Package 'rebook'

October 16, 2025

Version 1.18.0

Date 2023-05-25

Title Re-using Content in Bioconductor Books

Description Provides utilities to re-use content across chapters of a Bioconductor book. This is mostly based on functionality developed while writing the OSCA book, but generalized for potential use in other large books with heavy compute. Also contains some functions to assist book deployment.

Imports utils, methods, knitr (>= 1.32), rmarkdown, CodeDepends, dir.expiry, filelock, BiocStyle

Suggests testthat, igraph, XML, BiocManager, RCurl, bookdown, rappdirs, yaml, BiocParallel, OSCA.intro, OSCA.workflows

License GPL-3

VignetteBuilder knitr

biocViews Software, Infrastructure, ReportWriting

RoxygenNote 7.2.3

git_url https://git.bioconductor.org/packages/rebook

git_branch RELEASE_3_21

git_last_commit 0f0347f

git_last_commit_date 2025-04-15

Repository Bioconductor 3.21

Date/Publication 2025-10-15

Author Aaron Lun [aut, cre, cph]

Maintainer Aaron Lun <infinite.monkeys.with.keyboards@gmail.com>

Contents

bioc-images 2
bookCache 3
buildChapterGraph 4
chapterPreamble 5

collapseStart 6
compileBook 7
compileChapter 8
configureBook 9
createMakefile 10
createRedirects 12
deployCustomCSS 13
extractCached 14
extractFromPackage 16
link 18
openingDetails 19
prettySessionInfo 20
rmd2id 21
scrapeDependencies 22
scrapeReferences 23
setupHTML 24
updateDependencies 25

Index **27**

`bioc-images` *Get various Bioconductor images*

Description

Helper functions to pull down images to use in the book. These aim to provide a sensible default for Bioconductor-related books.

Usage

```
BiocFavicon()

BiocSticker(mode = c("static", "animated"))
```

Arguments

`mode`	String specifying the type of sticker to show.

Value

`BiocFavicon` will return a path to a `favicon.ico` file.

`BiocSticker` will return a URL or path to a sticker.

Author(s)

Aaron Lun

Examples

```
BiocFavicon()

BiocSticker()
```

---

bookCache *Get the local book cache*

Description

Get the path to the cache directory in which the book will be built.

Usage

```
bookCache(package)

bookCacheExpiry()
```

Arguments

package	String containing the name of the book package.

Details

For bookCache, the last elements of the output path are the package and version, consistent with the expectations of **dir.expiry** functions. This path is located in a directory determined by **rappdirs**. If the environment variable REBOOK_CACHE is set, it is used to obtain the root of the path instead.

If the environment variable REBOOK_CACHE_EXPIRY is set, it is coerced into an integer and returned by bookCacheExpiry. This allows users to tune the expiry interval for older cached books.

Value

For bookCache, a string containing the path to the cache directory for this book package.

For bookCacheExpiry, an integer specifying the maximum number of days from last access for a book cache. Any unaccessed caches are subject to deletion by various **rebook** functions.

Author(s)

Aaron Lun

See Also

configureBook, where this function is used in the Makefile.

extractFromPackage, which populates the cache directory if this is not supplied.

Examples

```
bookCache('OSCA.workflows')
bookCacheExpiry()
```

`buildChapterGraph` *Build the chapter dependency graph*

Description

Build the dependency graph between chapter based on their `extractCached` calls to each other.

Usage

```
buildChapterGraph(dir, recursive = TRUE, pattern = "\\.Rmd$")
```

Arguments

`dir`	String containing the path to the directory containing Rmarkdown reports. This is searched recursively for all files ending in `".Rmd"`.
`recursive`, `pattern`	Further arguments to pass to `list.files` when searching for Rmarkdown reports.

Value

A directed graph object from the **igraph** package, where each node is a chapter and is connected to its dependencies by an edge.

