Type Package
Title A General-Purpose Package for Dynamic Report Generation in R
Version 1.9
Date 2015-01-20
Maintainer Yihui Xie <xie@yihui.name>

Description This package provides a general-purpose tool for dynamic report generation in R, which can be used to deal with any type of (plain text) files, including Sweave, HTML, Markdown, reStructuredText, AsciiDoc, and Textile. R code is evaluated as if it were copied and pasted in an R terminal thanks to the evaluate package (e.g., we do not need to explicitly print() plots from ggplot2 or lattice). R code can be reformatted by the formatR package so that long lines are automatically wrapped, with indent and spaces added, and comments preserved. A simple caching mechanism is provided to cache results from computations for the first time and the computations will be skipped the next time. Almost all common graphics devices, including those in base R and add-on packages like Cairo, cairoDevice and tikzDevice, are built-in with this package and it is straightforward to switch between devices without writing any special functions. The width and height as well as alignment of plots in the output document can be specified in chunk options (the size of plots for graphics devices is also supported). Multiple plots can be recorded in a single code chunk, and it is also allowed to rearrange plots to the end of a chunk or just keep the last plot. Warnings, messages and errors are written in the output document by default (can be turned off). The large collection of hooks in this package makes it possible for the user to control almost everything in the R code input and output. Hooks can be used either to format the output or to run R code fragments before or after a code chunk. The language in code chunks is not restricted to R (there is simple support to Python and shell scripts, etc). Many features are borrowed from or inspired by Sweave, cacheSweave, pgfSweave, brew and decumar.

Depends R (>= 2.14.1)
Imports evaluate (>= 0.5.5),
    digest,
    formatR (>= 1.0),
highr (>= 0.4),
markdown (>= 0.7.4),
stringr (>= 0.6),
tools

**Suggests** testit,
rgl (>= 0.95.1201),
codetools,
rmarkdown,
XML,
RCurl

**License** GPL

**URL** http://yihui.name/knitr/

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

**VignetteBuilder** knitr

**Collate** 'block.R'
'cache.R'
'citation.R'
'util.R'
'plot.R'
'hooks-html.R'
'defaults.R'
'concordance.R'
'engine.R'
'themes.R'
'highlight.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'
'parser.R'
'pattern.R'
'rocco.R'
'spin.R'
'table.R'
'template.R'
'themes-eclipse.R'
'util-base64.R'
'util-conversion.R'
R topics documented:

- 'utils-rd2html.R'
- 'utils-sweave.R'
- 'utils-upload.R'
- 'utils-vignettes.R'
- 'zzz.R'

R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>knitr-package</td>
<td>4</td>
</tr>
<tr>
<td>all_labels</td>
<td>5</td>
</tr>
<tr>
<td>all_patterns</td>
<td>5</td>
</tr>
<tr>
<td>asis_output</td>
<td>7</td>
</tr>
<tr>
<td>clean_cache</td>
<td>7</td>
</tr>
<tr>
<td>current_input</td>
<td>8</td>
</tr>
<tr>
<td>dep_auto</td>
<td>8</td>
</tr>
<tr>
<td>dep_prev</td>
<td>9</td>
</tr>
<tr>
<td>eclipse_theme</td>
<td>10</td>
</tr>
<tr>
<td>engine_output</td>
<td>11</td>
</tr>
<tr>
<td>fig_chunk</td>
<td>11</td>
</tr>
<tr>
<td>fig_path</td>
<td>12</td>
</tr>
<tr>
<td>hook_ffmpeg_html</td>
<td>13</td>
</tr>
<tr>
<td>hook_movecode</td>
<td>14</td>
</tr>
<tr>
<td>hook_plot_asciidoc</td>
<td>15</td>
</tr>
<tr>
<td>hook_rgl</td>
<td>16</td>
</tr>
<tr>
<td>image_uri</td>
<td>18</td>
</tr>
<tr>
<td>imgur_upload</td>
<td>18</td>
</tr>
<tr>
<td>inline_expr</td>
<td>20</td>
</tr>
<tr>
<td>kable</td>
<td>20</td>
</tr>
<tr>
<td>knit</td>
<td>22</td>
</tr>
<tr>
<td>knit2html</td>
<td>24</td>
</tr>
<tr>
<td>knit2pdf</td>
<td>25</td>
</tr>
<tr>
<td>knit2wp</td>
<td>27</td>
</tr>
<tr>
<td>knit_child</td>
<td>28</td>
</tr>
<tr>
<td>knit_engines</td>
<td>29</td>
</tr>
<tr>
<td>knit_exit</td>
<td>30</td>
</tr>
<tr>
<td>knit_expand</td>
<td>31</td>
</tr>
<tr>
<td>knit_filter</td>
<td>32</td>
</tr>
<tr>
<td>knit_global</td>
<td>32</td>
</tr>
<tr>
<td>knit_hooks</td>
<td>33</td>
</tr>
<tr>
<td>knit_meta</td>
<td>33</td>
</tr>
<tr>
<td>knit_patterns</td>
<td>34</td>
</tr>
<tr>
<td>knit_print</td>
<td>35</td>
</tr>
<tr>
<td>knit_rd</td>
<td>36</td>
</tr>
<tr>
<td>knit_theme</td>
<td>37</td>
</tr>
<tr>
<td>load_cache</td>
<td>38</td>
</tr>
<tr>
<td>opts_chunk</td>
<td>39</td>
</tr>
<tr>
<td>opts_knit</td>
<td>41</td>
</tr>
<tr>
<td>opts_template</td>
<td>42</td>
</tr>
</tbody>
</table>
**Description**

