Package 'knitr'

August 24, 2022

Type Package

Title A General-Purpose Package for Dynamic Report Generation in R

Version 1.40

Description Provides a general-purpose tool for dynamic report generation in R using Literate Programming techniques.

Depends R (>= 3.3.0)

Imports evaluate (>= 0.15), highr, methods, stringr (>= 0.6), yaml (>= 2.1.19), xfun (>= 0.29), tools

Suggests markdown, formatR, testit, digest, rgl (>= 0.95.1201), codetools, rmarkdown, htmlwidgets (>= 0.7), webshot, tikzDevice (>= 0.10), tinytex, reticulate (>= 1.4), JuliaCall (>= 0.11.1), magick, png, jpeg, gifski, xml2 (>= 1.2.0), httr, DBI (>= 0.4-1), showtext, tibble, sass, bslib, ragg, gridSVG, styler (>= 1.2.0), targets (>= 0.6.0)

License GPL

URL https://yihui.org/knitr/

BugReports https://github.com/yihui/knitr/issues

Encoding UTF-8

VignetteBuilder knitr

SystemRequirements Package vignettes based on R Markdown v2 or reStructuredText require Pandoc (http://pandoc.org). The function rst2pdf() requires rst2pdf (https://github.com/rst2pdf/rst2pdf).

Collate 'block.R' 'cache.R' 'utils.R' 'citation.R' 'hooks-html.R' 'plot.R' 'defaults.R' 'concordance.R' 'engine.R' 'highlight.R' 'themes.R' 'header.R' 'hooks-asciidoc.R' 'hooks-chunk.R' 'hooks-extra.R' 'hooks-latex.R' 'hooks-md.R' 'hooks-rst.R' 'hooks-textile.R' 'hooks.R' 'output.R' 'package.R' 'pandoc.R' 'params.R' 'parser.R' 'pattern.R' 'rocco.R' 'spin.R' 'table.R' 'template.R' 'utils-conversion.R' 'utils-rd2html.R' 'utils-sweave.R' 'utils-upload.R' 'utils-vignettes.R' 'zzz.R'

RoxygenNote 7.2.1

NeedsCompilation no

Author Yihui Xie [aut, cre] (<https://orcid.org/0000-0003-0645-5666>),
Abhraneel Sarma [ctb],
Adam Vogt [ctb],
Alastair Andrew [ctb],
Alex Zvoleff [ctb],
Amar Al-Zubaidi [ctb],
Andre Simon [ctb] (the CSS files under inst/themes/ were derived from the Highlight package http://www.andre-simon.de),
Aron Atkins [ctb],
Aaron Wolen [ctb],
Ashley Manton [ctb],
Atsushi Yasumoto [ctb] (<https://orcid.org/0000-0002-8335-495X>),
Ben Baumer [ctb],
Brian Diggs [ctb],
Brian Zhang [ctb],
Bulat Yapparov [ctb],
Cassio Pereira [ctb],
Christophe Dervieux [ctb],
David Hall [ctb],
David Hugh-Jones [ctb],
David Robinson [ctb],
Doug Hemken [ctb],
Duncan Murdoch [ctb],
Elio Campitelli [ctb],
Ellis Hughes [ctb],
Emily Riederer [ctb],
Fabian Hirschmann [ctb],
Fitch Simeon [ctb],
Forest Fang [ctb],
Frank E Harrell Jr [ctb] (the Sweavel package at inst/misc/Sweavel.sty),
Garrick Aden-Buie [ctb],
Gregoire Detrez [ctb],
Hadley Wickham [ctb],
Hao Zhu [ctb],
Heewon Jeon [ctb],
Henrik Bengtsson [ctb],
Hiroaki Yutani [ctb],
Ian Lyttle [ctb],
Hodges Daniel [ctb],
Jacob Bien [ctb],
Jake Burkhead [ctb],
James Manton [ctb],
Jared Lander [ctb],
Jason Punyon [ctb],
Javier Luraschi [ctb],
Jeff Arnold [ctb],

Jenny Bryan [ctb],
Jeremy Ashkenas [ctb, cph] (the CSS file at
inst/misc/docco-classic.css),
Jeremy Stephens [ctb],
Jim Hester [ctb],
Joe Cheng [ctb],
Johannes Ranke [ctb],
John Honaker [ctb],
John Muschelli [ctb],
Jonathan Keane [ctb],
JJ Allaire [ctb],
Johan Toloe [ctb],
Jonathan Sidi [ctb],
Joseph Larmarange [ctb],
Julien Barnier [ctb],
Kaiyin Zhong [ctb],
Kamil Slowikowski [ctb],
Karl Forner [ctb],
Kevin K. Smith [ctb],
Kirill Mueller [ctb],
Kohske Takahashi [ctb],
Lorenz Walthert [ctb],
Lucas Gallindo [ctb],
Marius Hofert [ctb],
Martin Modrák [ctb],
Michael Chirico [ctb],
Michael Friendly [ctb],
Michal Bojanowski [ctb],
Michel Kuhlmann [ctb],
Miller Patrick [ctb],
Nacho Caballero [ctb],
Nick Salkowski [ctb],
Niels Richard Hansen [ctb],
Noam Ross [ctb],
Obada Mahdi [ctb],
Pavel N. Krivitsky [ctb] (<https://orcid.org/0000-0002-9101-3362>),
Qiang Li [ctb],
Ramnath Vaidyanathan [ctb],
Richard Cotton [ctb],
Robert Krzyzanowski [ctb],
Rodrigo Copetti [ctb],
Romain Francois [ctb],
Ruaridh Williamson [ctb],
Sagiru Mati [ctb] (<https://orcid.org/0000-0003-1413-3974>),
Scott Kostyshak [ctb],
Sebastian Meyer [ctb],
Sietse Brouwer [ctb],
Simon de Bernard [ctb],

Sylvain Rousseau [ctb],
Taiyun Wei [ctb],
Thibaut Assus [ctb],
Thibaut Lamadon [ctb],
Thomas Leeper [ctb],
Tim Mastny [ctb],
Tom Torsney-Weir [ctb],
Trevor Davis [ctb],
Viktoras Veitas [ctb],
Weicheng Zhu [ctb],
Wush Wu [ctb],
Zachary Foster [ctb],
Zhian N. Kamvar [ctb] (<https://orcid.org/0000-0003-1458-7108>)

Maintainer Yihui Xie <xie@yihui.name>

Repository CRAN

Date/Publication 2022-08-24 15:30:03 UTC

R topics documented:

`knitr-package` *A general-purpose tool for dynamic report generation in R*

Description

The **knitr** package is an implementation of Literate Programming, a programming paradigm that intermingle code chunks (for computing) with prose (for documentation) in the same document.

Details

When the document is compiled, the code chunks can be executed, and the results from computing (text or graphics) are automatically written to the output along with the prose.

This package is an alternative tool to Sweave with a more flexible design and new features like caching and finer control of graphics. It is not limited to LaTeX and is ready to be customized to process other file formats. See the package website in the references for more information and examples.

Note

The pronunciation of **knitr** is similar to *neater* or you can think of *knitter* (but it is *single t*). The name comes from `knit` + `R` (while `Sweave` = `S` + `weave`).

Author(s)

Yihui Xie <https://yihui.org>

References

Full documentation and demos: https://yihui.org/knitr/; FAQ's: https://yihui.org/knitr/faq/

See Also

The core function in this package: `knit`. If you are an Sweave user, see `Sweave2knitr` on how to convert Sweave files to **knitr**.

`all_labels` *Get all chunk labels in a document*

Description

The function `all_labels()` returns all chunk labels as a character vector. Optionally, you can specify a series of conditions to filter the labels. The function 'all_rcpp_labels()' is a wrapper function for `all_labels(engine == 'Rcpp')`.

Usage

```
all_labels(...)

all_rcpp_labels(...)
```

Arguments

`...`	A vector of R expressions, each of which should return `TRUE` or `FALSE`. The expressions are evaluated using the *local* chunk options of each code chunk as the environment, which means global chunk options are not considered when evaluating these expressions. For example, if you set the global chunk option `opts_chunk$set(purl = TRUE)`, `all_labels(purl == TRUE)` will *not* return the labels of all code chunks, but will only return the labels of those code chunks that have local chunk options `purl = TRUE`.

Details

For example, suppose the condition expression is `engine == 'Rcpp'`, the object `engine` is the local chunk option `engine`. If an expression fails to be evaluated (e.g. when a certain object does not exist), `FALSE` is returned and the label for this chunk will be filtered out.

Value

A character vector.

Note

Empty code chunks are always ignored, including those chunks that are empty in the original document but filled with code using chunk options such as `ref.label` or `code`.

Examples

```
# the examples below are meaningless unless you put them in a knitr document
all_labels()
all_labels(engine == "Rcpp")
all_labels(echo == FALSE && results != "hide")
# or separate the two conditions
all_labels(echo == FALSE, results != "hide")
```

all_patterns *All built-in patterns*

Description

This object is a named list of all built-in patterns.

Usage

```
all_patterns
```

Format

An object of class `list` of length 8.

References

Usage: https://yihui.org/knitr/patterns/

See Also

knit_patterns

Examples

```
all_patterns$rnw
all_patterns$html

str(all_patterns)
```

asis_output *Mark an R object with a special class*

Description

This is a convenience function that assigns the input object a class named `knit_asis`, so that **knitr** will treat it as is (the effect is the same as the chunk option `results = 'asis'`) when it is written to the output.

Usage

```
asis_output(x, meta = NULL, cacheable = NA)
```

Arguments

x	An R object. Typically a character string, or an object which can be converted to a character string via as.character().
meta	Additional metadata of the object to be printed. The metadata will be collected when the object is printed, and accessible via knit_meta().
cacheable	Boolean indicating whether this object is cacheable. If FALSE, **knitr** will stop when caching is enabled on code chunks that contain asis_output().

Details

This function is normally used in a custom S3 method based on the printing function knit_print().

For the cacheable argument, you need to be careful when printing the object involves non-trivial side effects, in which case it is strongly recommended to use cacheable = FALSE to instruct **knitr** that this object should not be cached using the chunk option cache = TRUE, otherwise the side effects will be lost the next time the chunk is knitted. For example, printing a **shiny** input element or an HTML widget in an R Markdown document may involve registering metadata about some JavaScript libraries or stylesheets, and the metadata may be lost if we cache the code chunk, because the code evaluation will be skipped the next time. This particular issue has been solved in **knitr** after v1.13 (the metadata will be saved and loaded automatically when caching is enabled), but not all metadata can be saved and loaded next time and still works in the new R session.

Note

This function only works in top-level R expressions, and it will not work when it is called inside another expression, such as a for-loop. See https://github.com/yihui/knitr/issues/1137 for a discussion.

Examples

```
# see ?knit_print
```

---

cache_engines *Cache engines of other languages*

Description

This object controls how to load cached environments from languages other than R (when the chunk option engine is not 'R'). Each component in this object is a function that takes the current path to the chunk cache and loads it into the language environment.

Usage

```
cache_engines
```

Format

An object of class list of length 6.

Details

The cache engine function has one argument `options`, a list containing all chunk options. Note that `options$hash` is the path to the current chunk cache with the chunk's hash, but without any file extension, and the language engine may write a cache database to this path (with an extension).

The cache engine function should load the cache environment and should know the extension appropriate for the language.

References

See https://github.com/rstudio/reticulate/pull/167 for an implementation of a cache engine for Python.

`clean_cache` *Clean cache files that are probably no longer needed*

Description

If you remove or rename some cached code chunks, their original cache files will not be automatically cleaned. You can use this function to identify these possible files, and clean them if you are sure they are no longer needed.

Usage

```
clean_cache(clean = FALSE, path = opts_chunk$get("cache.path"))
```

Arguments

`clean`	Boolean; whether to remove the files.
`path`	Path to the cache.

Note

The identification is not guaranteed to be correct, especially when multiple documents share the same cache directory. You are recommended to call `clean_cache(FALSE)` and carefully check the list of files (if any) before you really delete them (`clean_cache(TRUE)`).

This function must be called within a code chunk in a source document, since it needs to know all chunk labels of the current document to determine which labels are no longer present, and delete cache corresponding to these labels.

`combine_words` *Combine multiple words into a single string*

Description

When a value from an inline R expression is a character vector of multiple elements, we may want to combine them into a phrase like 'a and b', or `a, b, and c`. That is what this a helper function does.

Usage

```
combine_words(
  words,
  sep = ", ",
  and = " and ",
  before = "",
  after = before,
  oxford_comma = TRUE
)
```

Arguments

`words`	A character vector.
`sep`	Separator to be inserted between words.
`and`	Character string to be prepended to the last word.
`before, after`	A character string to be added before/after each word.
`oxford_comma`	Whether to insert the separator between the last two elements in the list.

Details

If the length of the input `words` is smaller than or equal to 1, `words` is returned. When `words` is of length 2, the first word and second word are combined using the `and` string, or if blank, `sep` if is used. When the length is greater than 2, `sep` is used to separate all words, and the `and` string is prepended to the last word.

Value

A character string marked by `xfun::raw_string()`.

Examples

```
combine_words("a")
combine_words(c("a", "b"))
combine_words(c("a", "b", "c"))
combine_words(c("a", "b", "c"), sep = " / ", and = "")
combine_words(c("a", "b", "c"), and = "")
combine_words(c("a", "b", "c"), before = "\"", after = "\"")
combine_words(c("a", "b", "c"), before = "\"", after = "\"", oxford_comma = FALSE)
```

`convert_chunk_header` *Convert the in-header chunk option syntax to the in-body syntax*

Description

This is a helper function for moving chunk options from the chunk header to the chunk body using the new syntax.