Author(s)

Aaron Lun

Examples

```
dir <- tempfile()
dir.create(dir)

tmp1 <- file.path(dir, "alpha.Rmd")
write(file=tmp1, "```{r, echo=FALSE, results='asis'}
rebook::chapterPreamble()
```

```{r}
rodan <- 1
```")

tmp2 <- file.path(dir, "bravo.Rmd")
```
```

```
write(file=tmp2, "```{r, echo=FALSE, results='asis'}
rebook::chapterPreamble()
```

```{r}
extractCached('alpha.Rmd')
```")

# Building the chapter graph:
g <- buildChapterGraph(dir)
plot(g)
```

---

`chapterPreamble` *Execute chapter preamble code*

---

## Description

Execute code to set up the compilation environment at the start of every chapter.

## Usage

```
chapterPreamble(cache = TRUE)
```

Arguments

cache	Logical indicating whether to cache code chunks.

Details

Compilation is performed with no tolerance for errors, no printing of package start-up messages, and no printing of warnings.

Numbers are printed to 4 digits of precision.

The **BiocStyle** package is automatically attached, primarily for use of `Biocpkg` and similar functions.

HTML elements are defined using `setupHTML`.

Value

HTML is printed to standard output, see `setupHTML`.

Author(s)

Aaron Lun
```

Examples

```
tmp <- tempfile(fileext=".Rmd")
write(file=tmp, "```{r, echo=FALSE, results='asis'}
rebook::chapterPreamble()
```

```{r}
pi # four digits!
```

```{r}
warning('ASDASD') # warnings and messages are not saved in the HTML.
```

```{r, results='asis'}
prettySessionInfo()
```")

rmarkdown::render(tmp)

if (interactive()) browseURL(sub(".Rmd$", ".html", tmp))
```

---

# collapseStart *Print the collapse opening and ending*

---

## Description

Print HTML tags to open and close the collapsible chunks.

## Usage

```
collapseStart(message)

collapseEnd()
```

Arguments

message	String containing a message to insert in the collapsible header.

Value

Both functions will cat HTML tags; one to start and another to end each collapsible chunk.

Author(s)

Aaron Lun
```

Examples

```
collapseStart("This is collapsible")
cat("something inside the chunk\n")
collapseEnd()
```

`compileBook` *Compile the book*

Description

Copy a **bookdown** book to a separate workspace prior to compilation, and then copy the compiled book to a final location.

Usage

```
preCompileBook(src.dir, work.dir, desc = NULL)

postCompileBook(work.dir, final.dir, handle = NULL)
```

Arguments

`src.dir`	String containing the path to the book Rmarkdown sources.
`work.dir`	String containing the path to the versioned cache directory used to compile the book, see the **dir.expiry** package for details.
`desc`	String containing the path to a `DESCRIPTION` file to copy into `work.dir`. Typically used when the book is to inherit the `DESCRIPTION` of the enclosing package.
`final.dir`	String containing the path to the final location for the compiled book's HTMLs.
`handle`	The lock handle returned by `preCompileBook`.

Details

These two functions should bracket a `render_book` call. We do not make these into a single function as calling `render_book` inside another function inside a package does not interact properly with some imports. The offending example is that of `cbind`, which fails to be converted into an S4 generic (this would normally happen when **BiocGenerics** gets attached).

`preCompileBook` may take some time as it will compile all chapters via `compileChapter`. It does so by locking and unlocking each chapter as it is compiled, thus avoiding problems with concurrent attempts to compile the same chapter via `extractFromPackage`. (Concurrent compilation of different chapters is still supported and allows for parallel package builds.) The actual compilation of the book with **bookdown** will simply re-use these caches for efficiency.

After compilation of the individual chapters, `preCompileBook` will lock the entire `work.dir`. This ensures that **bookdown**'s directory shuffling does not break concurrent processes using the **knitr** cache directories. The lock can be released by passing the returned handle to `handle` in `postCompileBook`.

Value

For preCompileBook, work is populated with the book sources and intermediate content (e.g., caches). A lock handle is returned.

For postCompileBook, final is populated with the HTMLs. Cache directories are moved out of _bookdown_files into their original location.