This is an alternative tool to Sweave with more flexible design and new features like cache and fine 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 (neater than what?) 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 <http://yihui.name>

**References**


**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**

This function returns all chunk labels as a character vector.

**Usage**

`all_labels()`

**Value**

A character vector.

all_patterns

All built-in patterns

**Description**

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

**Usage**

`all_patterns`

**Format**

List of 8

`$ rnw`: List of 7

..$ chunk.begin: chr "\s*<<\(.*)>>\s*"  
..$ chunk.end: chr "\s*@\s*\([^+].+)\s*"  
..$ inline.code: chr "\\\\Sexpr\{([^}]*)\}"  
..$ inline.comment: chr "\s*.*"  
..$ ref.chunk: chr "\s*<<\(.*\)>\s*"  
..$ header.begin: chr "^\n[\%]+\s*\\\documentclass[^]++\}++\}"  
..$ document.begin: chr "\s*\begin{document}"  

$ brew: List of 1

..$ inline.code: chr "\s*{\%\=[0,1]}\s*{[\^%]+}\s*{[-]*%}"  

$ tex: List of 8

..$ chunk.begin: chr "\s*\begin{rcode}\s*\([^]*\)\s*"  
..$ chunk.end: chr "\s*\end{rcode}\s*"  
..$ chunk.code: chr "\s*\"  
..$ ref.chunk: chr "\s*<<\(.*\)>\s*"  
..$ header.begin: chr "\s*\documentclass[^]++\}++\}"  
..$ document.begin: chr "\s*\begin{document}"  
..$ inline.comment: chr "\s*\([^%]+\)\s*"  
..$ inline.code: chr "\s*\{\\\\{[^}]*\}\}"
all_patterns

Usage: http://yihui.name/knitr/patterns

See Also

knit_patterns

Examples

all_patterns$rnw
all_patterns$html

str(all_patterns)
Description

This is a convenience function that assigns the input object a class named `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

```r
asis_output(x, meta = NULL, cacheable = length(meta) == 0)
```

Arguments

- `x` an R object (typically a character string, or 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` a logical value indicating if this object is cacheable

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 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.

Examples

```r
# see ?knit_print
```

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  whether to remove the files
path   the cache path

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)).

current_input  Query the current input filename

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

Usage

current_input()

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"))
**dep_prev**

**Arguments**

- **path**: the 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**

[http://yihui.name/knitr/demo/cache/](http://yihui.name/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**

[http://yihui.name/knitr/demo/cache/](http://yihui.name/knitr/demo/cache/)

**See Also**

- `dep_auto`
eclipse_theme  Download and convert a theme from eclipsecolorthemes.org to CSS

Description
This function uses the XML package to parse the theme as an XML file, then converts to a CSS file using a brew template in the knitr package. The CSS file can be further parsed with knit_theme$get(), and the result will be ready for knit_theme$set() to set the highlighting theme.

Usage
eclipse_theme(id)

Arguments
id id of theme to save as CSS

Value
Path to the CSS file converted from the website.

Author(s)
Ramnath Vaidyanathan and Yihui Xie

References
http://www.eclipsecolorthemes.org/

See Also
knit_theme

Examples
# http://www.eclipsecolorthemes.org/?view=theme&id=1
library(knitr)

opts_knit_set(out.format = "latex")
(css = eclipse_theme(1))
(thm = knit_theme$get(css)
knit_theme$set(thm)
**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**

```r
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`: the source code of the chunk, to which the output hook `source` is applied, unless the chunk option `echo` is `FALSE`
- `out`: the 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

**Value**

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

---

**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**

```r
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`: the chunk option `fig.path`
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

```r
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

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

Arguments

- `suffix`: a suffix of the filename; if it is not empty, nor does it 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 (by default the internal chunk option `fig.cur` if 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**

```r
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_scianimator()` uses the JavaScript library SciAnimator to create animations; `hook_r2swf()` uses the **R2SWF** package.