Usage

```
convert_chunk_header(
  input,
  output = NULL,
  type = c("multiline", "wrap", "yaml"),
  width = 0.9 * getOption("width")
)
```

Arguments

`input`	File path to the document with code chunks to convert.
`output`	The default `NULL` will output to console. Other values can be a file path to write the converted content into or a function which takes `input` as argument and returns a file path to write into (e.g., `output = identity` to overwrite the input file).
`type`	This determines how the in-body options will be formatted. `"mutiline"` (the default, except for 'qmd' documents, for which the default is `"yaml"`) will write each chunk option on a separate line. Long chunk option values will be wrapped onto several lines, and you can use `width = 0` to keep one line per option only. `"wrap"` will wrap all chunk options together using `base::strwrap()`. `"yaml"` will convert chunk options to YAML.
`width`	An integer passed to `base::strwrap()` for `type = "wrap"` and `type = "multiline"`. If set to `0`, deactivate the wrapping (for `type = "multiline"` only).

Value

A character vector of converted `input` when `output = NULL`. The output file path with converted content otherwise.

About knitr option syntax

Historical chunk option syntax have chunk option in the chunk header using valid R syntax. This is an example for `.Rmd` document

````
```{r, echo = FALSE, fig.width: 10}
```
````

New syntax allows to pass option inside the chunk using several variants

- Passing options one per line using valid R syntax. This corresponds to `convert_chunk_header(type = "multiline")`.

  ```
  ```{r}
  #| echo = FALSE,
  #| fig.width = 10
  ```
  ```

- Passing option part from header in-chunk with several line if wrapping is needed. This corresponds to `convert_chunk_header(type = "wrap")`

  ```
  ```{r}
  #| echo = FALSE, fig.width = 10
  ```
  ```

- Passing options key value pairs in-chunk using YAML syntax. Values are no more R expression but valid YAML syntax. This corresponds to `convert_chunk_header(type = "yaml")` (not implement yet).

  ```
  ```{r}
  #| echo: false,
  #| fig.width: 10
  ```
  ```

Note

Learn more about the new chunk option syntax in https://yihui.org/en/2022/01/knitr-news/

Examples

```
knitr_example = function(...) system.file("examples", ..., package = "knitr")
# Convert a document for multiline type
convert_chunk_header(knitr_example("knitr-minimal.Rmd"))
# Convert a document for wrap type
convert_chunk_header(knitr_example("knitr-minimal.Rmd"), type = "wrap")
# Reduce default wrapping width
convert_chunk_header(knitr_example("knitr-minimal.Rmd"), type = "wrap", width = 0.6 *
    getOption("width"))
## Not run:
# Explicitly name the output
convert_chunk_header("test.Rmd", output = "test2.Rmd")
# Overwrite the input
convert_chunk_header("test.Rmd", output = identity)
# Use a custom function to name the output
convert_chunk_header("test.Rmd", output = function(f) sprintf("%s-new.%s",
    xfun::sans_ext(f), xfun::file_ext(f)))

## End(Not run)
```

current_input *Query the current input filename*

Description

Returns the name of the input file passed to `knit()`.

Usage

```
current_input(dir = FALSE)
```

Arguments

`dir`	Boolean; whether to prepend the current working directory to the file path, i.e. whether to return an absolute path or a relative path.

Value

A character string, if this function is called inside an input document. Otherwise `NULL`.

dep_auto *Build automatic dependencies among chunks*

Description

When the chunk option `autodep = TRUE`, all names of objects created in a chunk will be saved in a file named '`__objects`' and all global objects used in a chunk will be saved to '`__globals`'. This function can analyze object names in these files to automatically build cache dependencies, which is similar to the effect of the `dependson` option. It is supposed to be used in the first chunk of a document and this chunk must not be cached.

Usage

```
dep_auto(path = opts_chunk$get("cache.path"))
```

Arguments

`path`	Path to the dependency file.

Value

`NULL`. The dependencies are built as a side effect.

Note

Be cautious about `path`: because this function is used in a chunk, the working directory when the chunk is evaluated is the directory of the input document in `knit`, and if that directory differs from the working directory before calling `knit()`, you need to adjust the `path` argument here to make sure this function can find the cache files '`__objects`' and '`__globals`'.

References

https://yihui.org/knitr/demo/cache/

See Also

`dep_prev`

---

dep_prev *Make later chunks depend on previous chunks*

Description

This function can be used to build dependencies among chunks so that all later chunks depend on previous chunks, i.e. whenever the cache of a previous chunk is updated, the cache of all its later chunks will be updated.

Usage

```
dep_prev()
```

Value

`NULL`; the internal dependency structure is updated as a side effect.

References

https://yihui.org/knitr/demo/cache/

See Also

`dep_auto`

engine_output *An output wrapper for language engine output*

Description

If you have designed a language engine, you may call this function in the end to format and return the text output from your engine.

Usage

```
engine_output(options, code, out, extra = NULL)
```

Arguments

options	A list of chunk options. Usually this is just the object `options` passed to the engine function; see `knit_engines`.
code	Source code of the chunk, to which the output hook `source` is applied, unless the chunk option `echo` is `FALSE`.
out	Text output from the engine, to which the hook `output` is applied, unless the chunk option `results` is `'hide'`
extra	Any additional text output that you want to include.

Details

For expert users, an advanced usage of this function is `engine_output(options, out = LIST)` where `LIST` is a list that has the same structure as the output of `evaluate::evaluate()`. In this case, the arguments `code` and `extra` are ignored, and the list is passed to `knitr::sew()` to return a character vector of final output.

Value

A character string generated from the source code and output using the appropriate output hooks.

Examples

```
library(knitr)
engine_output(opts_chunk$merge(list(engine = "Rscript")),
    code = "1 + 1", out = "[1] 2")
engine_output(opts_chunk$merge(list(echo = FALSE, engine = "Rscript")),
    code = "1 + 1", out = "[1] 2")

# expert use only
engine_output(opts_chunk$merge(list(engine = "python")),
    out = list(structure(list(src = "1 + 1"), class = "source"),
        "2"))
```

extract_raw_output *Mark character strings as raw output that should not be converted*

Description

These functions provide a mechanism to protect the character output of R code chunks. The output is annotated with special markers in `raw_output`; `extract_raw_output()` will extract raw output wrapped in the markers, and replace the raw output with its MD5 digest; `restore_raw_output()` will restore the MD5 digest with the original raw output.

Usage

```
extract_raw_output(text, markers = raw_markers)

restore_raw_output(text, chunks, markers = raw_markers)

raw_output(x, markers = raw_markers, ...)
```

Arguments

`text`	For `extract_raw_output()`, the content of the input file (e.g. Markdown); for `restore_raw_output()`, the content of the output file (e.g. HTML generated by Pandoc from Markdown).
`markers`	A length-2 character vector to be used to wrap `x`; see `knitr:::raw_markers` for the default value.
`chunks`	A named character vector returned from `extract_raw_output()`.
`x`	The character vector to be protected.
`...`	Arguments to be passed to `asis_output()`.

Details

This mechanism is designed primarily for R Markdown pre/post-processors. In an R code chunk, you generate `raw_output()` to the Markdown output. In the pre-processor, you can `extract_raw_output()` from the Markdown file, store the raw output and MD5 digests, and remove the actual raw output from Markdown so Pandoc will never see it. In the post-processor, you can read the Pandoc output (e.g., an HTML or RTF file), and restore the raw output.

Value

For `extract_raw_output()`, a list of two components: `value` (the `text` with raw output replaced by MD5 digests) and `chunks` (a named character vector, of which the names are MD5 digests and values are the raw output). For `restore_raw_output()`, the restored `text`.

Examples

```
library(knitr)
out = c("*hello*", raw_output("<special>content</special> *protect* me!"),
    "*world*")
pre = extract_raw_output(out)
str(pre)
pre$value = gsub("[*]([^*]+)[*]", "<em>\\1</em>",
    pre$value)  # think this as Pandoc conversion
pre$value
# raw output was protected from the conversion
# (e.g. *protect* was not converted)
restore_raw_output(pre$value, pre$chunks)
```

`fig_chunk` *Obtain the figure filenames for a chunk*

Description

Given a chunk label, the figure file extension, the figure number(s), and the chunk option `fig.path`, return the filename(s).

Usage

```
fig_chunk(label, ext = "", number, fig.path = opts_chunk$get("fig.path"))
```

Arguments

`label`	The chunk label.
`ext`	The figure file extension, e.g. `png` or `pdf`.
`number`	The figure number (by default 1).
`fig.path`	Passed to `fig_path`. By default, the chunk option `fig.path` is used.

Details

This function can be used in an inline R expression to write out the figure filenames without hard-coding them. For example, if you created a plot in a code chunk with the label `foo` and figure path '`my-figure/`', you are not recommended to use hard-coded figure paths like '`\includegraphics{my-figure/foo-1.pdf}`' (in '`.Rnw`' documents) or '`![](my-figure/foo-1.png)`' (R Markdown) in your document. Instead, you should use '`\Sexpr{fig_chunk('foo', 'pdf')}`' or '`` ![](`r fig_chunk('foo', 'png')`) ``'.

You can generate plots in a code chunk but not show them inside the code chunk by using the chunk option `fig.show = 'hide'`. Then you can use this function if you want to show them elsewhere.

Value

A character vector of filenames.

Examples

```
library(knitr)
fig_chunk("foo", "png")
fig_chunk("foo", "pdf")
fig_chunk("foo", "svg", 2)  # the second plot of the chunk foo
fig_chunk("foo", "png", 1:5)  # if the chunk foo produced 5 plots
```

---

fig_path *Path for figure files*

Description

The filename of figure files is the combination of options `fig.path` and `label`. This function returns the path of figures for the current chunk by default.

Usage

```
fig_path(suffix = "", options = opts_current$get(), number)
```

Arguments

`suffix`	A filename suffix; if it is non-empty and does not contain a dot `.`, it will be treated as the filename extension (e.g. png will be used as `.png`)
`options`	A list of options; by default the options of the current chunk.
`number`	The current figure number. The default is the internal chunk option `fig.cur`, if this is available.

Value

A character vector of the form '`fig.path-label-i.suffix`'.

Note

When there are special characters (not alphanumeric or '`-`' or '`_`') in the path, they will be automatically replaced with '`_`'. For example, '`a b/c.d-`' will be sanitized to '`a_b/c_d-`'. This makes the filenames safe to LaTeX.

Examples

```
fig_path(".pdf", options = list(fig.path = "figure/abc-", label = "first-plot"))
fig_path(".png", list(fig.path = "foo-", label = "bar"), 1:10)
```

hook_ffmpeg_html *Hooks to create animations in HTML output*

Description

hook_ffmpeg_html() uses FFmpeg to convert images to a video; hook_gifski() uses the **gifski** to convert images to a GIF animation; hook_scianimator() uses the JavaScript library SciAnimator to create animations; hook_r2swf() uses the **R2SWF** package.

Usage

```
hook_ffmpeg_html(x, options)

hook_gifski(x, options)

hook_scianimator(x, options)

hook_r2swf(x, options)
```

Arguments

x	Filename for the plot (a character string).
options	A list of the current chunk options.

Details

These hooks are mainly for the package option animation.fun, e.g. you can set opts_knit$set(animation.fun = hook_scianimator).

hook_movecode *Some potentially useful document hooks*

Description

A document hook is a function to post-process the output document.

Usage

```
hook_movecode(x)
```

Arguments

x	A character string (the whole output document).

Details

hook_movecode() is a document hook to move code chunks out of LaTeX floating environments like 'figure' and 'table' when the chunks were actually written inside the floats. This function is primarily designed for LyX: we often insert code chunks into floats to generate figures or tables, but in the final output we do not want the code to float with the environments, so we use regular expressions to find out the floating environments, extract the code chunks and move them out. To disable this behavior, use a comment % knitr_do_not_move in the floating environment.

Value

The post-processed document as a character string.

Note

These functions are hackish. Also note hook_movecode() assumes you to use the default output hooks for LaTeX (not Sweave or listings), and every figure/table environment must have a label.

References

https://yihui.org/knitr/hooks/

Examples

```
## Not run:
knit_hooks$set(document = hook_movecode)

## End(Not run)
# see example 103 at https://github.com/yihui/knitr-examples
```

---

hook_pdfcrop *Built-in chunk hooks to extend knitr*

Description

Hook functions are called when the corresponding chunk options are not NULL to do additional jobs beside the R code in chunks. This package provides a few useful hooks, which can also serve as examples of how to define chunk hooks in **knitr**.

Usage

```
hook_pdfcrop(before, options, envir)

hook_optipng(before, options, envir)

hook_pngquant(before, options, envir)

hook_mogrify(before, options, envir)
```

```
hook_plot_custom(before, options, envir)

hook_purl(before, options, envir)
```

Arguments

before, options, envir
: See *References* below.

Details

The function hook_pdfcrop() calls plot_crop() to crop the white margins of PDF plots.

The function hook_optipng() calls the program optipng to optimize PNG images. Note the chunk option optipng can be used to provide additional parameters to the program optipng, e.g. optipng = '-o7'.

The function hook_pngquant() calls the program pngquant to optimize PNG images. Note the chunk option pngquant can be used to provide additional parameters to the program pngquant, e.g. pngquant = '--speed=1 --quality=0-50'.

The function hook_mogrify() calls the program mogrify. Note the chunk option mogrify can be used to provide additional parameters to the program mogrify (with default -trim to trim PNG files).

When the plots are not recordable via grDevices::recordPlot() and we save the plots to files manually via other functions (e.g. **rgl** plots), we can use the chunk hook hook_plot_custom to help write code for graphics output into the output document.

The hook hook_purl() can be used to write the code chunks to an R script. It is an alternative approach to purl, and can be more reliable when the code chunks depend on the execution of them (e.g. read_chunk(), or opts_chunk$set(eval = FALSE)). To enable this hook, it is recommended to associate it with the chunk option purl, i.e. knit_hooks$set(purl = hook_purl). When this hook is enabled, an R script will be written while the input document is being knit. Currently the code chunks that are not R code or have the chunk option purl=FALSE are ignored. Please note when the cache is turned on (the chunk option cache = TRUE), no chunk hooks will be executed, hence hook_purl() will not work, either. To solve this problem, we need cache = 2 instead of TRUE (see https://yihui.org/knitr/demo/cache/ for the meaning of cache = 2).