In both cases, a NULL is invisibly returned.

Author(s)

Aaron Lun

See Also

configureBook, where this function is called in the Makefile.

bookCache, the default choice for work.dir.

---

compileChapter *Compile a Rmarkdown file*

Description

Compile a Rmarkdown file - typically a chapter of a book - so that extractCached calls work correctly in other chapters.

Usage

```
compileChapter(path, cache = TRUE)
```

Arguments

path	String containing a path to an Rmarkdown file.
cache	Logical scalar indicating whether the compilation should be cached.

Details

Compilation is performed in a completely fresh R session, to ensure that objects, globals and loaded packages from one chapter do not affect the next chapter.

If an error is encountered during compilation of any Rmarkdown file, the standard output of render leading up to the error is printed out before the function exists.

Value

The specified file is (re)compiled to generate the corresponding *_cache directories. NULL is invisibly returned.

Author(s)

Aaron Lun

See Also

extractCached, which calls this function.

Examples

```
tmp <- tempfile(fileext=".Rmd")
write(file=tmp, "```{r, echo=FALSE, results='asis'}
rebook::chapterPreamble()
```

```{r}
rodan <- 1
```")

compileChapter(tmp)

file.exists(sub(".Rmd$", ".html", tmp)) # output HTML exists.
file.exists(sub(".Rmd$", "_cache", tmp)) # output cache exists.
exists("rodan") # FALSE
```

---

# configureBook *Helper configuration function for books*

---

## Description

Helper function to run at the top-level directory of Bioconductor book packages, to prepare for book compilation and to set up install-time resources for linking from other books.

## Usage

```
configureBook(prefix = NULL, input = "index.Rmd", redirect = NULL)
```

Arguments

prefix	Optional string containing the prefix to be used when linking from other books.
input	Name of the index file for the book, see scrapeReferences.
redirect	Optional name of the file containing redirection information, to be passed to createRedirects.
```

Details

This function assumes that the **bookdown**-formatted book is located at `inst/book` inside the package. `input` is interpreted relative to this location, e.g., if `input="index.Rmd"`, the file should be located at `inst/book/index.Rmd`.

Similarly, `redirect` is provided, the file should already be present in `vignettes/`. For example, if `redirect="redirect.txt"`, the file should be located at `vignettes/redirect.txt`.

Value

A number of files are created in the package directory.

- A `"references.csv"` file is created in the `inst/rebook` directory, containing the table of references from `scrapeReferences`.
- If `prefix` is specified, a `"prefix.csv"` file is also created in `inst/rebook`. This contains the preferred prefix of the book.
- A Makefile is created in `vignettes/` that triggers book compilation. This will also generate HTMLs for redirection via `createRedirects` if `redirect` is provided.
- A stub vignette at `vignettes/stub.Rmd` is created that redirects to the deployed book location.

Author(s)

Aaron Lun

See Also

`scrapeReferences`, which is called by this function to create the reference table.

`link`, which is used by other books to link to the configured book.

`createMakefile` *Create a compilation Makefile*

Description

Create a Makefile for compiling individual chapters, in a manner that respects the dependencies between chapters.

Usage

```
createMakefile(dir = ".", pattern = "\\.Rmd$", ..., fname = "Makefile")
```

Arguments

`dir`	String containing the path to the directory containing Rmarkdown reports. This is searched recursively for all files ending in `".Rmd"`.
`...`	Further arguments to pass to `buildChapterGraph`.
`fname`	String containing the name of the output Makefile.

Details

The main benefit of using a Makefile is that the generation of the chapter caches can be done in parallel. Then, the **bookdown** step can just serially retrieve the cache contents for rapid rendering.

The Makefile uses the markdown output file as an indicator of successful knitting of a chapter. Caches are left in the current working directory after the compilation of each report. It is assumed that **bookdown**'s render_book is smart enough to find and use these caches.

Value

A Makefile is created in dir with the name fname and a NULL is invisibly returned.