**Usage**

```r
hook_ffmpeg_html(x, options)
hook_scianimator(x, options)
hook_r2swf(x, options)
```

**Arguments**

- `x` the plot filename (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)`.
**Description**

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

**Usage**

```r
hook_movecode(x)
```

**Arguments**

- `x`: a character string (the content of the whole document output)

**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**

[http://yihui.name/knitr/hooks](http://yihui.name/knitr/hooks)

**Examples**

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

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

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

**Usage**

- `hook_plot_asciidoc(x, options)`
- `hook_plot_html(x, options)`
- `hook_plot_tex(x, options)`
- `hook_plot_md(x, options)`
- `hook_plot_rst(x, options)`
- `hook_plot_textile(x, options)`

**Arguments**

- `x` the plot filename (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 `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 (code with plot filenames wrapped)

**References**

[http://yihui.name/knitr/hooks](http://yihui.name/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)))

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_rgl(before, options, envir)

hook_pdfcrop(before, options, envir)

hook_optipng(before, options, envir)

hook_plot_custom(before, options, envir)

hook_webgl(before, options, envir)

hook_purl(before, options, envir)

Arguments

before, options, envir

see references
The function `hook_rgl()` can be set as a hook in `knitr` to save plots produced by the `rgl` package. According to the chunk option `dev` (graphical device), plots can be saved to different formats (postscript: 'eps'; pdf: 'pdf'; other devices correspond to the default PNG format). The plot window will be adjusted according to chunk options `fig.width` and `fig.height`. Filenames are derived from chunk labels and the `fig.path` option.

The function `hook_webgl()` is a wrapper for the `writeWebGL()` function in the `rgl` package. It writes WebGL code to the output to reproduce the `rgl` scene in a browser.

The function `hook_pdfcrop()` can use the program `pdfcrop` to crop the extra white margin when the plot format is PDF to make better use of the space in the output document, otherwise we often have to struggle with `par` to set appropriate margins. Note `pdfcrop` often comes with a LaTeX distribution such as MiKTeX or TeXLive, 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). Similarly, when the plot format is not PDF (e.g. PNG), the program `convert` in ImageMagick is used to trim the white margins (call `convert input Mtrim output`).

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'`. See [http://optipng.sourceforge.net/] for details.

When the plots are not recordable via `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 `knitr`. 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 hook chunks will be executed, hence `hook_purl()` will not work, either. To solve this problem, we need `cache = 2` instead of TRUE (see [http://yihui.name/knitr/demo/cache/] for the meaning of `cache = 2`).

**References**

[http://yihui.name/knitr/hooks#chunk_hooks]

**See Also**

`rgl.snapshot`, `rgl.postscript`

**Examples**

```r
knit_hooks$set(rgl = hook_rgl)
# then in code chunks, use the option rgl=TRUE
```
image_uri

Encode an image file to a data URI

Description
This function takes an image file and uses either the markdown package, or RCurl or the built-in function to encode it as a base64 string, which can be used in the img tag in HTML.

Usage
image_uri(f)

Arguments
f the path to the image file

Value
a character string (the data URI)

Author(s)
Wush Wu and Yihui Xie

References
http://en.wikipedia.org/wiki/Data_URI_scheme

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

imgur_upload

Upload an image to imgur.com

Description
This function uses the RCurl package to upload a image to imgur.com, and parses the XML response to a list with XML which contains information about the image in the Imgur website.

Usage
imgur_upload(file, key = "9f3460e67f308f6")
Arguments

`file` the path to the image file to be uploaded
`key` the client id for Imgur (by default 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


Examples

```r
## 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)
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

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

Arguments

code a character string of the inline R source code

syntax a character string to specify the syntax, e.g. rnw, html, or md, etc; if not specified, it will be guessed from the knitting context

Value

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

Examples

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

kable

Create tables in LaTeX, HTML, Markdown and reStructuredText

Description

This is a very simple table generator. It is simple by design. It is not intended to replace any other R packages for making tables.

Usage

```r
kable(x, format, digits = getOption("digits"), row.names = NA, col.names = colnames(x),
       align, caption = NULL, escape = TRUE, ...)
```
**Arguments**

- **x**: an R object (typically a matrix or data frame)
- **format**: a character string; possible values are `latex`, `html`, `markdown`, `pandoc`, and `rst`; this will be automatically determined if the function is called within `knitr`; it can also be set in the global option `knitr.table.format`
- **digits**: the maximum number of digits for numeric columns (passed to `round()`); it can also be a vector of length `ncol(x)` to set the number of digits for individual columns
- **row.names**: a logical value indicating 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**: the alignment of columns: a character vector consisting of 'l' (left), 'c' (center) and/or 'r' (right); by default, numeric columns are right-aligned, and other columns are left-aligned; if `align = NULL`, the default alignment is used
- **caption**: the table caption
- **escape**: escape special characters when producing HTML or LaTeX tables
- **...**: other arguments (see examples)

**Value**

A character vector of the table source code.

**Note**

The tables for `format = 'markdown'` also work for Pandoc when the `pipe_tables` extension is enabled (this is the default behavior for Pandoc >= 1.10).

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://github.com/yihui/knitr-examples/blob/master/091-knitr-table.Rnw](https://github.com/yihui/knitr-examples/blob/master/091-knitr-table.Rnw) for some examples in LaTeX, but they also apply to other document formats.

**See Also**

Other R packages such as `xtable` and `tables` for HTML and LaTeX tables, and `asci2` and `pander` for different flavors of markdown output and some advanced features and table styles.

**Examples**