Note

The two hook functions hook_rgl() and hook_webgl() were moved from **knitr** to the **rgl** package (>= v0.95.1247) after **knitr** v1.10.5, and you can library(rgl) to get them.

References

https://yihui.org/knitr/hooks/#chunk_hooks

See Also

rgl::rgl.snapshot, rgl::rgl.postscript, rgl::hook_rgl, rgl::hook_webgl

Examples

```
if (require("rgl") && exists("hook_rgl")) knit_hooks$set(rgl = hook_rgl)
# then in code chunks, use the option rgl=TRUE
```

`hook_plot_html` *Default plot hooks for different output formats*

Description

These hook functions define how to mark up graphics output in different output formats.

Usage

```
hook_plot_html(x, options)

hook_plot_asciidoc(x, options)

hook_plot_tex(x, options)

hook_plot_md(x, options)

hook_plot_rst(x, options)

hook_plot_textile(x, options)
```

Arguments

`x`	Filename for the plot (a character string).
`options`	A list of the current chunk options.

Details

Depending on the options passed over, `hook_plot_tex` may return the normal '`\includegraphics{}`' command, or '`\input{}`' (for tikz files), or '`\animategraphics{}`' (for animations); it also takes many other options into consideration to align plots and set figure sizes, etc. Similarly, `hook_plot_html`, `hook_plot_md` and `hook_plot_rst` return character strings which are HTML, Markdown, reST code.

In most cases we do not need to call these hooks explicitly, and they were designed to be used internally. Sometimes we may not be able to record R plots using `grDevices::recordPlot()`, and we can make use of these hooks to insert graphics output in the output document; see `hook_plot_custom` for details.

Value

A character string of code, with plot filenames wrapped.

References

https://yihui.org/knitr/hooks/

See Also

hook_plot_custom

Examples

```
# this is what happens for a chunk like this

# <<foo-bar-plot, dev='pdf', fig.align='right'>>=
hook_plot_tex("foo-bar-plot.pdf", opts_chunk$merge(list(fig.align = "right")))

# <<bar, dev='tikz'>>=
hook_plot_tex("bar.tikz", opts_chunk$merge(list(dev = "tikz")))

# <<foo, dev='pdf', fig.show='animate', interval=.1>>=

# 5 plots are generated in this chunk
hook_plot_tex("foo5.pdf", opts_chunk$merge(list(fig.show = "animate", interval = 0.1,
    fig.cur = 5, fig.num = 5)))
```

---

image_uri *Encode an image file to a data URI*

Description

This function is the same as `xfun::base64_uri()` (only with a different function name). It can encode an image file as a base64 string, which can be used in the `img` tag in HTML.

Usage

```
image_uri(f)
```

Arguments

`f`	Path to the image file.

Value

The data URI as a character string.

Author(s)

Wush Wu and Yihui Xie

References

https://en.wikipedia.org/wiki/Data_URI_scheme

Examples

```
uri = image_uri(file.path(R.home("doc"), "html", "logo.jpg"))
if (interactive()) {
    cat(sprintf("<img src=\"%s\" />", uri), file = "logo.html")
    browseURL("logo.html")  # you can check its HTML source
}
```

---

imgur_upload *Upload an image to imgur.com*

Description

This function uses the **httr** package to upload a image to https://imgur.com, and parses the XML response to a list with **xml2** which contains information about the image in the Imgur website.

Usage

```
imgur_upload(file, key = "9f3460e67f308f6")
```

Arguments

file	Path to the image file to be uploaded.
key	Client ID for Imgur. By default, this uses a client ID registered by Yihui Xie.

Details

When the output format from knit() is HTML or Markdown, this function can be used to upload local image files to Imgur, e.g. set the package option opts_knit$set(upload.fun = imgur_upload), so the output document is completely self-contained, i.e. it does not need external image files any more, and it is ready to be published online.

Value

A character string of the link to the image; this string carries an attribute named XML which is a list converted from the response XML file; see Imgur API in the references.

Note

Please register your own Imgur application to get your client ID; you can certainly use mine, but this ID is in the public domain so everyone has access to all images associated to it.

Author(s)

Yihui Xie, adapted from the **imguR** package by Aaron Statham

References

Imgur API version 3: https://apidocs.imgur.com; a demo: https://yihui.org/knitr/demo/upload/

Examples

```
## Not run:
f = tempfile(fileext = ".png")
png(f)
plot(rnorm(100), main = R.version.string)
dev.off()

res = imgur_upload(f)
res  # link to original URL of the image
attr(res, "XML")  # all information
if (interactive())
    browseURL(res)

# to use your own key
opts_knit$set(upload.fun = function(file) imgur_upload(file, key = "your imgur key"))

## End(Not run)
```

---

include_graphics *Embed external images in* **knitr** *documents*

Description

When plots are not generated from R code, there is no way for **knitr** to capture plots automatically. In this case, you may generate the images manually and pass their file paths to this function to include them in the output. The major advantage of using this function is that it is portable in the sense that it works for all document formats that **knitr** supports, so you do not need to think if you have to use, for example, LaTeX or Markdown syntax, to embed an external image. Chunk options related to graphics output that work for normal R plots also work for these images, such as `out.width` and `out.height`.

Usage

```
include_graphics(
  path,
  auto_pdf = getOption("knitr.graphics.auto_pdf", FALSE),
  dpi = NULL,
  rel_path = getOption("knitr.graphics.rel_path", TRUE),
  error = getOption("knitr.graphics.error", TRUE)
)
```

Arguments

path	A character vector of image paths. Both local file paths and web paths are supported. Note that the auto_pdf and dpi arguments are not supported for web paths.
auto_pdf	Whether to use PDF images automatically when the output format is LaTeX. If TRUE, then e.g. 'foo/bar.png' will be replaced by 'foo/bar.pdf' if the latter exists. This can be useful since normally PDF images are of higher quality than raster images like PNG, when the output is LaTeX/PDF.
dpi	DPI (dots per inch) value. Used to calculate the output width (in inches) of the images. This will be their actual width in pixels, divided by dpi. If not provided, the chunk option dpi is used; if NA, the output width will not be calculated.
rel_path	Whether to automatically convert absolute paths to relative paths. If you know for sure that absolute paths work, you may set this argument or the global option knitr.graphics.rel_path to FALSE.
error	Whether to signal an error if any files specified in the path argument do not exist and are not web resources.

Value

The same as the input character vector path but it is marked with special internal S3 classes so that **knitr** will convert the file paths to proper output code according to the output format.

Note

This function is supposed to be used in R code chunks or inline R code expressions. For local images, you are recommended to use relative paths with forward slashes instead of backslashes (e.g., 'images/fig1.png' instead of '/Users/me/code/images/fig1.png').

The automatic calculation of the output width requires the **png** package (for PNG images) or the **jpeg** package (for JPEG images). The width will not be calculated if the chunk option out.width is already provided or dpi = NA.

---

include_url *Embed a URL as an HTML iframe or a screenshot in* **knitr** *documents*

Description

When the output format is HTML, include_url() inserts an iframe in the output; otherwise it takes a screenshot of the URL and insert the image in the output. include_app() takes the URL of a Shiny app and adds '?showcase=0' to it (to disable the showcase mode), then passes the URL to include_url().

Usage

```
include_url(url, height = "400px")

include_app(url, height = "400px")
```

Arguments

`url`	A character vector of URLs.
`height`	A character vector to specify the height of iframes.

Value

An R object with a special class that **knitr** recognizes internally to generate the iframes or screenshots.

See Also

`include_graphics`

`inline_expr` *Wrap code using the inline R expression syntax*

Description

This is a convenience function to write the "source code" of inline R expressions. For example, if you want to write ‘`` `r 1+1` ``’ literally in an R Markdown document, you may write ‘```` ``` `r knitr::inline_expr('1+1')` ``` ````’; for Rnw documents, this may be ‘`\verb|\Sexpr{knitr::inline_expr{'1+1'}}|`’.

Usage

```
inline_expr(code, syntax)
```

Arguments

`code`	Character string of the inline R source code.
`syntax`	A character string to specify the syntax, e.g. `rnw`, `html`, or `md`. If not specified, this will be guessed from the knitting context.

Value

A character string marked up using the inline R code syntax.

Examples

```
library(knitr)
inline_expr("1+1", "rnw")
inline_expr("1+1", "html")
inline_expr("1+1", "md")
```

`is_latex_output` *Check the current input and output type*

Description

The function `is_latex_output()` returns `TRUE` when the output format is LaTeX; it works for both '`.Rnw`' and R Markdown documents (for the latter, the two Pandoc formats `latex` and `beamer` are considered LaTeX output). The function `is_html_output()` only works for R Markdown documents and will test for several Pandoc HTML based output formats (by default, these formats are considered as HTML formats: `c('markdown', 'epub', 'epub2', 'html', 'html4', 'html5', 'revealjs', 's5','slideous', 'slidy', 'gfm')`).

Usage

```
is_latex_output()

is_html_output(fmt = pandoc_to(), excludes = NULL)

pandoc_to(fmt)

pandoc_from()
```

Arguments

`fmt`	A character vector of output formats to be checked against. If not provided, `is_html_output()` uses `pandoc_to()`, and `pandoc_to()` returns the output format name.
`excludes`	A character vector of output formats that should not be considered as HTML format. Options are: markdown, epub, epub2, html, html4, html5, revealjs, s5, slideous, slidy, and gfm.

Details

The function `pandoc_to()` returns the Pandoc output format, and `pandoc_from()` returns Pandoc input format. `pandoc_to(fmt)` allows to check the current output format against a set of format names. Both are to be used with R Markdown documents.

These functions may be useful for conditional output that depends on the output format. For example, you may write out a LaTeX table in an R Markdown document when the output format is LaTeX, and an HTML or Markdown table when the output format is HTML. Use `pandoc_to(fmt)` to test a more specific Pandoc format.

Internally, the Pandoc output format of the current R Markdown document is stored in `knitr::opts_knit$get('rmarkdown` and the Pandoc input format in `knitr::opts_knit$get('rmarkdown.pandoc.from')`

Note

See available Pandoc formats, in Pandoc's Manual

Examples

```
# check for output formats type
knitr::is_latex_output()
knitr::is_html_output()
knitr::is_html_output(excludes = c("markdown", "epub"))
# Get current formats
knitr::pandoc_from()
knitr::pandoc_to()
# Test if current output format is 'docx'
knitr::pandoc_to("docx")
```

---

is_low_change *Compare two recorded plots*

Description

Check if one plot only contains a low-level update of another plot.

Usage

```
is_low_change(p1, p2)
```

Arguments

p1, p2	Plot objects.

Value

Logical value indicating whether p2 is a low-level update of p1.

Examples

```
pdf(NULL)
dev.control("enable")  # enable plot recording
plot(1:10)
p1 = recordPlot()
abline(0, 1)  # add a line (a low-level change)
p2 = recordPlot()
plot(rnorm(100))
p3 = recordPlot()  # draw a completely new plot
dev.off()
knitr::is_low_change(p1, p2)  # true
knitr::is_low_change(p1, p3)  # false
```

kable *Create tables in LaTeX, HTML, Markdown and reStructuredText*

Description

A very simple table generator, and it is simple by design. It is not intended to replace any other R packages for making tables. The `kable()` function returns a single table for a single data object, and returns a table that contains multiple tables if the input object is a list of data objects. The `kables()` function is similar to `kable(x)` when `x` is a list of data objects, but `kables()` accepts a list of `kable()` values directly instead of data objects (see examples below).

Usage

```
kable(
  x,
  format,
  digits = getOption("digits"),
  row.names = NA,
  col.names = NA,
  align,
  caption = NULL,
  label = NULL,
  format.args = list(),
  escape = TRUE,
  ...
)

kables(x, format, caption = NULL, label = NULL)
```

Arguments

`x`	For `kable()`, `x` is an R object, which is typically a matrix or data frame. For `kables()`, a list with each element being a returned value from `kable()`.
`format`	A character string. Possible values are `latex`, `html`, `pipe` (Pandoc's pipe tables), `simple` (Pandoc's simple tables), and `rst`. The value of this argument will be automatically determined if the function is called within a **knitr** document. The `format` value can also be set in the global option `knitr.table.format`. If `format` is a function, it must return a character string.
`digits`	Maximum number of digits for numeric columns, passed to `round()`. This can also be a vector of length `ncol(x)`, to set the number of digits for individual columns.
`row.names`	Logical: whether to include row names. By default, row names are included if `rownames(x)` is neither `NULL` nor identical to `1:nrow(x)`.
`col.names`	A character vector of column names to be used in the table.

align	Column alignment: a character vector consisting of 'l' (left), 'c' (center) and/or 'r' (right). By default or if align = NULL, numeric columns are right-aligned, and other columns are left-aligned. If length(align) == 1L, the string will be expanded to a vector of individual letters, e.g. 'clc' becomes c('c', 'l', 'c'), unless the output format is LaTeX.
caption	The table caption.
label	The table reference label. By default, the label is obtained from knitr::opts_current$get('label'). To disable the label, use label = NA.
format.args	A list of arguments to be passed to format() to format table values, e.g. list(big.mark = ',').
escape	Boolean; whether to escape special characters when producing HTML or LaTeX tables. When escape = FALSE, you have to make sure that special characters will not trigger syntax errors in LaTeX or HTML.
...	Other arguments (see Examples and References).

Details

Missing values (NA) in the table are displayed as NA by default. If you want to display them with other characters, you can set the option knitr.kable.NA, e.g. options(knitr.kable.NA = '') to hide NA values.

Value

A character vector of the table source code.

Note

When using kable() as a *top-level* expression, you do not need to explicitly print() it due to R's automatic implicit printing. When it is wrapped inside other expressions (such as a for loop), you must explicitly print(kable(...)).