Author(s)

Aaron Lun

See Also

buildChapterGraph, to detect dependencies between chapters.

Examples

```
dir <- tempfile()
dir.create(dir)

tmp1 <- file.path(dir, "alpha.Rmd")
write(file=tmp1, "```{r, echo=FALSE, results='asis'}
rebook::chapterPreamble()
```

```{r}
rodan <- 1
```")

tmp2 <- file.path(dir, "bravo.Rmd")
write(file=tmp2, "```{r, echo=FALSE, results='asis'}
rebook::chapterPreamble()
```

```{r}
extractCached('alpha.Rmd')
```")

# Creating the Makefile:
createMakefile(dir)
cat(readLines(file.path(dir, "Makefile")), sep="\n")
```

`createRedirects` *Create redirection pages*

Description

Create HTML pages to redirect users to the latest version of the relevant Bioconductor book. This is useful for preserving compatibility with old links when reorganizing the contents of a book.

Usage

```
createRedirects(
  name,
  pkg,
  page,
  dir = "../inst/doc/book",
  file = NULL,
  check = FALSE,
  include.gif = TRUE
)
```

Arguments

`name`	Character vector containing the name of each HTML.
`pkg`	Character vector containing the name of the Bioconductor book package to redirect to.
`page`	Character vector containing the name of the new chapter to redirect to.
`dir`	String containing the path to the output directory for the HTMLs.
`file`	String containing the name of a comma-separate file with three unnamed columns, from which to derive `name`, `pkg` and `page`.
`check`	Logical scalar indicating whether to check if the destination URL exists.
`include.gif`	Logical scalar indicating whether a GIF should be included in the redirection notice.

Details

This function is intended to be called inside the Makefile generated by `configureBook`, which will create the necessary HTMLs at package build time. The expectation is that there is a file like `redirect.txt` that can be passed in as the `file` argument. The default `dir` is the same as the final destination for all HTMLs that is defined in the Makefile.

In `file`, the last column can be left empty for any row. This will instruct `createRedirects` to reuse `name` as `page`, which is convenient when a chapter is simply moved to another package without a change in the HTML file name.

It is probably a good idea to run with `check=TRUE` on occasion, to verify that the redirections are working. This is not done by default to avoid a chicken-and-egg situation where two books cannot build because they redirect to each other.

Value

HTMLs of the specified `name` are created in `dir`, redirected to the sites defined by their respective `pkg` and `page` entries. A `NULL` is invisibly returned.

Author(s)

Aaron Lun

Examples

```
tmp <- tempfile()
dir.create(tmp)
createRedirects("BLAH.html", pkg="OSCA.intro", page="installation.html", dir=tmp)

if (interactive()) {
    browseURL(file.path(tmp, "BLAH.html"))
}
```

---

deployCustomCSS *Deploy a custom CSS*

Description

Deploy a custom CSS to change the colors of the book's section headers, mostly to add some flavor to the book.

Usage

```
deployCustomCSS(path = "style.css", h2.col = "#87b13f", h3.col = "#1a81c2")
```

Arguments

`path`	String containing the path to the output CSS file.
`h2.col`	String containing the color to use for the section headers.
`h3.col`	String containing the color to use for the subsection headers.

Details

We quickly learned that it was unwise to be too adventurous with the colors. In particular, changing the colors of the table of contents was quite distracting. Altering the colors of the section headers provides a tasteful level of customization, with the default colors set (almost) to the Bioconductor color palette.

Value

The CSS file is overwritten at `path`. A `NULL` is invisibly returned.

Author(s)

Aaron Lun, based on work by Rob Amezquita and Kevin Rue-Albrecht

Examples

```
fname <- tempfile(fileext=".css")
deployCustomCSS(fname)
cat(readLines(fname), sep="\n")
```

---

extractCached *Extract cached objects*

Description

Extract specific R objects from the **knitr** cache of a previously compiled Rmarkdown file (the "donor") so that it can be used in the compilation process of another Rmarkdown file (the "acceptor").