```r
kable(head(iris), format = "latex")
kable(head(iris), format = "html")
kable(head(iris), format = "latex", caption = "Title of the table")
kable(head(iris), format = "html", caption = "Title of the table")
# use the booktabs package
```
```r
kable(mtcars, format = "latex", booktabs = TRUE)
# use the longtable package
kable(matrix(1:100, ncol = 5), format = "latex", digits = 2, longtable = TRUE)
# add some table attributes
kable(head(iris), format = "html", table.attr = "id="mytable"")
# reST output
kable(head(mtcars), format = "rst")
# no row names
kable(head(mtcars), format = "rst", row.names = FALSE)
# R Markdown/Github Markdown tables
kable(head(mtcars[, 1:5]), format = "markdown")
# no inner padding
kable(head(mtcars), format = "markdown", padding = 0)
# more padding
kable(head(mtcars), format = "markdown", padding = 2)
# Pandoc tables
kable(head(mtcars), format = "pandoc", caption = "Title of the table")
# save the value
x = kable(mtcars, format = "html")
cat(x, sep = "\n")
# can also set options(knitr.table.format = 'html') so that the output is HTML

---

**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)).

**Usage**

```r
knit(input, output = NULL, tangle = FALSE, text = NULL, quiet = FALSE,
    envir = parent.frame(), encoding = getOption("encoding"))

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

**Arguments**

- **input**: path of the input file
- **output**: path of the output file for knit(); if NULL, this function will try to guess and it
  will be under the current working directory
- **tangle**: whether to tangle the R code from the input file (like Stangle)
- **text**: a character vector as an alternative way to provide the input file
- **quiet**: whether to suppress the progress bar and messages
- **envir**: the environment in which the code chunks are to be evaluated (for example,
  parent.frame(), new.env(), or globalenv())
encoding the encoding of the input file; see file
... arguments passed to knit() from purl()
documentation an integer specifying the level of documentation to go the tangled script: 0 means pure code (discard all text chunks); 1 (default) means add the chunk headers to code; 2 means 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 a 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$rnw, ‘tex’ uses the list apat$latex, ‘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 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).

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 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.
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 \texttt{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 \texttt{opts_knit$set(root.dir = \ldots)} so all paths in code chunks are relative to this root.dir.

The arguments \texttt{input} and \texttt{output} do not have to be restricted to files; they can be \texttt{stdin}/\texttt{stdout} or other types of connections, but the pattern list to read the input has to be set in advance (see \texttt{pat_rnw}), and the output hooks should also be set (see \texttt{render_latex}), otherwise \texttt{knitr} will try to guess the patterns and output format.

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, you are 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 do \texttt{setwd(‘out/’); knit(‘../foo.Rmd’).} instead of \texttt{knit(‘foo.Rmd’, ‘out/foo.md’).}

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

References

Package homepage: \url{http://yihui.name/knitr/}. The \texttt{knitr} main manual: and graphics manual. See \texttt{citation(‘knitr’)} for the citation information.

Examples

\begin{verbatim}
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
\end{verbatim}

---

\texttt{knit2html} \textit{Convert markdown to HTML using \texttt{knit()} and \texttt{markdownToHTML()} \texttt{}}

\textbf{Description}

This is a convenience function to knit the input markdown source and call \texttt{markdownToHTML()} in the \texttt{markdown} package to convert the result to HTML.


**knit2pdf**

Usage

\[
\text{knit2pdf(input, output = NULL, ..., envir = parent.frame(), text = NULL, quiet = FALSE, encoding = getOption("encoding"))}
\]

Arguments

- `input` path of the input file
- `output` path of the output file for `knit()`; if NULL, this function will try to guess and it will be under the current working directory
- `...` options passed to `markdownToHTML`
- `envir` the environment in which the code chunks are to be evaluated (for example, `parent.frame()`, `new.env()`, or `globalenv()`)
- `text` a character vector as an alternative way to provide the input file
- `quiet` whether to suppress the progress bar and messages
- `encoding` the encoding of the input file; see `file`

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.

See Also

`knit`, `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")

---

**knit2pdf**

Convert Rnw or Rrst files to PDF using `knit()` and `texi2pdf()` or `rst2pdf()`

Description

Knit the input Rnw or Rrst document, and compile to PDF using `texi2pdf` or `rst2pdf`.

Usage

\[
\text{knit2pdf(input, output = NULL, compiler = NULL, envir = parent.frame(), quiet = FALSE, encoding = getOption("encoding"), \ldots)}
\]
Arguments

input path of the input file
output path of the output file for knit(); if NULL, this function will try to guess and it will be under the current working directory
compiler a character string which gives the LaTeX program used to compile the tex document to PDF (by default it uses the default setting of texi2pdf, which is often PDFLaTeX); this argument will be used to temporarily set the environmental variable 'pdflatex'. For an Rrst file, setting compiler to 'rst2pdf' will use rst2pdf to compiles the rst file to PDF using the ReportLab open-source library.
envir the environment in which the code chunks are to be evaluated (for example, parent.frame(), new.env(), or globalenv())
quiet whether to suppress the progress bar and messages
encoding the encoding of the input file; see file
... options to be passed to texi2pdf 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

See Also

knit, texi2pdf, rst2pdf

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.

**Usage**

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

**Arguments**

- **input**: the filename of the Rmd document
- **title**: the post title
- **...**: other meta information of the post, e.g. categories = c('R', 'Stats') and mt_keywords = c('knitr', 'wordpress'), etc
- **shortcode**: a logical vector of length 2: whether to use the shortcode `[[source code 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**: 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
- **encoding**: the encoding of the input file; see `file`
- **publish**: whether to 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**

[http://yihui.name/knitr/demo/wordpress/](http://yihui.name/knitr/demo/wordpress/)

**Examples**