References

See https://bookdown.org/yihui/rmarkdown-cookbook/kable.html for some examples about this function, including specific arguments according to the format selected.

See Also

Other R packages such as **huxtable**, **xtable**, **kableExtra**, **gt** and **tables** for HTML and LaTeX tables, and **ascii** and **pander** for different flavors of markdown output and some advanced features and table styles. For more on other packages for creating tables, see https://bookdown.org/yihui/rmarkdown-cookbook/table-other.html.

Examples

```
d1 = head(iris)
d2 = head(mtcars)
# pipe tables by default
kable(d1)
```

```
kable(d2[, 1:5])
# no inner padding
kable(d2, format = "pipe", padding = 0)
# more padding
kable(d2, format = "pipe", padding = 2)
kable(d1, format = "latex")
kable(d1, format = "html")
kable(d1, format = "latex", caption = "Title of the table")
kable(d1, format = "html", caption = "Title of the table")
# use the booktabs package
kable(mtcars, format = "latex", booktabs = TRUE)
# use the longtable package
kable(matrix(1000, ncol = 5), format = "latex", digits = 2, longtable = TRUE)
# change LaTeX default table environment
kable(d1, format = "latex", caption = "My table", table.envir = "table*")
# add some table attributes
kable(d1, format = "html", table.attr = "id=\"mytable\"")
# reST output
kable(d2, format = "rst")
# no row names
kable(d2, format = "rst", row.names = FALSE)
# Pandoc simple tables
kable(d2, format = "simple", caption = "Title of the table")
# format numbers using , as decimal point, and ' as thousands separator
x = as.data.frame(matrix(rnorm(60, 1e+06, 10000), 10))
kable(x, format.args = list(decimal.mark = ",", big.mark = "'"))
# save the value
x = kable(d2, format = "html")
cat(x, sep = "\n")
# can also set options(knitr.table.format = 'html') so that the output is HTML

# multiple tables via either kable(list(x1, x2)) or kables(list(kable(x1),
# kable(x2)))
kable(list(d1, d2), caption = "A tale of two tables")
kables(list(kable(d1, align = "l"), kable(d2)), caption = "A tale of two tables")
```

`knit` *Knit a document*

Description

This function takes an input file, extracts the R code in it according to a list of patterns, evaluates the code and writes the output in another file. It can also tangle R source code from the input document (`purl()` is a wrapper to `knit(..., tangle = TRUE)`). The `knitr.purl.inline` option can be used to also tangle the code of inline expressions (disabled by default).

Usage

```
knit(
  input,
```

```
  output = NULL,
  tangle = FALSE,
  text = NULL,
  quiet = FALSE,
  envir = parent.frame(),
  encoding = "UTF-8"
)

purl(..., documentation = 1L)
```

Arguments

input	Path to the input file.
output	Path to the output file for knit(). If NULL, this function will try to guess a default, which will be under the current working directory.
tangle	Boolean; whether to tangle the R code from the input file (like utils::Stangle).
text	A character vector. This is an alternative way to provide the input file.
quiet	Boolean; suppress the progress bar and messages?
envir	Environment in which code chunks are to be evaluated, for example, parent.frame(), new.env(), or globalenv()).
encoding	Encoding of the input file; always assumed to be UTF-8 (i.e., this argument is effectively ignored).
...	arguments passed to knit() from purl()
documentation	An integer specifying the level of documentation to add to the tangled script. 0 means to output pure code, discarding all text chunks); 1 (the default) means to add the chunk headers to the code; 2 means to add all text chunks to code as roxygen comments.

Details

For most of the time, it is not necessary to set any options outside the input document; in other words, a single call like knit('my_input.Rnw') is usually enough. This function will try to determine many internal settings automatically. For the sake of reproducibility, it is better practice to include the options inside the input document (to be self-contained), instead of setting them before knitting the document.

First the filename of the output document is determined in this way: 'foo.Rnw' generates 'foo.tex', and other filename extensions like '.Rtex', '.Rhtml' ('.Rhtm') and '.Rmd' ('.Rmarkdown') will generate '.tex', '.html' and '.md' respectively. For other types of files, if the filename contains '_knit_', this part will be removed in the output file, e.g., 'foo_knit_.html' creates the output 'foo.html'; if '_knit_' is not found in the filename, 'foo.ext' will produce 'foo.txt' if ext is not txt, otherwise the output is 'foo-out.txt'. If tangle = TRUE, 'foo.ext' generates an R script 'foo.R'.

We need a set of syntax to identify special markups for R code chunks and R options, etc. The syntax is defined in a pattern list. All built-in pattern lists can be found in all_patterns (call it apat). First **knitr** will try to decide the pattern list based on the filename extension of the input document, e.g. 'Rnw' files use the list apat$rnw, 'tex' uses the list apat$tex, 'brew' uses apat$brew and

HTML files use `apat$html`; for unkown extensions, the content of the input document is matched against all pattern lists to automatically determine which pattern list is being used. You can also manually set the pattern list using the `knit_patterns` object or the `pat_rnw` series functions in advance and **knitr** will respect the setting.

According to the output format (`opts_knit$get('out.format')`), a set of output hooks will be set to mark up results from R (see `render_latex`). The output format can be LaTeX, Sweave and HTML, etc. The output hooks decide how to mark up the results (you can customize the hooks).

The name `knit` comes from its counterpart 'weave' (as in Sweave), and the name `purl` (as 'tangle' in Stangle) comes from a knitting method 'knit one, purl one'.

If the input document has child documents, they will also be compiled recursively. See `knit_child`.

See the package website and manuals in the references to know more about **knitr**, including the full documentation of chunk options and demos, etc.

Value

The compiled document is written into the output file, and the path of the output file is returned. If the `text` argument is not `NULL`, the compiled output is returned as a character vector. In other words, if you provide a file input, you get an output filename; if you provide a character vector input, you get a character vector output.

Note

The working directory when evaluating R code chunks is the directory of the input document by default, so if the R code involves external files (like `read.table()`), it is better to put these files under the same directory of the input document so that we can use relative paths. However, it is possible to change this directory with the package option `opts_knit$set(root.dir = ...)` so all paths in code chunks are relative to this `root.dir`. It is not recommended to change the working directory via `setwd()` in a code chunk, because it may lead to terrible consequences (e.g. figure and cache files may be written to wrong places). If you do use `setwd()`, please note that **knitr** will always restore the working directory to the original one. Whenever you feel confused, print `getwd()` in a code chunk to see what the working directory really is.

If the `output` argument is a file path, it is strongly recommended to be in the current working directory (e.g. 'foo.tex' instead of 'somewhere/foo.tex'), especially when the output has external dependencies such as figure files. If you want to write the output to a different directory, it is recommended to set the working directory to that directory before you knit a document. For example, if the source document is 'foo.Rmd' and the expected output is 'out/foo.md', you can write `setwd('out/'); knit('../foo.Rmd')` instead of `knit('foo.Rmd', 'out/foo.md')`.

N.B. There is no guarantee that the R script generated by `purl()` can reproduce the computation done in `knit()`. The `knit()` process can be fairly complicated (special values for chunk options, custom chunk hooks, computing engines besides R, and the `envir` argument, etc). If you want to reproduce the computation in a report generated by `knit()`, be sure to use `knit()`, instead of merely executing the R script generated by `purl()`. This seems to be obvious, but some people do not get it.

References

Package homepage: https://yihui.org/knitr/. The **knitr** main manual: and graphics manual.

See `citation('knitr')` for the citation information.

Examples

```
library(knitr)
(f = system.file("examples", "knitr-minimal.Rnw", package = "knitr"))
knit(f)  # compile to tex

purl(f)  # tangle R code
purl(f, documentation = 0)  # extract R code only
purl(f, documentation = 2)  # also include documentation

unlink(c("knitr-minimal.tex", "knitr-minimal.R", "figure"), recursive = TRUE)
```

---

knit2html *Convert markdown to HTML using knit() and markdownToHTML()*

Description

This is a convenience function to knit the input markdown source and call `markdown::markdownToHTML()` in the **markdown** package to convert the result to HTML.

Usage

```
knit2html(
  input,
  output = NULL,
  ...,
  envir = parent.frame(),
  text = NULL,
  quiet = FALSE,
  encoding = "UTF-8",
  force_v1 = getOption("knitr.knit2html.force_v1", FALSE)
)
```

Arguments

`input`	Path to the input file.
`output`	Path to the output file for `knit()`. If `NULL`, this function will try to guess a default, which will be under the current working directory.
`...`	Options passed to `markdown::markdownToHTML()`.
`envir`	Environment in which code chunks are to be evaluated, for example, `parent.frame()`, `new.env()`, or `globalenv()`).
`text`	A character vector. This is an alternative way to provide the input file.
`quiet`	Boolean; suppress the progress bar and messages?
`encoding`	Encoding of the input file; always assumed to be UTF-8 (i.e., this argument is effectively ignored).

`force_v1` Boolean; whether to force rendering the input document as an R Markdown v1 document, even if it is for v2.

Value

If the argument `text` is NULL, a character string (HTML code) is returned; otherwise the result is written into a file and the filename is returned.

Note

The **markdown** package is for R Markdown v1, which is much less powerful than R Markdown v2, i.e. the **rmarkdown** package (https://rmarkdown.rstudio.com). To render R Markdown v2 documents to HTML, please use `rmarkdown::render()` instead.

See Also

`knit`, `markdown::markdownToHTML`

Examples

```
# a minimal example
writeLines(c("# hello markdown", "```{r hello-random, echo=TRUE}", "rnorm(5)", "```"),
    "test.Rmd")
knit2html("test.Rmd")
if (interactive()) browseURL("test.html")

unlink(c("test.Rmd", "test.html", "test.md"))
```

---

knit2pandoc *Convert various input files to various output files using* `knit()` *and Pandoc*

Description

Knits the input file and compiles to an output format using Pandoc.

Usage

```
knit2pandoc(
  input,
  output = NULL,
  tangle = FALSE,
  text = NULL,
  quiet = FALSE,
  envir = parent.frame(),
  to = "html",
  pandoc_wrapper = NULL,
  ...,
  encoding = "UTF-8"
)
```

Arguments

input	Path to the input file.
output	Path to the output file for knit(). If NULL, this function will try to guess a default, which will be under the current working directory.
tangle	Boolean; whether to tangle the R code from the input file (like utils::Stangle).
text	A character vector. This is an alternative way to provide the input file.
quiet	Boolean; suppress the progress bar and messages?
envir	Environment in which code chunks are to be evaluated, for example, parent.frame(), new.env(), or globalenv()).
to	Character string giving the Pandoc output format to use.
pandoc_wrapper	An R function used to call Pandoc. If NULL (the default), rmarkdown::pandoc_convert() will be used if **rmarkdown** is installed, otherwise pandoc().
...	Options to be passed to the pandoc_wrapper function.
encoding	Ignored (always assumes UTF-8).

Value

Returns the output of the pandoc_wrapper function.

Author(s)

Trevor L. Davis

---

knit2pdf *Convert Rnw or Rrst files to PDF*

Description

Knit the input Rnw or Rrst document, and compile to PDF using tinytex::latexmk() or rst2pdf().

Usage

```
knit2pdf(
  input,
  output = NULL,
  compiler = NULL,
  envir = parent.frame(),
  quiet = FALSE,
  ...
)
```

Arguments

input	Path to the input file.
output	Path to the output file for knit(). If NULL, this function will try to guess a default, which will be under the current working directory.
compiler	A character string giving the LaTeX engine used to compile the tex document to PDF. For an Rrst file, setting compiler to 'rst2pdf' will use rst2pdf to compile the rst file to PDF using the ReportLab open-source library.
envir	Environment in which code chunks are to be evaluated, for example, parent.frame(), new.env(), or globalenv()).
quiet	Boolean; suppress the progress bar and messages?
...	Options to be passed to tinytex::latexmk() or rst2pdf().

Value

The filename of the PDF file.

Note

The output argument specifies the output filename to be passed to the PDF compiler (e.g. a tex document) instead of the PDF filename.

Author(s)

Ramnath Vaidyanathan, Alex Zvoleff and Yihui Xie

Examples

```
#' compile with xelatex
## knit2pdf(..., compiler = 'xelatex')

#' compile a reST file with rst2pdf
## knit2pdf(..., compiler = 'rst2pdf')
```

---

knit2wp *Knit an R Markdown document and post it to WordPress*

Description

This function is a wrapper around the **RWordPress** package. It compiles an R Markdown document to HTML and post the results to WordPress. Please note that **RWordPress** has not been updated for several years, which is not a good sign. For blogging with R, you may want to try the **blogdown** package instead.

Usage

```
knit2wp(
  input,
  title = "A post from knitr",
  ...,
  envir = parent.frame(),
  shortcode = FALSE,
  action = c("newPost", "editPost", "newPage"),
  postid,
  publish = TRUE
)
```

Arguments

input	Filename of the Rmd document.
title	Title of the post.
...	Other meta information of the post, e.g. categories = c('R','Stats') and mt_keywords = c('knitr', 'wordpress'), et cetera.
envir	Environment in which code chunks are to be evaluated, for example, parent.frame(), new.env(), or globalenv()).
shortcode	A length-2 logical vector: whether to use the shortcode '[sourcecode lang='lang']', which can be useful to WordPress.com users for syntax highlighting of source code and output. The first element applies to source code, and the second applies to text output. By default, both are FALSE.
action	Whether to create a new post, update an existing post, or create a new page.
postid	If action is editPost, the post id postid must be specified.
publish	Boolean: publish the post immediately?

Note

This function will convert the encoding of the post and the title to UTF-8 internally. If you have additional data to send to WordPress (e.g. keywords and categories), you may have to manually convert them to the UTF-8 encoding with the iconv(x, to = 'UTF-8') function (especially when using Windows).