Usage

```
extractCached(path, chunk, objects, envir = parent.frame(1), link.text = NULL)
```

Arguments

path	String containing the path to the donor Rmarkdown file.
chunk	String containing the name of the requested chunk.
objects	Character vector containing variable names for one or more objects to be extracted.
envir	Environment where the loaded objects should be stored. Defaults to the environment in which this function is called.
link.text	String containing an Rmarkdown-formatted link to the donor file, to be inserted in the collapsible element's title. If NULL, we attempt to construct this automatically from path using rmd2id. If NA, no link text is inserted.

Details

Each R object is extracted in its state at the requested chunk and inserted into envir. Note that the object does not have to be generated or even referenced in the requested chunk, provided it was generated in a previous *named* chunk.

The parser in this function is rather limited, so the donor Rmarkdown file is subject to several constraints:

- All chunks involved in generating the requested objects (indirectly or otherwise) should be named.
- All named chunks should be executed; eval=FALSE is not respected.

- All relevant code occurs within triple backticks, i.e., any inline code should be read-only.

Unnamed chunks are allowed but cannot be referenced and will not be used for searching for objects. Chunks with names starting with `unref-` are considered to be the same as unnamed chunks and will be ignored; this is useful for figure-generating chunks that need to be referenced inside the donor report. In general, neither of these should be used for code that might affect variables in the named chunks, i.e., code in unnamed chunks should be "read-only" with respect to variables in the named chunks.

Obviously, this entire process assumes that donor report has already been compiled with `cache=TRUE`. If not, `extractCached` will compile it (and thus generate the cache) using `compileChapter`. A report-specific lock is applied during this process to avoid problems with concurrent compilation.

Value

Variables with names `objects` are created in `envir`. A markdown chunk (wrapped in a collapsible element) is printed that contains all commands needed to generate those objects, based on the code in the named chunks of the donor Rmarkdown file.

Author(s)

Aaron Lun

See Also

`setupHTML` and `chapterPreamble`, to set up the code for the collapsible element.

`compileChapter`, to compile a Rmarkdown report to generate the cache.

Examples

````
# Mocking up an Rmarkdown report.
donor <- tempfile(fileext=".Rmd")
write(file=donor, "```{r some-monsters}
destoroyah <- 1
mecha.king.ghidorah <- 2
```

```{r more-monsters}
space.godzilla <- 3
```

```{r}
msg <- 'I am not referenced.'
```

```{r unref-figure}
plot(1, 1, main='I am also not referenced.')
```

```{r even-more-monsters}
megalon <- 4
```")
````
````

```
# Extracting stuff from it in another report.
acceptor <- tempfile(fileext=".Rmd")
dpb <- deparse(basename(donor))
write(file=acceptor, sprintf("```{r, echo=FALSE, results='asis'}
chapterPreamble()
```

```{r, results='asis', echo=FALSE}
extractCached(%s, chunk='more-monsters',
   objects=c('space.godzilla', 'destoroyah'))
```

```{r}
space.godzilla * destoroyah
```

```{r, results='asis', echo=FALSE}
extractCached(%s, chunk='even-more-monsters',
   objects=c('megalon', 'mecha.king.ghidorah'))
```

```{r}
mecha.king.ghidorah * megalon
```", dpb, dpb))

rmarkdown::render(acceptor)
if (interactive()) browseURL(sub(".Rmd$", ".html", acceptor))
```

---

`extractFromPackage` *Extract cached objects from package's Rmarkdown files*

---

## Description

Extract and compile Rmarkdown files from a "donor" package's installation directory, extracting cached objects from the subsequent **knitr** cache.

## Usage

```
extractFromPackage(
  rmd.name,
  ...,
  package,
  envir = parent.frame(1),
  src.name = "book",
  work.dir = bookCache(package)
)
```
```

Arguments

rmd.name	String containing the path to the donor Rmarkdown file, relative to work.
..., envir	Further arguments to pass to extractCached.
package	String containing the name of the donor package.
src.name	String containing the name or relative path of the subdirectory in the donor package's installation directory that contains all the Rmarkdown files.
work.dir	String containing the path to a versioned cache directory to hold the contents of donor book, see the **dir.expiry** package for details.