```r
# 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

```r
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
  (ignored if not a list); when we use the child option in a parent chunk, the
  chunk options of the parent chunk will be passed to the options argument here
- `envir` the environment in which the 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 relative to the parent document.

References

http://yihui.name/knitr/demo/child/

Examples

```r
# you can write `$expr(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 `$expr(knit_child('child-doc.Rnw', eval =
# FALSE))`
```
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

List of 4
$ get :function (name, default = FALSE, drop = TRUE)
$ set :function (...)
$ merge :function (values)
$ restore:function (target = value)

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'.

Below is a list of built-in language engines, retrieved via knit_engines$get():

List of 26
$ awk :function (options)
$ bash :function (options)
$ coffee :function (options)
$ gawk :function (options)
$ groovy :function (options)
$ haskell :function (options)
$ node :function (options)
$ perl :function (options)
$ python :function (options)
$ Rscript :function (options)
$ ruby :function (options)
$ sas :function (options)
$ scala :function (options)
$ sed :function (options)
$ sh :function (options)
$ zsh :function (options)
$ highlight:function (options)
$ Rcpp :function (options)
$ tikz :function (options)
$ dot :function (options)
$ c :function (options)
$ fortran :function (options)
$ asy :function (options)
$ cat :function (options)
$ asis :function (options)
$ stan :function (options)

References

Usage: http://yihui.name/knitr/objects; examples: http://yihui.name/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)

Arguments

append a character vector to be appended to the results from knit() so far; by default, it 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

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.
knit_expand

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 = readLines(file, warn = FALSE), delim = c("{", "}"))

Arguments

file the template file
... a list of variables to be used for the code in the template; note the variables will be searched in the parent frame as well
text an alternative way to file to specify the template code directly (if provided, file will be ignored)
delim the (opening and ending) 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 (http://pyexpander.sourceforge.net) 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**

```r
knit_filter(ifile, encoding = "unknown")
```

**Arguments**

- `ifile` the filename of the source document
- `encoding` the file encoding

**Value**

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

**Examples**

```r
library(knitr)
kntir_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**

```r
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

List of 4

$ get :function (name, default = FALSE, drop = TRUE)

$ set :function (...)

$ merge :function (values)

$ restore:function (target = value)

References

Usage: [http://yihui.name/knitr/objects](http://yihui.name/knitr/objects)

Components in `knit_hooks`: [http://yihui.name/knitr/hooks](http://yihui.name/knitr/hooks)

Examples

```r
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.

Usage

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

**knit_patterns**

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

**Description**

Patterns are regular expressions and will be used in functions like `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**

List of 4

- `$get`: function (name, default = FALSE, drop = TRUE)
- `$set`: function (...)
- `$merge`: function (values)
- `$restore`: function (target = value)

**References**

Usage: [http://yihui.name/knitr/objects](http://yihui.name/knitr/objects)

Components in `knit_patterns`: [http://yihui.name/knitr/patterns](http://yihui.name/knitr/patterns)

**See Also**

- `all_patterns`

**Examples**

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

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


```r
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**

```r
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 References)

**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

```r
library(knitr)

# write tables for data frames
knit_print.data.frame = function(x, ...) {
  res = paste(c('',''), kable(x, output = FALSE), collapse = '
')
  asis_output(res)
}

# after you defined the above method, data frames will be printed as tables in
# knitr, which is different with the default print() behavior
```

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

```r
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 `Rd2HTML`
- `frame` whether to put a navigation frame on 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 DESCRIPTION file).
Examples

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

knit_rd("maps")
knot_rd("rpart")
setwd(system.file("html", package = "ggplot2"))
knot_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

List of 2
- `$ set: function (theme)`
- `$ get: function (theme = NULL)`

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 [http://rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)).

Author(s)

Ramnath Vaidyanathan and Yihui Xie
References


See Also

eclipse_theme (use Eclipse themes)

Examples

```r
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
```

---

**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

```r
load_cache(label, object, notfound = "NOT AVAILABLE",
           path = opts_chunk$get("cache.path"). 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 missing, NULL is returned)
- **notfound**: a value to use when the object cannot be found
- **path**: the path of the cache database (normally set in the global chunk option ```cache.path``)
- **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.

data.frame

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

List of 4
$ get :function (name, default = FALSE, drop = TRUE)
$ set :function (...)
$ merge :function (values)
$ restore: function (target = value)

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.