Author(s)

William K. Morris, Yihui Xie, and Jared Lander

References

https://yihui.org/knitr/demo/wordpress/

Examples

```
# see the reference
```

knit_child *Knit a child document*

Description

This function knits a child document and returns a character string to input the result into the main document. It is designed to be used in the chunk option `child` and serves as the alternative to the `SweaveInput` command in Sweave.

Usage

```
knit_child(..., options = NULL, envir = knit_global())
```

Arguments

`...`	Arguments passed to `knit`.
`options`	A list of chunk options to be used as global options inside the child document. When one uses the `child` option in a parent chunk, the chunk options of the parent chunk will be passed to the `options` argument here. Ignored if not a list.
`envir`	Environment in which code chunks are to be evaluated, for example, `parent.frame()`, `new.env()`, or `globalenv()`).

Value

A character string of the content of the compiled child document is returned as a character string so it can be written back to the parent document directly.

Note

This function is not supposed be called directly like `knit()`; instead it must be placed in a parent document to let `knit()` call it indirectly.

The path of the child document is determined relative to the parent document.

References

https://yihui.org/knitr/demo/child/

Examples

```
# you can write \Sexpr{knit_child('child-doc.Rnw')} in an Rnw file 'main.Rnw'
# to input results from child-doc.Rnw in main.tex

# comment out the child doc by \Sexpr{knit_child('child-doc.Rnw', eval =
# FALSE)}
```

knit_code *The code manager to manage code in all chunks*

Description

This object provides methods to manage code (as character vectors) in all chunks in **knitr** source documents. For example, `knitr::knit_code$get()` returns a named list of all code chunks (the names are chunk labels), and `knitr::knit_code$get('foo')` returns the character vector of the code in the chunk with the label `foo`.

Usage

```
knit_code
```

Format

An object of class `list` of length 6.

Note

The methods on this object include the `set()` method (i.e., you could do something like `knitr::knit_code$set(foo = "'my precious new code'")`), but we recommend that you do not use this method to modify the content of code chunks, unless you are as creative as Emi Tanaka and know what you are doing.

knit_engines *Engines of other languages*

Description

This object controls how to execute the code from languages other than R (when the chunk option `engine` is not `'R'`). Each component in this object is a function that takes a list of current chunk options (including the source code) and returns a character string to be written into the output.

Usage

```
knit_engines
```

Format

An object of class `list` of length 6.

Details

The engine function has one argument `options`: the source code of the current chunk is in `options$code`. Usually we can call external programs to run the code via `system2`. Other chunk options are also contained in this argument, e.g. `options$echo` and `options$eval`, etc.

In most cases, `options$engine` can be directly used in command line to execute the code, e.g. `python` or `ruby`, but sometimes we may want to specify the path of the engine program, in which case we can pass it through the `engine.path` option. For example, `engine='ruby',engine.path='/usr/bin/ruby1.9.1'`. Additional command line arguments can be passed through `options$engine.opts`, e.g. `engine='ruby',engine.opts='-v`

See `str(knitr::knit_engines$get())` for a list of built-in language engines.

Note

The Leiningen engine `lein` requires lein-exec plugin; see https://github.com/yihui/knitr/issues/1176 for details.

References

Usage: https://yihui.org/knitr/objects/; examples: https://yihui.org/knitr/demo/engines/

Examples

```
knit_engines$get("python")
knit_engines$get("awk")
names(knit_engines$get())
```

---

knit_exit *Exit knitting early*

Description

Sometimes we may want to exit the knitting process early, and completely ignore the rest of the document. This function provides a mechanism to terminate `knit()`.

Usage

```
knit_exit(append, fully = TRUE)
```

Arguments

append	A character vector to be appended to the results from `knit()` so far. By default, this is '`\end{document}`' for LaTeX output, and '`</body></html>`' for HTML output, to make the output document complete. For other types of output, it is an empty string.
fully	Whether to fully exit the knitting process if `knit_exit()` is called from a child document. If `FALSE`, only exit the knitting process of the child document.

Value

Invisible NULL. An internal signal is set up (as a side effect) to notify knit() to quit as if it had reached the end of the document.

Examples

```
# see https://github.com/yihui/knitr-examples/blob/master/096-knit-exit.Rmd
```

---

knit_expand *A simple macro preprocessor for templating purposes*

Description

This function expands a template based on the R expressions in {{}} (this tag can be customized by the delim argument). These expressions are extracted, evaluated and replaced by their values in the original template.

Usage

```
knit_expand(file, ..., text = read_utf8(file), delim = c("{{", "}}"))
```

Arguments

file	The template file.
...	A list of variables to be used for the code in the template; note that the variables will be searched for in the parent frame as well.
text	Character vector of lines of code. An alternative way to specify the template code directly. If text is provided, file will be ignored.
delim	A pair of opening and closing delimiters for the templating tags.

Value

A character vector, with the tags evaluated and replaced by their values.

References

This function was inspired by the pyexpander and m4 (http://www.gnu.org/software/m4/), thanks to Frank Harrell.

Examples

```
# see the knit_expand vignette
if (interactive()) browseVignettes(package = "knitr")
```

knit_filter *Spell check filter for source documents*

Description

When performing spell checking on source documents, we may need to skip R code chunks and inline R expressions, because many R functions and symbols are likely to be identified as typos. This function is designed for the `filter` argument of `aspell()` to filter out code chunks and inline expressions.

Usage

```
knit_filter(ifile, encoding = "UTF-8")
```

Arguments

`ifile`	Filename of the source document.
`encoding`	Ignored (the file `ifile` must be encoded in UTF-8).

Value

A character vector of the file content, excluding code chunks and inline expressions.

Examples

```
library(knitr)
knitr_example = function(...) system.file("examples", ..., package = "knitr")

if (Sys.which("aspell") != "") {
    # -t means the TeX mode
    utils::aspell(knitr_example("knitr-minimal.Rnw"), knit_filter, control = "-t")

    # -H is the HTML mode
    utils::aspell(knitr_example("knitr-minimal.Rmd"), knit_filter, control = "-H -t")
}
```

knit_global *The global environment in which code chunks are evaluated*

Description

This function makes the environment of a code chunk accessible inside a chunk.

Usage

```
knit_global()
```

Details

It returns the `envir` argument of `knit`, e.g. if we call `knit()` in the global environment, `knit_global()` returns R's global environment by default. You can call functions like `ls()` on this environment.

---

knit_hooks *Hooks for R code chunks, inline R code and output*

Description

A hook is a function of a pre-defined form (arguments) that takes values of arguments and returns desired output. The object `knit_hooks` is used to access or set hooks in this package.

Usage

```
knit_hooks
```

Format

An object of class `list` of length 6.

References

Usage: https://yihui.org/knitr/objects/

Components in `knit_hooks`: https://yihui.org/knitr/hooks/

Examples

```
knit_hooks$get("source")
knit_hooks$get("inline")
```

---

knit_meta *Metadata about objects to be printed*

Description

As an object is printed, **knitr** will collect metadata about it (if available). After knitting is done, all the metadata is accessible via this function. You can manually add metadata to the **knitr** session via `knit_meta_add()`.

Usage

```
knit_meta(class = NULL, clean = TRUE)

knit_meta_add(meta, label = "")
```

Arguments

class	Optionally return only metadata entries that inherit from the specified class. The default, NULL, returns all entries.
clean	Whether to clean the collected metadata. By default, the metadata stored in **knitr** is cleaned up once retrieved, because we may not want the metadata to be passed to the next knit() call; to be defensive (i.e. not to have carryover metadata), you can call knit_meta() before knit().
meta	A metadata object to be added to the session.
label	A chunk label to indicate which chunk the metadata belongs to.

Value

knit_meta() returns the matched metadata specified by class; knit_meta_add() returns all current metadata.

---

knit_params *Extract knit parameters from a document*

Description

This function reads the YAML front-matter section of a document and returns a list of any parameters declared there. This function exists primarily to support the parameterized reports feature of the **rmarkdown** package, however is also used by the knitr purl function to include the default parameter values in the R code it emits.

Usage

```
knit_params(text, evaluate = TRUE)
```

Arguments

text	Character vector containing the document text.
evaluate	Boolean. If TRUE (the default), expression values embedded within the YAML will be evaluated. If FALSE, parameters defined with an expression will have the parsed but unevaluated expression in their value field.

Details

Parameters are included in YAML front matter using the params key. This key can have any number of subkeys each of which represents a parameter. For example:

```
---
title: My Document
output: html_document
params:
  frequency: 10
  show_details: true
---
```

Parameter values can be provided inline as illustrated above or can be included in a `value` sub-key. For example:

```
---
title: My Document
output: html_document
params:
  frequency:
    value: 10
---
```

This second form is useful when you need to provide additional details about the parameter (e.g. a `label` field as describe above).

You can also use R code to yield the value of a parameter by prefacing the value with `!r`, for example:

```
---
title: My Document
output: html_document
params:
  start: !r Sys.Date()
---
```

Value

List of objects of class `knit_param` that correspond to the parameters declared in the `params` section of the YAML front matter. These objects have the following fields:

`name` The parameter name.

`value` The default value for the parameter.

`expr` The R expression (if any) that yielded the default value.

In addition, other fields included in the YAML may also be present alongside the name, type, and value fields (e.g. a `label` field that provides front-ends with a human readable name for the parameter).

---

knit_params_yaml *Extract knit parameters from YAML text*

Description

This function reads the YAML front-matter that has already been extracted from a document and returns a list of any parameters declared there.

Usage

```
knit_params_yaml(yaml, evaluate = TRUE)
```

Arguments

yaml	Character vector containing the YAML text.
evaluate	If TRUE (the default) expression values embedded within the YAML will be evaluated. If FALSE, parameters defined with an expression will have the parsed but unevaluated expression in their value field.

Value

List of objects of class knit_param that correspond to the parameters declared in the params section of the YAML. See knit_params for a full description of these objects.

See Also

knit_params

---

knit_patterns *Patterns to match and extract R code in a document*

Description

Patterns are regular expressions and will be used in functions like base::grep() to extract R code and chunk options. The object knit_patterns controls the patterns currently used; see the references and examples for usage. All built-in patterns are available in the list all_patterns.

Usage

```
knit_patterns
```

Format

An object of class list of length 6.

References

Usage: https://yihui.org/knitr/objects/

Components in knit_patterns: https://yihui.org/knitr/patterns/

See Also

all_patterns

Examples

```
library(knitr)
opat = knit_patterns$get()  # old pattern list (to restore later)

apats = all_patterns  # a list of all built-in patterns
str(apats)
knit_patterns$set(apats[["rnw"]])  # set pattern list from apats

knit_patterns$get(c("chunk.begin", "chunk.end", "inline.code"))

# a customized pattern list; has to empty the original patterns first!
knit_patterns$restore()
# we may want to use this in an HTML document
knit_patterns$set(list(chunk.begin = "<!--helloR\\s+(.*)", chunk.end = "^byeR-->"))
str(knit_patterns$get())

knit_patterns$set(opat)  # put the old patterns back
```

`knit_print` *A custom printing function*

Description

The S3 generic function `knit_print` is the default printing function in **knitr**. The chunk option `render` uses this function by default. The main purpose of this S3 generic function is to customize printing of R objects in code chunks. We can fall back to the normal printing behavior by setting the chunk option `render = normal_print`.

Usage

```
knit_print(x, ...)

normal_print(x, ...)
```

Arguments

`x`	An R object to be printed
`...`	Additional arguments passed to the S3 method. Currently ignored, except two optional arguments `options` and `inline`; see the references below.

Details

Users can write custom methods based on this generic function. For example, if we want to print all data frames as tables in the output, we can define a method `knit_print.data.frame` that turns a data.frame into a table (the implementation may use other R packages or functions, e.g. **xtable** or `kable()`).

Value

The value returned from the print method should be a character vector or can be converted to a character value. You can wrap the value in `asis_output()` so that **knitr** writes the character value as is in the output.

Note

It is recommended to leave a `...` argument in your method, to allow future changes of the `knit_print()` API without breaking your method.

References

See `vignette('knit_print', package = 'knitr')`.

Examples

```
library(knitr)
# write tables for data frames
knit_print.data.frame = function(x, ...) {
    res = paste(c("", "", kable(x, output = FALSE)), collapse = "\n")
    asis_output(res)
}
# register the method
registerS3method("knit_print", "data.frame", knit_print.data.frame)
# after you define and register the above method, data frames will be printed
# as tables in knitr, which is different with the default print() behavior
```

`knit_rd` *Knit package documentation*

Description

Run examples in a package and insert output into the examples code; `knit_rd_all()` is a wrapper around `knit_rd()` to build static HTML help pages for all packages under the '`html`' directory of them.

Usage

```
knit_rd(pkg, links = tools::findHTMLlinks(), frame = TRUE)

knit_rd_all()
```

Arguments

`pkg`	Package name.
`links`	A character vector of links to be passed to `tools::Rd2HTML()`.
`frame`	Boolean: whether to put a navigation frame on the left of the index page.

Value

All HTML pages corresponding to topics in the package are written under the current working directory. An 'index.html' is also written as a table of content.

Note

Ideally the html pages should be put under the 'html' directory of an installed package which can be found via system.file('html', package = 'your_package_name'), otherwise some links may not work (e.g. the link to the DESCRITION file).

Examples

```
library(knitr)
## Not run:

knit_rd("maps")
knit_rd("rpart")
setwd(system.file("html", package = "ggplot2"))
knit_rd("ggplot2")  # time-consuming!

knit_rd_all()  # this may take really long time if you have many packages installed

## End(Not run)
```

---

knit_theme *Syntax highlighting themes*

Description

This object can be used to set or get themes in **knitr** for syntax highlighting.

Usage

```
knit_theme
```

Format

An object of class list of length 2.

Details

We can use knit_theme$set(theme) to set the theme, and knit_theme$get(theme) to get a theme. The theme is a character string for both methods (either the name of the theme, or the path to the CSS file of a theme), and for the set() method, it can also be a list returned by the get() method. See examples below.