Details

This function assumes that all potential donor Rmarkdown files for package are present in the directory src.name. It copies the contents of src.name into work.dir and calls extractCached on the rmd.name inside. The desired objects are then extracted from the subsequent **knitr** cache.

The work.dir directory should be set to a persistent cache to enable greater re-use of the cache across calls and R sessions. Indeed, the default here is the same as that used by preCompileBook, so we can avoid recopmilation if the donor book has already been compiled via the latter function. This function will respect any global locks imposed by other functions in the process of performing the copy (or other rearrangements).

Value

A NULL is invisibly returned. Objects are created in envir and a code chunk is printed; see extractCached for more details.

Author(s)

Aaron Lun

Examples

```
# Only specifyin 'work.dir' here for demonstration purposes.
# For actual use, just leave it as the default.
tmp <- file.path(tempfile(), "1.0.0")

extractFromPackage("test.Rmd", chunk="ghidorah-1964", src.name="example",
    objects="godzilla", package="rebook", work.dir=tmp)

list.files(tmp)
godzilla
```

`link` *Create a link to a different book*

Description

From another Rmarkdown file, create a link to a section or figure of a **rebook**-configured book.

Usage

```
link(id, package, type = NULL, prefix = NULL, df = NULL, error = TRUE)
```

Arguments

`id`	String containing an identifier for a section or figure.
`package`	String containing the name of the package containing the target book.
`type`	String containing the type of the link, e.g., `"Section"` or `"Figure"`, to be added to the link text. This is automatically determined if not provided. If `NA`, the type is not added to the link text.
`prefix`	String specifying the prefix to use on `type`. This is automatically determined from `package`'s chosen prefix or, if that is not provided, using the package name itself. If `NA`, no prefix is added. Only used if `type` is not `NA`.
`df`	A data.frame containing all links for `package`. Only used for testing.
`error`	Logical scalar indicating whether an error should be raised if the link cannot be found.

Details

We expect that the target book is set up as a Bioconductor package with a `configure` file that runs `configureBook`. This function will then retrieve install-time information from that package to create necessary hyperlinks to the Bioconductor-hosted book content.

Value

String containing a markdown-formatted link to the relevant part of the target book. If the link cannot be constructed and `error=FALSE`, a `NULL` is instead returned.

Author(s)

Aaron Lun

See Also

`configureBook`, which should be run by the authors of `package`.

`scrapeReferences`, to generate a `df` for testing.

Examples

```
# Only using 'df=' here because 'testpackage' doesn't actually exist.
link("fig:xxx", package="testpackage",
    df=data.frame(id='fig:xxx', file='whee.html', text='3.1'))

link("fig:xxx", package="testpackage", type=NA,
    df=data.frame(id='fig:xxx', file='whee.html', text='3.1'))

link("fig:xxx", package="testpackage", prefix=NA,
    df=data.frame(id='fig:xxx', file='whee.html', text='3.1'))
```

---

openingDetails *Report opening details about the book*

Description

Report opening details about the book, to be executed as an R expression in the `Date:` field.

Usage

```
openingDetails(..., Copyright = NULL)
```

Arguments

`...`	Further named strings to be included in the opening details.
`Copyright`	String containing copyright information; defaults to "`Bioconductor, <current year>`".

Details

It is usually sufficient to set something like

```
date: "`r rebook::openingDetails()`"
```

in the YAML header of the book, thereby ensuring that the book details are printed after the title but before any contents. This assumes that none of the details have problematic characters, particularly double quotes.

Details are extracted from a `DESCRIPTION` file in the current or any parent directory. This assumes that authors are formatted as `Authors@R` and the `License` and `Date` fields are specified.

Value

A string containing the formatted details for inclusion into a YAML header.