Below is a list of default chunk options, retrieved via opts_chunk$get():
List of 52
$ engine : chr "R"
$ split : logi FALSE
$ include : logi TRUE
$ purl : logi TRUE

References

Usage: http://yihui.name/knitr/objects
A list of available options: http://yihui.name/knitr/options#chunk_options

Examples

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

---

**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**

```r
opts_knit
```

**Format**

```
List of 4
$ get :function (name, default = FALSE, drop = TRUE)
$ set :function (...)
$ merge :function (values)
$ restore :function (target = value)
```

**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`.

Below is a list of default package options, retrieved via `opts_knit$get()`:

```
List of 24
$ progress : logi TRUE
$ verbose : logi FALSE
$ width : int 75
```
**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.

References

Usage: [http://yihui.name/knitr/options](http://yihui.name/knitr/options)

A list of available options: [http://yihui.name/knitr/options#package_options](http://yihui.name/knitr/options#package_options)

Examples

```r
opts_knit$set("verbose")
opts_knit$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
}
```

```r
```
Usage

 opts_template

Format

 List of 4
 $ get function (name, default = FALSE, drop = TRUE)
 $ set function (...) 
 $ merge function (values)
 $ restore function (target = value)

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 Markdown documents to other formats

Description

This function calls Pandoc to convert Markdown 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, encoding = getOption("encoding"))

Arguments

 input a character vector of the Markdown filenames
 format the output format (see References); it 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 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 the filename extensions; by default, the extension is inferred from the format, e.g. latex creates pdf, and dzslides creates html, and so on
 encoding the encoding of the input file; see file
Details

There are two ways to input the Pandoc configurations – through a config file, or embed the configurations in the markdown 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: http://johnmacfarlane.net/pandoc/; Examples and rules of the configurations: http://yihui.name/knitr/demo/pandoc

Also see R Markdown (v2) at http://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

---

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.
plot_crop

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 command pdfcrop x x is executed on a PDF plot file, and convert x trim x is executed for other types of plot files, where x is the plot filename.

Usage
plot_crop(x)

Arguments
x the plot filename
Details

The utility `pdfcrop` is often shipped with a LaTeX distribution, and `convert` is a command in ImageMagick (Windows users may have to put the bin path of ImageMagick into the `PATH` variable).

Value

The original filename.

References


---

**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`

Format

```r
language {  if (exists(".Random.seed", envir = globalenv())); get(".Random.seed", envir = globalenv()); }
```

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

http://yihui.name/knitr/demo/cache/

Examples

```r
eval(rand_seed)
norm(1) # .Random.seed is created (or modified)
eval(rand_seed)
```
**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**

```r
read_chunk(path, lines = readLines(path, warn = FALSE), labels = NULL, from = NULL,
            to = NULL, from.offset = 0L, to.offset = 0L)
read_demo(topic, package = NULL, ...)
```

**Arguments**

- `path`: the path to the R script
- `lines`: a character vector of the code lines (by default read from `path`)
- `labels`: a character vector of chunk labels (default `NULL`)
- `from`, `to`: a numeric vector specifying the starting/ending line numbers of code chunks, or a character vector; see Details
- `from.offset`, `to.offset`: an offset to be added to `from/to`
- `topic`, `package`: name of the demo and the package see `demo`
- `...`: arguments to be 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.
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

http://yihui.name/knitr/demo/externalization/

Examples

```r
## 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

```r
library(knitr)

read_rforge("rgl/R/axes.R", project = "rgl")
read_rforge("rgl/R/axes.R", project = "rgl", extra = \'&revision=519\')
```

render_asciidoc  
Set output hooks for different output formats

**Description**

These functions set built-in output hooks for LaTeX, HTML, Markdown, reStructuredText, Asciidoc and Textile.

Usage

```r
render_asciidoc()

render_html()

render_latex()

render_sweave()

render_listings()

render_markdown(strict = FALSE)

