12) )

# chprop usage example ------
textProp <- textProperties()

chprop( textProp, color = "red" )
chprop( textProp, font.size = 12 )
chprop( textProp, font.weight = "bold" )
chprop( textProp, font.style = "italic" )
chprop( textProp, underlined = TRUE )
chprop( textProp, font.family = "Arial" )
chprop( textProp, vertical.align = "superscript" )
chprop( textProp, font.size = 12,
        font.weight = "bold", shading.color = "red" )
```

---

text_extract *Simple Text Extraction From a Word Document*

Description

Provides a simple method to get text from a docx document. It returns a character vector containing all chunk of text found in the document.

Usage

```
text_extract(x, body = TRUE, header = TRUE, footer = TRUE, bookmark)
```

Arguments

x	docx object
body	specifies to scan document body
header	specifies to scan document header
footer	specifies to scan document footer
bookmark	a character value ; id of the Word bookmark to scan.

Value

a character vector

See Also

docx, list_bookmarks

Examples

```
doc <- docx( title = "My example", template = file.path(
  system.file(package = "ReporteRs"), "templates/bookmark_example.docx") )
text_extract( doc )
text_extract( doc, header = FALSE, footer = FALSE )
text_extract( doc, bookmark = "author" )
```

---

toc.options *Set TOC options*

Description

Set custom table of contents options for a document object

Usage

```
toc.options(doc, ...)

## S3 method for class 'docx'
toc.options(doc, list.separator, ...)
```

Arguments

doc	Object of class docx
...	further arguments passed to other methods
list.separator	list separator (should be the same than in computer's regional settings)

Details

This function is to be used if TOC cannot be built. It is occuring when list separator used when building the TOC is different from the list separator in your computer's regional settings.

See entry 302865 of Microsoft knowledge database for more information.

Value

a document object

See Also

docx, addTOC.docx

docx, addTOC.docx

Examples

```
doc = docx( title = "title" )
doc = toc.options( doc, list.separator = "," )
```

---

vanilla.table *vanilla FlexTable shortcut*

Description

get a simple FlexTable from a dataset

Usage

```
vanilla.table(dataset, add.rownames = FALSE, text.direction = "lrtb")
```

Arguments

dataset	the data to use
add.rownames	logical value - should the row.names be included in the table.
text.direction	header cell text rotation - a single character value, expected value is one of "lrtb", "tbrl", "btlr".

See Also

FlexTable

Examples

```
ft <- vanilla.table( iris)
```

writeDoc *Write a document object*

Description

Write a document object into a file

Usage

```
writeDoc(doc, ...)

## S3 method for class 'docx'
writeDoc(doc, file, ...)

## S3 method for class 'pptx'
writeDoc(doc, file, ...)
```

Arguments

doc	document object
...	unused
file	single character value, name of the html file to write.

See Also

docx, pptx

Examples

```
doc <- docx()
writeDoc( doc, "ex_write_doc.docx")

doc <- pptx()
doc <- addSlide(doc, "Title and Content")
writeDoc( doc, "ex_write_doc.pptx")
```

[<-.FlexRow *modify FlexRow content*

Description

add or replace FlexCell into a FlexRow object

Usage

```
## S3 replacement method for class 'FlexRow'
x[i] <- value
```

Arguments

x	the FlexRow object
i	a single integer value.
value	an object of class FlexCell

See Also

FlexTable, addFlexTable, FlexRow , addHeaderRow, addFooterRow

[<-.FlexTable *alter FlexTable content and format*

Description

add text or format a FlexTable object.

Usage

```
## S3 replacement method for class 'FlexTable'
 x[i, j, text.properties,
  newpar = FALSE, byrow = FALSE, to = "body", side = "top"] <- value
```

Arguments

x	the FlexTable object
i	vector (integer index, row.names values or boolean vector) for rows selection.
j	vector (integer index, col.names values or boolean vector) for columns selection.
text.properties	formatting properties (an object of class textProperties). Used only when value is a data.frame, a maxtrix or a vector. It will be used to format added text.

newpar	logical value specifying wether or not the content should be added as a new paragraph (therefore added on a new line).
byrow	logical. If FALSE (the default) content is added by columns , otherwise content is added by rows.
to	specify on which part of the FlexTable to apply the value, must be one of the following values "body" (default) or "header" or "footer"
side	used only when value is a borderProperties, specify on which side to apply the properties. It must be one of "bottom", "top", "left", "right".
value	see details.

Details

Use `ft_object[1:4, 2:3] <- value` to perform the operation on the body subset of of the FlexTable.

Use `ft_object[] <- value` to perform the operation on the whole part (body, header or footer) of the FlexTable.

Use `ft_object[1, 2, to = "header"] <- value` to perform the operation on the header subset of the FlexTable.

Use `ft_object[1, 2, , to = "footer"] <- value` to perform the operation on the footer subset of the FlexTable.

To **format content**, argument value (the right side of the <-) should be one of the following:

- *for table cells:* an object of class cellProperties
- *for paragraphs contained in table cells:* an object of class parProperties
- *for text contained in table cells:* an object of class textProperties
- *for borders of table cells:* an object of class borderProperties

To **add content**, there are two options:

- *option 1:* value should be a data.frame or a matrix or a vector with as many elements as defined by the selection.
- *option 2:* value is a pot object, its value will be added in all cells defined by the selection.

If dealing with borderProperties objects, use also argument side to specify on which side of cells to apply border properties.

See Also

FlexTable, borderProperties , cellProperties, parProperties , textProperties

Examples

```
MyFTable <- FlexTable( data = mtcars[1:10, ], add.rownames=TRUE )
# modify the text formatting properties for the row.names column
MyFTable[ , 1] <- textProperties( font.style="italic", font.size = 9)
# align text to right for the row.names column
```

```
MyFTable[ , 1] <- parProperties( text.align = "right" )

# change cell formatting properties for various columns
MyFTable[ c(3,6:9), c( "mpg", "disp", "hp", "drat", "wt",
  "qsec" ) ] <- cellProperties( background.color="#CCCCCC")
# add text to elements of the column cyl
MyFTable[, "cyl", text.properties = textProperties(
  vertical.align="superscript", font.size = 9) ] <- " miles/gallon"
```

Index