Author(s)

Aaron Lun

Examples

```
wd <- getwd()
setwd(system.file(package="rebook"))
cat(openingDetails(), '\n')
setwd(wd)
```

`prettySessionInfo` *Pretty session info*

Description

Wraps the session information output chunk in a collapsible HTML element so that it doesn't dominate the compiled chapter.

Usage

```
prettySessionInfo()
```

Value

Prints a HTML block containing a collapsible section with session information.

Author(s)

Aaron Lun

See Also

`setupHTML` and `chapterPreamble`, to set up the code for the collapsible element.

Examples

````
tmp <- tempfile(fileext=".Rmd")
write(file=tmp, "```{r, echo=FALSE, results='asis'}
rebook::setupHTML()
```

```{r, results='asis'}
prettySessionInfo()
```")

rmarkdown::render(tmp)

if (interactive()) browseURL(sub(".Rmd$", ".html", tmp))
````
````

rmd2id *Get the chapter identifier*

Description

Get the identifier for a book chapter given the Rmarkdown source code. This is usually derived from the chapter title but can also be explicitly specified.

Usage

```
rmd2id(path)
```

Arguments

path	String containing the path to the Rmarkdown file for a chapter.

Value

String containing the identifier for this chapter. If no identifier can be determined, NULL is returned.

Author(s)

Aaron Lun

Examples

```
tmp <- tempfile(fileext='.Rmd')
write('# some chapter name

blah', file=tmp)
rmd2id(tmp)

tmp2 <- tempfile(fileext='.Rmd')
write('# some chapter name {#chapter-id}

blah', file=tmp2)
rmd2id(tmp2)
```

`scrapeDependencies` *Scrape dependencies*

Description

Scrape Rmarkdown reports in the book for all required dependencies.

Usage

```
scrapeDependencies(dir, recursive = TRUE, pattern = "\\.Rmd$")
```

Arguments

`dir`	String containing the path to the directory containing Rmarkdown reports. This is searched recursively for all files ending in `".Rmd"`.
`recursive, pattern`	Further arguments to pass to `list.files` when searching for Rmarkdown reports.

Details

The output of this should be added to the `Suggests` field of the book's `DESCRIPTION`, to make it easier to simply install all of its required dependencies.

Note that dependencies in inline code sections are not detected, so these should be explicitly mentioned in a standalone code chunk to be captured.

Value

Character vector of required packages.

Author(s)

Aaron Lun

Examples

```
tmp <- tempfile(fileext=".Rmd")
write(file=tmp, "```{r}
A::a()
```

```{r}
library(B)
require(C)
```")

scrapeDependencies(tempdir())
```
```

`scrapeReferences` *Scrape references from a* **bookdown** *directory*

Description

Scrape references to sections and figures from all Rmarkdown files in a **bookdown** directory.

Usage

```
scrapeReferences(dir, input = "index.Rmd", workdir = tempfile(), clean = TRUE)
```

Arguments

`dir`	String containing a path to a **bookdown**-containing directory.
`input`	String containing the name of the file to use in the `render_book` statement.
`workdir`	String containing a path to a working directory to use to store bits and pieces.
`clean`	Logical scalar indicating whether the working directory should be removed upon function completion.

Details

This function works by performing a quick dummy compilation of the book, turning off all evaluations with a global `eval=FALSE`. It then trawls the set of newly created HTML files, pulling out the section/figure identifiers and collating them into a data.frame.

The goal is to facilitate convenient linking between books by automatically filling in the file and text for a given link. Packages that deploy books should run this in their `configure` scripts to obtain a reference mapping that they can serve to other packages via `link`.

Extraction of the figure text assumes that the figure prefix ends with a non-numeric character, e.g., `"Figure "` or `"Figure S"`.

Value

A data.frame where each row corresponds to a reference. It has `id`, the name of the reference; `file`, the compiled HTML file that the reference comes from; and `text`, the text to be associated with that reference.

Author(s)

Aaron Lun

See Also

`link`, to create links given a package name and identifier.