Note

The syntax highlighting here only applies to '.Rnw' (LaTeX) and '.Rhtml' (HTML) documents, and it does not work for other types of documents, such as '.Rmd' (R Markdown, which has its own syntax highlighting themes; see https://rmarkdown.rstudio.com).

Author(s)

Ramnath Vaidyanathan and Yihui Xie

References

For a preview of all themes, see https://gist.github.com/yihui/3422133.

Examples

```
opts_knit$set(out.format = "latex")
knit_theme$set("edit-vim")

knit_theme$get()  # names of all available themes

thm = knit_theme$get("acid")  # parse the theme to a list
knit_theme$set(thm)

opts_knit$set(out.format = NULL)  # restore option
```

---

knit_watch *Watch an input file continuously and knit it when it is updated*

Description

Check the modification time of an input file continously in an infinite loop. Whenever the time indicates the file has been modified, call a function to recompile the input file.

Usage

```
knit_watch(input, compile = knit, interval = 1, ...)
```

Arguments

input	An input file path, or a character vector of mutiple input file paths.
compile	A function to compile the input file. This could be e.g. knit or knit2pdf, depending on the input file and the output you want.
interval	A time interval to pause in each cycle of the infinite loop.
...	Other arguments to be passed to the compile function.

Details

This is actually a general function not necessarily restricted to applications in **knitr**. You may specify any `compile` function to process the `input` file. To stop the infinite loop, press the 'Escape' key or 'Ctrl + C' (depending on your editing environment and operating system).

Examples

```
# knit_watch('foo.Rnw', knit2pdf)

# knit_watch('foo.Rmd', rmarkdown::render)
```

---

load_cache *Load the cache database of a code chunk*

Description

If a code chunk has turned on the chunk option `cache = TRUE`, a cache database will be established after the document is compiled. You can use this function to manually load the database anywhere in the document (even before the code chunk). This makes it possible to use objects created later in the document earlier, e.g. in an inline R expression before the cached code chunk, which is normally not possible because **knitr** compiles the document in a linear fashion, and objects created later cannot be used before they are created.

Usage

```
load_cache(
  label,
  object,
  notfound = "NOT AVAILABLE",
  path = opts_chunk$get("cache.path"),
  dir = opts_knit$get("output.dir"),
  envir = NULL,
  lazy = TRUE
)
```

Arguments

`label`	The chunk label of the code chunk that has a cache database.
`object`	The name of the object to be fetched from the database. If it is missing, `NULL` is returned).
`notfound`	A value to use when the `object` cannot be found.
`path`	Path of the cache database (normally set in the global chunk option `cache.path`).
`dir`	Path to use as the working directory. Defaults to the output directory if run inside a **knitr** context and to the current working directory otherwise. Any relative `path` is defined from `dir`.

envir	Environment to use for cache loading, into which all objects in the cache for the specified chunk (not just that in object) will be loaded. Defaults to the value in knit_global.
lazy	Whether to lazyLoad the cache database (depending on the chunk option cache.lazy = TRUE or FALSE of that code chunk).

Value

Invisible NULL when object is not specified (the cache database will be loaded as a side effect), otherwise the value of the object if found.

Note

Apparently this function loads the value of the object from the *previous* run of the document, which may be problematic when the value of the object becomes different the next time the document is compiled. Normally you must compile the document twice to make sure the cache database is created, and the object can be read from it. Please use this function with caution.

References

See the example #114 at https://github.com/yihui/knitr-examples.

---

opts_chunk *Default and current chunk options*

Description

Options for R code chunks. When running R code, the object opts_chunk (default options) is not modified by chunk headers (local chunk options are merged with default options), whereas opts_current (current options) changes with different chunk headers and it always reflects the options for the current chunk.

Usage

```
opts_chunk

opts_current
```

Format

An object of class list of length 6.

An object of class list of length 6.

Details

Normally we set up the global options once in the first code chunk in a document using `opts_chunk$set()`, so that all *latter* chunks will use these options. Note the global options set in one chunk will not affect the options in this chunk itself, and that is why we often need to set global options in a separate chunk.

See `str(knitr::opts_chunk$get())` for a list of default chunk options.

Note

`opts_current` should be treated as read-only and you are supposed to only query its values via `opts_current$get()`. Technically you could also call `opts_current$set()` to change the values, but you are not recommended to do so unless you understand the consequences.

References

Usage: https://yihui.org/knitr/objects/

A list of available options: https://yihui.org/knitr/options/#chunk-options

Examples

```
opts_chunk$get("prompt")
opts_chunk$get("fig.keep")
```

---

opts_hooks *Hooks for code chunk options*

Description

Like `knit_hooks`, this object can be used to set hook functions to manipulate chunk options.

Usage

```
opts_hooks
```

Format

An object of class `list` of length 6.

Details

For every code chunk, if the chunk option named, say, `FOO`, is not `NULL`, and a hook function with the same name has been set via `opts_hooks$set(FOO = function(options) { options })` (you can manipuate the `options` argument in the function and return it), the hook function will be called to update the chunk options.

References

https://yihui.org/knitr/hooks/

Examples

```
# make sure the figure width is no smaller than fig.height
opts_hooks$set(fig.width = function(options) {
    if (options$fig.width < options$fig.height) {
        options$fig.width = options$fig.height
    }
    options
})
# remove all hooks
opts_hooks$restore()
```

`opts_knit` *Options for the knitr package*

Description

Options including whether to use a progress bar when knitting a document, and the base directory of images, etc.

Usage

```
opts_knit
```

Format

An object of class `list` of length 6.

Details

Besides the standard usage (`opts_knit$set()`), we can also set package options prior to loading `knitr` or calling `knit()` using `options()` in base R. A global option `knitr.package.foo` in `options()` will be set as an option `foo` in `opts_knit`, i.e. global options in base R with the prefix `knitr.package.` correspond to options in `opts_knit`. This can be useful to set package options in '`~/.Rprofile`' without loading **knitr**.

See `str(knitr::opts_knit$get())` for a list of default package options.

References

Usage: https://yihui.org/knitr/objects/

A list of available options: https://yihui.org/knitr/options/#package_options

Examples

```
opts_knit$get("verbose")
opts_knit$set(verbose = TRUE)  # change it
if (interactive()) {
    # for unnamed chunks, use 'fig' as the figure prefix
    opts_knit$set(unnamed.chunk.label = "fig")
    knit("001-minimal.Rmd")  # from https://github.com/yihui/knitr-examples
}
```

---

opts_template *Template for creating reusable chunk options*

Description

Creates a template binding a label to a set of chunk options. Every chunk that references the template label will have the specified set of options applied to it.

Usage

```
opts_template
```

Format

An object of class `list` of length 6.

Examples

```
opts_template$set(myfigures = list(fig.height = 4, fig.width = 4))
# later you can reuse these chunk options by 'opts.label', e.g.

# <<foo, opts.label='myfigures'>>=

# the above is equivalent to <<foo, fig.height=4, fig.width=4>>=
```

---

pandoc *A Pandoc wrapper to convert documents to other formats*

Description

This function calls Pandoc to convert documents to other formats such as HTML, LaTeX/PDF and Word, etc, (optionally) based on a configuration file or in-file configurations which specify the options to use for Pandoc.

Usage

```
pandoc(input, format, config = getOption("config.pandoc"), ext = NA)
```

Arguments

input	A character vector of Markdown filenames (must be encoded in UTF-8).
format	Name of the output format (see References). This can be a character vector of multiple formats; by default, it is obtained from the t field in the configuration. If the configuration is empty or the t field is not found, the default output format will be 'html'.
config	Path to the Pandoc configuration file. If missing, it is assumed to be a file with the same base name as the input file and an extension .pandoc (e.g. for 'foo.md' it looks for 'foo.pandoc')
ext	Filename extensions. By default, the extension is inferred from the format, e.g. latex creates pdf, dzslides creates html, and so on

Details

There are two ways to input the Pandoc configurations – through a config file, or embed the configurations in the input file as special comments between <!--pandoc and -->.

The configuration file is a DCF file (see read.dcf). This file must contain a field named t which means the output format. The configurations are written in the form of tag:value and passed to Pandoc (if no value is needed, just leave it empty, e.g. the option standalone or s for short). If there are multiple output formats, write each format and relevant configurations in a block, and separate blocks with blank lines.

If there are multiple records of the t field in the configuration, the input markdown file will be converted to all these formats by default, unless the format argument is specified as one single format.

Value

The output filename(s) (or an error if the conversion failed).

References

Pandoc: https://pandoc.org; Examples and rules of the configurations: https://yihui.org/knitr/demo/pandoc/

Also see R Markdown (v2) at https://rmarkdown.rstudio.com. The **rmarkdown** package has several convenience functions and templates that make it very easy to use Pandoc. The RStudio IDE also has comprehensive support for it, so I'd recommend users who are not familiar with command-line tools to use the **rmarkdown** package instead.

See Also

read.dcf

Examples

```
system("pandoc -h")  # see possible output formats
```

partition_chunk *Partition chunk options from the code chunk body*

Description

Chunk options can be written in special comments (e.g., after `#|` for R code chunks) inside a code chunk. This function partitions these options from the chunk body.

Usage

```
partition_chunk(engine, code)
```

Arguments

engine	The name of the language engine (to determine the appropriate comment character).
code	A character vector (lines of code).

Value

A list with the following items:

`options` The parsed options (if any) as a list.

`src` The part of the input that contains the options.

`code` The part of the input that contains the code.

Note

Chunk options must be written on *continuous* lines (i.e., all lines must start with the special comment prefix such as `#|`) at the beginning of the chunk body.

Examples

```
# parse yaml-like items
yaml_like = c("#| label: mine", "#| echo: true", "#| fig.width: 8", "#| foo: bar",
    "1 + 1")
writeLines(yaml_like)
knitr::partition_chunk("r", yaml_like)

# parse CSV syntax
csv_like = c("#| mine, echo = TRUE, fig.width = 8, foo = 'bar'", "1 + 1")
writeLines(csv_like)
knitr::partition_chunk("r", csv_like)
```

pat_rnw *Set regular expressions to read input documents*

Description

These are convenience functions to set pre-defined pattern lists (the syntax to read input documents). The function names are built from corresponding file extensions, e.g. `pat_rnw()` can set the Sweave syntax to read Rnw documents.

Usage

```
pat_rnw()

pat_brew()

pat_tex()

pat_html()

pat_md()

pat_rst()

pat_asciidoc()

pat_textile()
```

Value

The patterns object `knit_patterns` is modified as a side effect.

Examples

```
# see how knit_patterns is modified
knit_patterns$get()
pat_rnw()
knit_patterns$get()

knit_patterns$restore()  # empty the list
```

plot_crop *Crop a plot (remove the edges) using PDFCrop or ImageMagick*

Description

The program `pdfcrop` (often shipped with a LaTeX distribution) is executed on a PDF plot file, and `magick::image_trim()` is executed for other types of plot files.

Usage

```
plot_crop(x, quiet = TRUE)
```

Arguments

`x`	Filename of the plot.
`quiet`	Whether to suppress standard output from the command.

Details

The program `pdfcrop` can crop the extra white margins when the plot format is PDF, to make better use of the space in the output document, otherwise we often have to struggle with `graphics::par()` to set appropriate margins. Note `pdfcrop` often comes with a LaTeX distribution such as TinyTeX, MiKTeX, or TeX Live, and you may not need to install it separately (use `Sys.which('pdfcrop')` to check it; if it not empty, you are able to use it). Note that `pdfcrop` depends on GhostScript. You can check if GhostScript is installed via `tools::find_gs_cmd()`.

Value

The original filename.

References

PDFCrop: https://www.ctan.org/pkg/pdfcrop. If you use TinyTeX, you may install `pdfcrop` with `tinytex::tlmgr_install('pdfcrop')`.

rand_seed *An unevaluated expression to return .Random.seed if exists*

Description

This expression returns `.Random.seed` when `eval(rand_seed)` and `NULL` otherwise.

Usage

```
rand_seed
```

Details

It is designed to work with `opts_chunk$set(cache.extra = rand_seed)` for reproducibility of chunks that involve with random number generation. See references.

References

https://yihui.org/knitr/demo/cache/

Examples

```
eval(rand_seed)
rnorm(1)  # .Random.seed is created (or modified)
eval(rand_seed)
```

---

raw_block *Mark character strings as raw blocks in R Markdown*

Description

Wraps content in a raw attribute block, which protects it from being escaped by Pandoc. See https://pandoc.org/MANUAL.html#generic-raw-attribute. Functions `raw_latex()` and `raw_html()` are shorthands of `raw_block(x, 'latex')` and `raw_block(x, 'html')`, respectively.

Usage

```
raw_block(x, type = "latex", ...)

raw_latex(x, ...)

raw_html(x, ...)
```

Arguments

x	The character vector to be protected.
type	The type of raw blocks (i.e., the Pandoc output format). If you are not sure about the Pandoc output format of your document, insert a code chunk `knitr:::pandoc_to()` and see what it returns after the document is compiled.
...	Arguments to be passed to `asis_output()`.

Examples

```
knitr::raw_latex("\\emph{some text}")
```

read_chunk *Read chunks from an external script*

Description

Chunks can be put in an external script, and this function reads chunks into the current **knitr** session; `read_demo()` is a convenience function to read a demo script from a package.

Usage

```
read_chunk(
  path,
  lines = read_utf8(path),
  labels = NULL,
  from = NULL,
  to = NULL,
  from.offset = 0L,
  to.offset = 0L,
  roxygen_comments = TRUE
)

read_demo(topic, package = NULL, ...)
```

Arguments

`path`	Path to the R script.
`lines`	Character vector of lines of code. By default, this is read from `path`.
`labels`	Character vector of chunk labels (default `NULL`).
`from, to`	Numeric vector specifying the starting/ending line numbers of code chunks, or a character vector; see Details.
`from.offset, to.offset`	Offsets to be added to `from`/`to`.
`roxygen_comments`	Logical dictating whether to keep trailing roxygen-style comments from code chunks in addition to whitespace
`topic, package`	Name of the demo and the package. See `utils::demo`.
`...`	Arguments passed to `read_chunk`.