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

render_rst(strict = FALSE)
```
render_textile()

Arguments

strict 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 is put under the `sourcecode` directive (e.g. it is useful for Sphinx)

highlight which code highlighting engine to use: for pygments, the Liquid syntax is used (default approach Jekyll); for prettify, the output is prepared for the JavaScript library `prettify.js`: for none, no highlighting engine will be used (code blocks are indented by 4 spaces)

extra extra tags for the highlighting engine; for pygments, it can be 'linenos'; for prettify, it can be 'linenums'

Details

There are three variants of markdown documents: ordinary markdown (render_markdown(strict = TRUE)), extended markdown (e.g. GitHub Flavored Markdown and pandoc; render_markdown(strict = FALSE)), and Jekyll (a blogging system on GitHub; render_jekyll()). For LaTeX output, there are three variants as well: knitr's default style (render_latex(); use the LaTeX framed package), Sweave style (render_sweave(); use `Sweave.sty`) and listings style (render_listings(); use LaTeX listings package). Default HTML output hooks are set by render_html(); render_rst() and render_asciidoc() are for reStructuredText and AsciiDoc respectively.

These 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 use knit_hooks to further customize output hooks; see references.

Value

NULL; corresponding hooks are set as a side effect

References

See output hooks in http://yihui.name/knitr/hooks.

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) {
  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**: a character string which gives the path of the rst2pdf program (if it is not in PATH, the full path has to be given)
- **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


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 is a named character vector; the names are aliases and the elements in this vector are the real option names.

Usage

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

Arguments

... named arguments (argument names are aliases, and argument values are real chunk options)

Value

NULL (**opts_knit$<!-- get -->**) is modified as the side effect

Examples

```r
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

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

Arguments

... the 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 HTML output does not use the tikz and framed components (they do not make sense to HTML)
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}')`. Similarly, 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**

```r
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**

```r
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

http://yihui.name/knitr/demo/child/

Examples

```r
## can use, e.g. \Expr{set_parent('parent_doc.Rnw')}
#
## <<setup-child, include=FALSE>>=
## set_parent('parent_doc.Rnw')
## @
```

---

spin

**Spin goat’s hair into wool**

Description

This function takes a specially formatted R script and converts it to a literate programming docu-
ment. By default normal text (documentation) should be written after the roxygen comment (#')
and code chunk options are written after #+ or #- or # ---.

Usage

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

Arguments

- **hair**: the path to the R script
- **knit**: logical: whether to compile the document after conversion
- **report**: logical: whether to generate report for ‘Rmd’, ‘Rnw’ and ‘Rtex’ output (ignored
  if knit = FALSE)
- **text**: a character vector as an alternative way to hair to provide the R source; if text
  is not NULL, hair will be ignored
- **envir**: the environment for knit() to evaluate the code
format character: the output format (it takes five possible values); 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 `/` in the beginning and `*` in the end of a line (following the convention of C comments)

precious logical: whether intermediate files (e.g., .Rmd files when format is "Rmd") should be removed; default TRUE if knit == TRUE and 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.

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

http://yihui.name/knitr/demo/stitch/

See Also

- `stitch` (feed a template with an R script)

Examples

```
# write normal text like this and chunk options like below

# + label, opt=value
```
spin_child

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` the filename of the input R script
- `format` to be 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

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

stitch_rhtml(...)

stitch_rmd(...)
```

### 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`  
  the output filename (passed to knitr); by default it uses the base filename of the script

- `text`  
  a character vector as an alternative way to provide the input file

- `envir`  
  the environment in which the 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

```r
s = system.file("misc", "stitch-test.R", package = "knitr")

stitch(s)
```

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

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

---

**Sweave2knitr**

Convert Sweave to knitr documents

**Description**

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

**Usage**

```r
Sweave2knitr(file, output = gsub("\.[.]\([.]\)+", "~knitr.\1", file),
            encoding = getOption("encoding"), text = NULL)
```

**Arguments**

- `file`: the filename of the Rnw file
- `output`: the output filename (by default `file.Rnw` produces `file-knitr.Rnw`); if text is not NULL, no output file will be produced
- `encoding`: the encoding of the Rnw file
- `text`: an alternative way to provide the Sweave code as a character string (if provided, the file will be ignored)

**Details**

The pseudo command `\SweaveInput{file.Rnw}` is converted to a code chunk header ```\begin{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/jpg/png/eps/tikz=TRUE are converted to dev='pdf'/'jpg'/'png'/'postscript'/'tikz'; pdf/jpg/png/eps/tikz=FALSE are removed; results=txt/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: http://yihui.name/knitr/demo/swave/

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")
knit("Sweave-test-knitr.Rnw") # or knit2pdf() directly

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 http://yihui.name/knitr/demo/vignette/ for more information.
Examples

```r
library(knitr)

vig_list = if (getRversion() > "3.0.0") 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**

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

**Arguments**

- `file`: the input Rmd file
- `width`: the expected line width
- `text`: an alternative to `file` to input the text lines
- `backup`: the path to back up the original file (in case anything goes wrong); if `NULL`, it is ignored; by default it 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**

```r
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 citation and toBibtex to create bib entries for R packages and write them in a file. Only the auto-generated citations are included for a package. This function 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, prefix =getOption("knitr.bib.prefix", "R-"))

Arguments

x package names (packages which are not installed are ignored)
file the (\'\Nbib\') file to write (by default writes to the R console; ignored if it is NULL)
tweak whether to fix some known problems in the citations, especially non-standard format of authors
prefix a prefix string for keys in BibTeX entries; by default, it is \'R-\' unless option\(\text{\('knitr.bib.prefix\')}\) has been set to another string

Details

The citation is forced to be generated from the DESCRIPTION file of the package, the keyword \`pkpname\' is used for the bib item where \`pkpname\' is the name of the package.

Value

a list containing the citations (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 \url{https://stat.ethz.ch/pipermail/r-devel/2010-December/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\' field (see the manual Writing R Extensions) to make it easier for other people to cite R packages. See \texttt{knitr:::tweak.bib} for details of tweaks. Also note this is subject to future changes since R packages are being updated.
Author(s)

Yihui Xie and Michael Friendly

Examples