Examples

```
book.dir <- system.file("example", package="rebook")
df <- scrapeReferences(book.dir)
df
```

`setupHTML` *Set up HTML elements*

Description

Set up Javascript and CSS elements for each chapter, primarily for the custom collapsible class.

Usage

```
setupHTML()
```

Details

The custom collapsible class allows us to hide details until requested by the user. This improves readability by reducing the clutter in the compiled chapter.

Value

Prints HTML to standard output set up JS and CSS elements.

Author(s)

Aaron Lun

See Also

`chapterPreamble`, which calls this function.

`extractCached` and `prettySessionInfo`, which use the custom collapsible class.

Examples

```
setupHTML()
```

`updateDependencies` *Update the dependencies*

Description

Update the book package's DESCRIPTION file with the latest dependencies.

Usage

```
updateDependencies(
  dir = ".",
  path = file.path(dir, "DESCRIPTION"),
  extra = NULL,
  indent = 4,
  field = "Depends",
  ...
)
```

Arguments

`dir`	String containing the path to the directory containing the book.
`path`	String containing the path to the DESCRIPTION file.
`extra`	Character vector of extra packages to be added to imports, usually from packages that are in `Suggests` and thus not caught directly by `scrapeDependencies`.
`indent`	Integer scalar specifying the size of the indent to use when listing packages.
`field`	String specifying the dependency field to store the packages in. Defaults to `"Suggests"` by convention.
`...`	Further arguments to pass to `scrapeDependencies`.

Details

The book `DESCRIPTION` is useful for quick installation of all packages required across all chapters. For example, it is used by `https://github.com/LTLA/TrojanBookBuilder` to populate a trojan package's dependencies, ensuring that all packages are available when the book itself is compiled.

Value

The specified `field` in the `DESCRIPTION` file in `dir` is updated. `NULL` is invisibly returned.

Author(s)

Aaron Lun

Examples

```
dir <- tempfile()
dir.create(dir)

write(file=file.path(dir, "DESCRIPTION"),
"Package: son.of.godzilla
Version: 0.0.1
Description: Like godzilla, but smaller.")

tmp <- file.path(dir, "alpha.Rmd")
write(file=tmp, "```{r, echo=FALSE, results='asis'}
rebook::chapterPreamble()
```

```{r}
A::func
library(C)
```")

tmp <- file.path(dir, "bravo.Rmd")
write(file=tmp, "```{r, echo=FALSE, results='asis'}
rebook::chapterPreamble()
```

```{r}
require(D)
B::more
```")

updateDependencies(dir)
cat(readLines(file.path(dir, "DESCRIPTION")), sep="\n")
```

Index

bioc-images, 2
BiocFavicon *(*bioc-images*)*, 2
Biocpkg, *5*
BiocSticker *(*bioc-images*)*, 2
bookCache, 3, *8*
bookCacheExpiry *(*bookCache*)*, 3
buildChapterGraph, 4, *10, 11*

cat, *6*
chapterPreamble, 5, *15*, *20*, *24*
collapseEnd *(*collapseStart*)*, 6
collapseStart, 6
compileBook, 7
compileChapter, *7*, 8, *15*
configureBook, *3*, *8*, 9, *12*, *18*
createMakefile, 10
createRedirects, *9, 10*, 12

deployCustomCSS, 13

extractCached, *4*, *8, 9*, 14, *17*, *24*
extractFromPackage, *3*, *7*, 16

graph, *4*

knit, *11*

link, *9, 10*, 18, *23*
list.files, *4*, *22*

openingDetails, 19

postCompileBook *(*compileBook*)*, 7
preCompileBook, *17*
preCompileBook *(*compileBook*)*, 7
prettySessionInfo, 20, *24*

render, *8*
render_book, *7*, *23*
rmd2id, *14*, 21

scrapeDependencies, 22, *25*
scrapeReferences, *9, 10*, *18*, 23
setupHTML, *5*, *15*, *20*, 24

updateDependencies, 25