Details

There are two approaches to read external code into the current session: (1) Use a special separator of the from `## ---- chunk-label` (at least four dashes before the chunk label) in the script; (2) Manually specify the labels, starting and ending positions of code chunks in the script.

The second approach will be used only when `labels` is not `NULL`. For this approach, if `from` is `NULL`, the starting position is 1; if `to` is `NULL`, each of its element takes the next element of `from` minus 1,

and the last element of `to` will be the length of `lines` (e.g. when `from = c(1, 3, 8)` and the script has 10 lines in total, `to` will be `c(2, 7, 10)`). Alternatively, `from` and `to` can be character vectors as regular expressions to specify the positions; when their length is 1, the single regular expression will be matched against the `lines` vector, otherwise each element of `from`/`to` is matched against `lines` and the match is supposed to be unique so that the numeric positions returned from `grep()` will be of the same length of `from`/`to`. Note `labels` always has to match the length of `from` and `to`.

Value

As a side effect, code chunks are read into the current session so that future chunks can (re)use the code by chunk label references. If an external chunk has the same label as a chunk in the current session, chunk label references by future chunks will refer to the external chunk.

Note

This function can only be used in a chunk which is *not* cached (chunk option `cache = FALSE`), and the code is read and stored in the current session *without* being executed (to actually run the code, you have to use a chunk with a corresponding label).

Author(s)

Yihui Xie; the idea of the second approach came from Peter Ruckdeschel (author of the **SweaveListingUtils** package)

References

https://yihui.org/knitr/demo/externalization/

Examples

```
## put this in foo.R and read_chunk('foo.R')

## ---- my-label ----
1 + 1
lm(y ~ x, data = data.frame(x = 1:10, y = rnorm(10)))

## later you can use <<my-label>>= to reference this chunk

## the 2nd approach
code = c("#@a", "1+1", "#@b", "#@a", "rnorm(10)", "#@b")
read_chunk(lines = code, labels = "foo")  # put all code into one chunk named foo
read_chunk(lines = code, labels = "foo", from = 2, to = 2)  # line 2 into chunk foo
read_chunk(lines = code, labels = c("foo", "bar"), from = c(1, 4), to = c(3, 6))
# automatically figure out 'to'
read_chunk(lines = code, labels = c("foo", "bar"), from = c(1, 4))
read_chunk(lines = code, labels = c("foo", "bar"), from = "^#@a", to = "^#@b")
read_chunk(lines = code, labels = c("foo", "bar"), from = "^#@a", to = "^#@b",
    from.offset = 1, to.offset = -1)

## later you can use, e.g., <<foo>>=
knitr::knit_code$get()  # use this to check chunks in the current session
knitr::knit_code$restore()  # clean up the session
```

read_rforge *Read source code from R-Forge*

Description

This function reads source code from the SVN repositories on R-Forge.

Usage

```
read_rforge(path, project, extra = "")
```

Arguments

path	Relative path to the source script on R-Forge.
project	Name of the R-Forge project.
extra	Extra parameters to be passed to the URL (e.g. `extra = '&revision=48'` to check out the source of revision 48).

Value

A character vector of the source code.

Author(s)

Yihui Xie and Peter Ruckdeschel

Examples

```
library(knitr)
# relies on r-forge.r-project.org being accessible
read_rforge("rgl/R/axes.R", project = "rgl")
read_rforge("rgl/R/axes.R", project = "rgl", extra = "&revision=519")
```

render_html *Set or get output hooks for different output formats*

Description

The `render_*()` functions set built-in output hooks for LaTeX, HTML, Markdown, reStructuredText, AsciiDoc, and Textile. The `hooks_*()` functions return a list of the output hooks for the corresponding format.

Usage

```
render_html()

hooks_html()

render_asciidoc()

hooks_asciidoc()

render_latex()

hooks_latex()

render_sweave()

hooks_sweave(envirs = c("Sinput", "Soutput", "Schunk"))

render_listings()

hooks_listings(envirs = c("Sinput", "Soutput", "Schunk"))

render_markdown(strict = FALSE, fence_char = "`")

hooks_markdown(strict = FALSE, fence_char = "`")

render_jekyll(highlight = c("pygments", "prettify", "none"), extra = "")

hooks_jekyll(highlight = c("pygments", "prettify", "none"), extra = "")

render_rst(strict = FALSE)

hooks_rst(strict = FALSE)

render_textile()

hooks_textile()
```

Arguments

envirs	Names of LaTeX environments for code input, output, and chunk.
strict	Boolean; whether to use strict markdown or reST syntax. For markdown, if TRUE, code blocks will be indented by 4 spaces, otherwise they are put in fences made by three backticks. For reST, if TRUE, code is put under two colons and indented by 4 spaces, otherwise it is put under the 'sourcecode' directive (this is useful for e.g. Sphinx).
fence_char	A single character to be used in the code blocks fence. This can be e.g. a backtick or a tilde, depending on your Markdown rendering engine.

highlight	Which code highlighting engine to use: if pygments, the Liquid syntax is used (default approach Jekyll); if prettify, the output is prepared for the JavaScript library 'prettify.js'; if none, no highlighting engine will be used, and code blocks are simply indented by 4 spaces).
extra	Extra tags for the highlighting engine. For pygments, this can be 'linenos'; for prettify, it can be 'linenums'.

Details

There are three variants of Markdown documents: ordinary Markdown (render_markdown(strict = TRUE), which calls hooks_markdown(strict = TRUE)), extended Markdown (e.g., GitHub Flavored Markdown and Pandoc; render_markdown(strict = FALSE), which calls hooks_markdown(strict = FALSE)), and Jekyll (a blogging system on GitHub; render_jekyll(), which calls hooks_jekyll()).

For LaTeX output, there are three variants: **knitr**'s default style (render_latex(), which calls hooks_latex() and uses the LaTeX **framed** package), Sweave style (render_sweave(), which calls hooks_sweave() and uses 'Sweave.sty'), and listings style (render_listings(), which calls hooks_listings() and uses LaTeX **listings** package).

Default HTML output hooks are set by render_html() (which calls hooks_html()); render_rst() (which calls hooks_rst()) is for reStructuredText; render_textile() (which calls hooks_textile()) is for Textile, and render_asciidoc() (which calls hooks_asciidoc()) is AsciiDoc.

The render_*() functions can be used before knit() or in the first chunk of the input document (ideally this chunk has options include = FALSE and cache = FALSE) so that all the following chunks will be formatted as expected.

You can also use knit_hooks to set the format's hooks with the hooks_*() functions; see references for more info on further customizing output hooks.

Value

NULL for render_* functions; corresponding hooks are set as a side effect. A list of output hooks for hooks_*() functions.

References

See output hooks in https://yihui.org/knitr/hooks/, and some examples in https://bookdown.org/yihui/rmarkdown-cookbook/output-hooks.html

Jekyll and Liquid: https://github.com/jekyll/jekyll/wiki/Liquid-Extensions; prettify.js: https://code.google.com/archive/p/google-code-prettify

Examples

```
# below is pretty much what knitr::render_markdown() does:
knitr::knit_hooks$set(knitr::hooks_markdown())

# you can retrieve a subset of the hooks and set them, e.g.,
knitr::knit_hooks$set(knitr::hooks_markdown()["source"])

knitr::knit_hooks$restore()
```

rnw2pdf *Convert an 'Rnw' document to PDF*

Description

Call `knit()` to compile the '.Rnw' input to '.tex', and then `tinytex::latexmk()` to convert '.tex' to '.pdf'.

Usage

```
rnw2pdf(
  input,
  output = with_ext(input, "pdf"),
  compiler = "xelatex",
  envir = parent.frame(),
  quiet = FALSE,
  clean = TRUE,
  error = FALSE,
  ...
)
```

Arguments

`input`	Path to the input file.
`output`	Path of the PDF output file. By default, it uses the same name as the `input`, but changes the file extension to ".pdf".
`compiler, ...`	The LaTeX engine and other arguments to be passed to `tinytex::latexmk()`. The default compiler is `xelatex`.
`envir`	Environment in which code chunks are to be evaluated, for example, `parent.frame()`, `new.env()`, or `globalenv()`).
`quiet`	Boolean; suppress the progress bar and messages?
`clean`	If `TRUE`, the intermediate files will be removed.
`error`	If `FALSE`, knitting stops when any error occurs.

Details

This function is similar to `knit2pdf()`, with the following differences:

1. The default compiler is "xelatex" instead of "pdflatex".
2. `output` uses the file extension ".pdf" instead of ".tex".
3. Before knitting, it tries to remove the `output` file and will throw a clear error if the file cannot be removed.
4. `output` could be under any dir, not necessarily the same directory as `input`.
5. It cleans up intermediate files by default, including the ".tex" file.
6. It stops knitting when any error occurs (by setting the chunk option `error = FALSE`).

Value

The `output` file path.

`rocco` *Knit R Markdown using the classic Docco style*

Description

The classic Docco style is a two-column layout, with text in the left and code in the right column.

Usage

```
rocco(input, ...)
```

Arguments

`input`	Path of the input R Markdown file.
`...`	Arguments to be passed to `knit2html`

Details

The output HTML page supports resizing and hiding/showing the two columns. Move the cursor to the center of the page, and it will change to a bidirectional resize cursor; drag the cursor to resize the two columns. Press the key `t` to hide the code column (show the text column only), and press again to hide the text column (show code).

Value

An HTML file is written, and its name is returned.

Author(s)

Weicheng Zhu and Yihui Xie

References

The Docco package by Jeremy Ashkenas: https://github.com/jashkenas/docco

Examples

```
rocco_view = function(input) {
    owd = setwd(tempdir())
    on.exit(setwd(owd))
    if (!file.exists(input))
        return()
    o = rocco(input, header = "", quiet = TRUE)
    if (interactive())
        browseURL(o)
```

```
}
# knit these two vignettes using the docco style
rocco_view(system.file("doc", "docco-classic.Rmd", package = "knitr"))
rocco_view(system.file("doc", "knit_expand.Rmd", package = "knitr"))
```

`rst2pdf` *A wrapper for rst2pdf*

Description

Convert reST to PDF using `rst2pdf` (which converts from rst to PDF using the ReportLab open-source library).

Usage

```
rst2pdf(input, command = "rst2pdf", options = "")
```

Arguments

`input`	The input rst file.
`command`	Character string giving the path of the `rst2pdf` program. If the program is not in your PATH, the full path has to be given here.
`options`	Extra command line options, e.g. `'-v'`.

Value

An input file '`*.rst`' will produce '`*.pdf`' and this output filename is returned if the conversion was successful.

Author(s)

Alex Zvoleff and Yihui Xie

References

https://github.com/rst2pdf/rst2pdf

See Also

`knit2pdf`

set_alias *Set aliases for chunk options*

Description

We do not have to use the chunk option names given in **knitr**; we can set aliases for them. The aliases are a named character vector; the names are aliases and the elements in this vector are the real option names.

Usage

```
set_alias(...)
```

Arguments

...	Named arguments. Argument names are aliases, and argument values are real option names.

Value

NULL. `opts_knit$get('aliases')` is modified as the side effect.

Examples

```
set_alias(w = "fig.width", h = "fig.height")
# then we can use options w and h in chunk headers instead of fig.width and
# fig.height
```

set_header *Set the header information*

Description

Some output documents may need appropriate header information. For example, for LaTeX output, we need to write '`\usepackage{tikz}`' into the preamble if we use tikz graphics; this function sets the header information to be written into the output.

Usage

```
set_header(...)
```

Arguments

...	Header components; currently possible components are `highlight`, `tikz` and `framed`, which contain the necessary commands to be used in the HTML header or LaTeX preamble. Note that HTML output does not use the `tikz` and `framed` components, since they do not make sense in the context of HTML.

Details

By default, **knitr** will set up the header automatically. For example, if the tikz device is used, **knitr** will add '`\usepackage{tikz}`' to the LaTeX preamble, and this is done by setting the header component `tikz` to be a character string: `set_header(tikz = '\usepackage{tikz}')`. Similary, when we highlight R code using the **highlight** package (i.e. the chunk option `highlight = TRUE`), **knitr** will set the `highlight` component of the header vector automatically; if the output type is HTML, this component will be different – instead of LaTeX commands, it contains CSS definitions.

For power users, all the components can be modified to adapt to a customized type of output. For instance, we can change `highlight` to LaTeX definitions of the **listings** package (and modify the output hooks accordingly), so we can decorate R code using the **listings** package.

Value

The header vector in `opts_knit` is set.

Examples

```
set_header(tikz = "\\usepackage{tikz}")
opts_knit$get("header")
```

---

set_parent *Specify the parent document of child documents*

Description

This function extracts the LaTeX preamble of the parent document to use for the child document, so that the child document can be compiled as an individual document.

Usage

```
set_parent(parent)
```

Arguments

`parent`	Path to the parent document, relative to the current child document.

Details

When the preamble of the parent document also contains code chunks and inline R code, they will be evaluated as if they were in this child document. For examples, when **knitr** hooks or other options are set in the preamble of the parent document, it will apply to the child document as well.

Value

The preamble is extracted and stored to be used later when the complete output is written.

Note

Obviously this function is only useful when the output format is LaTeX. This function only works when the child document is compiled in a standalone mode using `knit()` (instead of being called in `knit_child()`); when the parent document is compiled, this function in the child document will be ignored.

References

https://yihui.org/knitr/demo/child/

Examples

```
## can use, e.g. \Sexpr{set_parent('parent_doc.Rnw')} or

# <<setup-child, include=FALSE>>=

# set_parent('parent_doc.Rnw')

# @
```

`sew` *Wrap evaluated results for output*

Description

This function is mainly for internal use: it is called on each part of the output of the code chunk (code, messages, text output, and plots, etc.) after all statements in the code chunk have been evaluated, and will sew these pieces of output together into a character vector.