```r
write_bib(c("RGtk2", "gWidgets"), file = "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(c("rpart", "survival"))
write_bib(c("rpart", "survival"), tweak = FALSE)  # original version

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

*Topic datasets
  all_patterns, 5
  knit_engines, 29
  knit_hooks, 33
  knit_patterns, 34
  knit_theme, 37
  opts_chunk, 39
  opts_knit, 41
  opts_template, 42
  rand_seed, 46

all_labels, 5
all_patterns, 5, 34
asNcharacter, 7
asis_output, 7, 35
aspell, 32

citation, 62
clean_cache, 7
current_input, 8
demo, 47
dep_auto, 8, 9
dep_prev, 9, 9
eclipse_theme, 10, 38
gine_output, 11

fig_chunk, 11
fig_path, 12
file, 23, 25–27, 43
for, 21
globalenv, 22, 25, 26, 28, 58
grep, 34
gsub, 60

hook_ffmpeg_html, 13
hook_movecode, 14
hook_optipng (hook_rgl), 16
hook_pdfcrop (hook_rgl), 16

hook_plot_asciidoc, 15
hook_plot_custom, 15, 16
hook_plot_custom (hook_rgl), 16
hook_plot_html (hook_plot_asciidoc), 15
hook_plot_md (hook_plot_asciidoc), 15
hook_plot_rst (hook_plot_asciidoc), 15
hook_plot_tex (hook_plot_asciidoc), 15
hook_plot_textile (hook_plot_asciidoc), 15
hook_purl (hook_rgl), 16
hook_r2swf (hook_ffmpeg_html), 13
hook_rgl, 16
hook_scianimator (hook_ffmpeg_html), 13
hook_webgl (hook_rgl), 16

iconv, 27
image_uri, 18
imgur_upload, 18
inline_expr, 20

kable, 20, 35
knit, 4, 8, 9, 17, 19, 22, 23, 25, 26, 28, 30, 33, 55, 58, 60
knit2html, 24, 51
knit2pdf, 25, 52
knit2wp, 27
knit_child, 23, 28, 55, 57
knit_engines, 11, 29
knit_exit, 30
knit_expand, 31
knit_filter, 32
knit_global, 32
knit_hooks, 33, 50
knit_meta, 33
knit_patterns, 36, 30, 33, 34, 45
knit_print, 7, 35
knit_rd, 36
knit_rd_all (knit_rd), 36
knit_theme, 10, 37
knitr (knitr-package), 4
knitr-package, 4
lazyLoad, 38
load_cache, 38
ls, 33
markdownToHTML, 24, 25
new.env, 22, 25, 26, 28, 58
normal_print (knit_print), 35
option, 62
options, 41
opts_chunk, 17, 39
opts_current (opts_chunk), 39
opts_knit, 24, 41
opts_template, 42
pandoc, 43
par, 17
parent.frame, 22, 25, 26, 28, 58
pat_asciidoc (pat_rnw), 44
pat_brew (pat_rnw), 44
pat_html (pat_rnw), 44
pat_md (pat_rnw), 44
pat_rnw, 23, 24, 44
pat_tex (pat_rnw), 44
pat_textile (pat_rnw), 44
plot_crop, 45
purl, 17
purl (knit), 22
rand_seed, 46
Rd2HTML, 36
read.dcf, 44
read_chunk, 17, 47, 47
read_demo (read_chunk), 47
read_rforge, 48
recordPlot, 15, 17
render_asciidoc, 49
render_html (render_asciidoc), 49
render_jekyll (render_asciidoc), 49
render_latex, 23, 24
render_latex (render_asciidoc), 49
render_listings (render_asciidoc), 49
render_markdown (render_asciidoc), 49
render_rst (render_asciidoc), 49
render_sweave (render_asciidoc), 49
render_textile (render_asciidoc), 49
rgl.postscript, 17
rgl.snapshot, 17
rocco, 51
rst2pdf, 26, 52
set_alias, 53
set_header, 53
set_parent, 54
spin, 55, 57, 58
spin_child, 57
Stangle, 22
stitch, 56, 58
stitch_rhtml (stitch), 58
stitch_rmd (stitch), 58
Sweave, 60
Sweave2knitr, 4, 59
sys.source, 57
system2, 29
texi2pdf, 26
toBibTex, 62
vignette_engines, 60
wrap_rmd, 61
write_bib, 62
writeWebGL, 17