Usage

```
sew(x, options = list(), ...)
```

Arguments

`x`	Output from `evaluate::evaluate()`.
`options`	A list of chunk options used to control output.
`...`	Other arguments to pass to methods.

spin *Spin goat's hair into wool*

Description

This function takes a specially formatted R script and converts it to a literate programming document. By default normal text (documentation) should be written after the roxygen comment (`#'`) and code chunk options are written after `#+` or `#-` or `# ----` or any of these combinations replacing # with `--`.

Usage

```
spin(
  hair,
  knit = TRUE,
  report = TRUE,
  text = NULL,
  envir = parent.frame(),
  format = c("Rmd", "Rnw", "Rhtml", "Rtex", "Rrst"),
  doc = "^#+'[ ]?",
  inline = "^[{][{](.+)[}][}][ ]*$",
  comment = c("^[# ]*/[*]", "^.*[*]/ *$"),
  precious = !knit && is.null(text)
)
```

Arguments

`hair`	Path to the R script. The script must be encoded in UTF-8 if it contains multibyte characters.
`knit`	Logical; whether to compile the document after conversion.
`report`	Logical; whether to generate a report for 'Rmd', 'Rnw' and 'Rtex' output. Ignored if `knit = FALSE`.
`text`	A character vector of code, as an alternative way to provide the R source. If `text` is not `NULL`, `hair` will be ignored.
`envir`	Environment for `knit()` to evaluate the code.
`format`	Character; the output format. The default is R Markdown.
`doc`	A regular expression to identify the documentation lines; by default it follows the roxygen convention, but it can be customized, e.g. if you want to use ## to denote documentation, you can use `'^##\\s*'`.
`inline`	A regular expression to identify inline R expressions; by default, code of the form `{{code}}` on its own line is treated as an inline expression.
`comment`	A pair of regular expressions for the start and end delimiters of comments; the lines between a start and an end delimiter will be ignored. By default, the delimiters are `/*` at the beginning of a line, and `*/` at the end, following the convention of C comments.

precious — logical: whether intermediate files (e.g., .Rmd files when format is "Rmd") should be preserved. The default is FALSE if knit is TRUE and the input is a file.

Details

Obviously the goat's hair is the original R script, and the wool is the literate programming document (ready to be knitted).

Value

If text is NULL, the path of the final output document, otherwise the content of the output.

Note

If the output format is Rnw and no document class is specified in roxygen comments, this function will automatically add the article class to the LaTeX document so that it is complete and can be compiled. You can always specify the document class and other LaTeX settings in roxygen comments manually.

When the output format is Rmd, it is compiled to HTML via knit2html(), which uses R Markdown v1 instead of v2. If you want to use the latter, you should call rmarkdown::render() instead.

Author(s)

Yihui Xie, with the original idea from Richard FitzJohn (who named it as sowsear() which meant to make a silk purse out of a sow's ear)

References

https://yihui.org/knitr/demo/stitch/

See Also

stitch (feed a template with an R script)

---

spin_child *Spin a child R script*

Description

This function is similar to knit_child() but is used in R scripts instead. When the main R script is not called via spin(), this function simply executes the child script via sys.source(), otherwise it calls spin() to spin the child script into a source document, and uses knit_child() to compile it. You can call this function in R code, or using the syntax of inline R expressions in spin() (e.g. {{knitr::spin_child('script.R')}}).

Usage

```
spin_child(input, format)
```

Arguments

input	Filename of the input R script.
format	Passed to format in spin(). If not provided, it will be guessed from the current knitting process.

Value

A character string of the knitted R script.

---

stitch *Automatically create a report based on an R script and a template*

Description

This is a convenience function for small-scale automatic reporting based on an R script and a template. The default template is an Rnw file (LaTeX); stitch_rhtml() and stitch_rmd() are wrappers on top of stitch() using the R HTML and R Markdown templates respectively.

Usage

```
stitch(
  script,
  template = system.file("misc", "knitr-template.Rnw", package = "knitr"),
  output = NULL,
  text = NULL,
  envir = parent.frame()
)

stitch_rhtml(..., envir = parent.frame())

stitch_rmd(..., envir = parent.frame())
```

Arguments

script	Path to the R script.
template	Path of the template to use. By default, the Rnw template in this package; there is also an HTML template in **knitr**.
output	Output filename, passed to knit). By default, the base filename of the script is used.
text	A character vector. This is an alternative way to provide the input file.
envir	Environment in which code chunks are to be evaluated, for example, parent.frame(), new.env(), or globalenv()).
...	Arguments passed to stitch().

Details

The first two lines of the R script can contain the title and author of the report in comments of the form '`## title:`' and '`## author:`'. The template must have a token '`%sCHUNK_LABEL_HERE`', which will be used to input all the R code from the script. See the examples below.

The R script may contain chunk headers of the form '`## ---- label,opt1=val1, opt2=val2`', which will be copied to the template; if no chunk headers are found, the whole R script will be inserted into the template as one code chunk.

Value

path of the output document

See Also

`spin` (turn a specially formatted R script to a report)

Examples

```
s = system.file("misc", "stitch-test.R", package = "knitr")
if (interactive()) stitch(s)  # compile to PDF

# HTML report
stitch(s, system.file("misc", "knitr-template.Rhtml", package = "knitr"))

# or convert markdown to HTML
stitch(s, system.file("misc", "knitr-template.Rmd", package = "knitr"))

unlink(c("stitch-test.html", "stitch-test.md", "figure"), recursive = TRUE)
```

`Sweave2knitr` *Convert Sweave to knitr documents*

Description

This function converts an Sweave document to a **knitr**-compatible document.

Usage

```
Sweave2knitr(
  file,
  output = gsub("[.]([^.]+)$", "-knitr.\\1", file),
  text = NULL
)
```

Arguments

file	Path to the Rnw file (must be encoded in UTF-8).
output	Output file path. By default, 'file.Rnw' produces 'file-knitr.Rnw'); if text is not NULL, no output file will be produced.
text	An alternative way to provide the Sweave code as a character string. If text is provided, file will be ignored.

Details

The pseudo command '\SweaveInput{file.Rnw}' is converted to a code chunk header <<child='file.Rnw'>>=.

Similarly '\SweaveOpts{opt = value}' is converted to a code chunk 'opts_chunk$set(opt = value)' with the chunk option include = FALSE; the options are automatically fixed in the same way as local chunk options (explained below).

The Sweave package '\usepackage{Sweave}' in the preamble is removed because it is not required.

Chunk options are updated if necessary: option values true and false are changed to TRUE and FALSE respectively; fig=TRUE is removed because it is not necessary for **knitr** (plots will be automatically generated); fig=FALSE is changed to fig.keep='none'; the devices pdf/jpeg/png/eps/tikz=TRUE are converted to dev='pdf'/'jpeg'/'png'/'postscript'/'tikz'; pdf/jpeg/png/eps/tikz=FALSE are removed; results=tex/verbatim/hide are changed to results='asis'/'markup'/'hide'; width/height are changed to fig.width/fig.height; prefix.string is changed to fig.path; print/term/prefix=TRUE/FALSE are removed; most of the character options (e.g. engine and out.width) are quoted; keep.source=TRUE/FALSE is changed to tidy=FALSE/TRUE (note the order of values).

If a line @ (it closes a chunk) directly follows a previous @, it is removed; if a line @ appears before a code chunk and no chunk is before it, it is also removed, because **knitr** only uses one '@' after '<<>>=' by default (which is not the original Noweb syntax but more natural).

Value

If text is NULL, the output file is written and NULL is returned. Otherwise, the converted text string is returned.

Note

If '\SweaveOpts{}' spans across multiple lines, it will not be fixed, and you have to fix it manually. The LaTeX-style syntax of Sweave chunks are ignored (see ?SweaveSyntaxLatex); only the Noweb syntax is supported.

References

The motivation of the changes in the syntax: https://yihui.org/knitr/demo/sweave/

See Also

Sweave, gsub

Examples

```
Sweave2knitr(text = "<<echo=TRUE>>=")  # this is valid
Sweave2knitr(text = "<<png=true>>=")  # dev='png'
Sweave2knitr(text = "<<eps=TRUE, pdf=FALSE, results=tex, width=5, prefix.string=foo>>=")
Sweave2knitr(text = "<<,png=false,fig=TRUE>>=")
Sweave2knitr(text = "\\SweaveOpts{echo=false}")
Sweave2knitr(text = "\\SweaveInput{hello.Rnw}")
# Sweave example in utils
testfile = system.file("Sweave", "Sweave-test-1.Rnw", package = "utils")
Sweave2knitr(testfile, output = "Sweave-test-knitr.Rnw")
if (interactive()) knit("Sweave-test-knitr.Rnw")  # or knit2pdf() directly
unlink("Sweave-test-knitr.Rnw")
```

---

vignette_engines *Package vignette engines*

Description

Since R 3.0.0, package vignettes can use non-Sweave engines, and **knitr** has provided a few engines to compile vignettes via `knit()` with different templates. See https://yihui.org/knitr/demo/vignette/ for more information.

Note

If you use the `knitr::rmarkdown` engine, please make sure that you put **rmarkdown** in the 'Suggests' field of your 'DESCRIPTION' file. Also make sure pandoc is available during `R CMD build`. If you build your package from RStudio, this is normally not a problem. If you build the package outside RStudio, run `rmarkdown::find_pandoc()` in an R session to check if Pandoc can be found.

When the **rmarkdown** package is not installed or not available, or pandoc cannot be found, the `knitr::rmarkdown` engine will fall back to the `knitr::knitr` engine, which uses R Markdown v1 based on the **markdown** package.

Examples

```
library(knitr)
vig_list = tools::vignetteEngine(package = "knitr")
str(vig_list)
vig_list[["knitr::knitr"]][c("weave", "tangle")]
vig_list[["knitr::knitr_notangle"]][c("weave", "tangle")]
vig_list[["knitr::docco_classic"]][c("weave", "tangle")]
```

`wrap_rmd` *Wrap long lines in Rmd files*

Description

This function wraps long paragraphs in an R Markdown file. Other elements are not wrapped: the YAML preamble, fenced code blocks, section headers and indented elements. The main reason for wrapping long lines is to make it easier to review differences in version control.

Usage

```
wrap_rmd(file, width = 80, text = NULL, backup)
```

Arguments

`file`	The input Rmd file.
`width`	The expected line width.
`text`	A character vector of text lines, as an alternative to `file`. If `text` is not `NULL`, `file` is ignored.
`backup`	Path to back up the original file in case anything goes wrong. If set to `NULL`, no backup is made. The default value is constructed from `file` by adding `__` before the base filename.

Value

If `file` is provided, it is overwritten; if `text` is provided, a character vector is returned.

Note

Currently it does not wrap blockquotes or lists (ordered or unordered). This feature may or may not be added in the future.

Examples

```
wrap_rmd(text = c("```", "1+1", "```", "- a list item", "> a quote", "",
    paste(rep("this is a normal paragraph", 5), collapse = " ")))
```

write_bib *Generate BibTeX bibliography databases for R packages*

Description

This function uses `utils::citation()` and `utils::toBibtex()` to create bib entries for R packages and write them in a file. It can facilitate the auto-generation of bibliography databases for R packages, and it is easy to regenerate all the citations after updating R packages.

Usage

```
write_bib(
  x = .packages(),
  file = "",
  tweak = TRUE,
  width = NULL,
  prefix = getOption("knitr.bib.prefix", "R-"),
  lib.loc = NULL
)
```

Arguments

`x`	Package names. Packages which are not installed are ignored.
`file`	The (‘`.bib`’) file to write. By default, or if `NULL`, output is written to the R console.
`tweak`	Whether to fix some known problems in the citations, especially non-standard format of author names.
`width`	Width of lines in bibliography entries. If `NULL`, lines will not be wrapped.
`prefix`	Prefix string for keys in BibTeX entries; by default, it is ‘`R-`’ unless `option('knitr.bib.prefix')` has been set to another string.
`lib.loc`	A vector of path names of R libraries.

Details

For a package, the keyword ‘`R-pkgname`’ is used for its bib item, where ‘`pkgname`’ is the name of the package. Citation entries specified in the ‘`CITATION`’ file of the package are also included. The main purpose of this function is to automate the generation of the package citation information because it often changes (e.g. author, year, package version, ...).

Value

A list containing the citations. Citations are also written to the `file` as a side effect.

Note

Some packages on CRAN do not have standard bib entries, which was once reported by Michael Friendly at https://stat.ethz.ch/pipermail/r-devel/2010-November/058977.html. I find this a real pain, and there are no easy solutions except contacting package authors to modify their DESCRIPTION files. Anyway, the argument `tweak` has provided ugly hacks to deal with packages which are known to be non-standard in terms of the format of citations; `tweak = TRUE` is by no means intended to hide or modify the original citation information. It is just due to the loose requirements on package authors for the DESCRIPTION file. On one hand, I apologize if it really mangles the information about certain packages; on the other, I strongly recommend package authors to consider the 'Authors@R' field (see the manual *Writing R Extensions*) to make it easier for other people to cite R packages. See `knitr::::tweak.bib` for details of tweaks. Also note this is subject to future changes since R packages are being updated. If you want to contribute more tweaks, please edit the file 'inst/misc/tweak_bib.csv' in the source package.

Author(s)

Yihui Xie and Michael Friendly

Examples

```
write_bib(c("RGtk2", "gWidgets"), file = "R-GUI-pkgs.bib")
unlink("R-GUI-pkgs.bib")

write_bib(c("animation", "rgl", "knitr", "ggplot2"))
write_bib(c("base", "parallel", "MASS"))  # base and parallel are identical
write_bib("cluster", prefix = "")  # a empty prefix
write_bib("digest", prefix = "R-pkg-")  # a new prefix
write_bib("digest", tweak = FALSE)  # original version

# what tweak=TRUE does
str(knitr:::.tweak.bib)
```

Index