Package ‘Rdpack’

February 19, 2022

Type Package

Title Update and Manipulate Rd Documentation Objects

Version 2.1.4

Date 2022-02-18

Description Functions for manipulation of R documentation objects, including functions reprompt() and ereprompt() for updating 'Rd' documentation for functions, methods and classes; 'Rd' macros for citations and import of references from 'bibtex' files for use in 'Rd' files and 'roxygen2' comments; 'Rd' macros for evaluating and inserting snippets of 'R' code and the results of its evaluation or creating graphics on the fly; and many functions for manipulation of references and Rd files.

URL https://geobosh.github.io/Rdpack/ (website),
https://github.com/GeoBosh/Rdpack (devel)

BugReports https://github.com/GeoBosh/Rdpack/issues

Depends R (>= 2.15.0), methods

Imports tools, utils, rbibutils (>= 1.3)

Suggests grDevices, testthat, rstudioapi, rprojroot, gbRd

License GPL (>= 2)

LazyLoad yes

RoxygenNote 7.1.1

NeedsCompilation no

Author Georgi N. Boshnakov [aut, cre], Duncan Murdoch [ctb]

Maintainer Georgi N. Boshnakov <georgi.boshnakov@manchester.ac.uk>

Repository CRAN

Date/Publication 2022-02-18 23:30:02 UTC
**R topics documented:**

- Rdpack-package ........................................... 3
- append_to_Rd_list ....................................... 9
- char2Rdpiece ............................................. 11
- compare_usage ........................................... 12
- c_Rd ...................................................... 13
- deparse_usage ........................................... 14
- ereprompt ................................................ 16
- format_funusage ......................................... 17
- get_bibentries ........................................... 19
- get_sig_text ............................................. 21
- get_usage_text ........................................... 23
- insert_all_ref .......................................... 24
- insert_citeOnly ......................................... 25
- insert_ref ................................................ 26
- inspect_args ............................................ 30
- inspect_Rd ............................................... 31
- inspect_signatures ....................................... 32
- inspect_slots ............................................ 33
- inspect_usage ........................................... 34
- list_Rd ................................................... 35
- makeVignetteReference .................................. 36
- parse_pairlist ......................................... 38
- parse_Rdname ........................................... 40
- parse_Rdpiece .......................................... 41
- parse_Rdtext ........................................... 42
- parse_usage_text ....................................... 43
- predefined ............................................... 44
- promptPackageSexpr ..................................... 46
- promptUsage ............................................. 47
- Rdapply .................................................. 50
- Rdo2Rdf .................................................. 52
- Rdo_append_argument .................................... 54
- Rdo_collect_aliases ..................................... 55
- Rdo_empty_sections ...................................... 56
- Rdo_fetch ................................................ 57
- Rdo_flatinsert ......................................... 59
- Rdo_get_argument_names ................................ 60
- Rdo_get_insert_pos ..................................... 61
- Rdo_get_item_labels .................................... 62
- Rdo_insert ............................................... 63
- Rdo_insert_element ..................................... 64
- Rdo_is_newline .......................................... 64
- Rdo_locate ............................................... 65
- Rdo_locate_leaves ...................................... 67
- Rdo_macro ................................................ 68
- Rdo_modify ............................................... 69
Rdpack-package

Update and Manipulate Rd Documentation Objects

Description

Functions for manipulation of R documentation objects, including functions reprompt() and ere-
prompt() for updating 'Rd' documentation for functions, methods and classes; 'Rd' macros for
citations and import of references from 'bibtex' files for use in 'Rd' files and 'roxygen2' comments;
'Rd' macros for evaluating and inserting snippets of 'R' code and the results of its evaluation or
creating graphics on the fly; and many functions for manipulation of references and Rd files.

Details

Package: Rdpack
Type: Package
Version: 2.1.4
Date: 2022-02-18
License: GPL (>= 2)
LazyLoad: yes
Built: R 4.0.3; ; 2022-02-18 09:42:54 UTC; unix

Package Rdpack provides a number of functions for maintenance of documentation in R packages. Although base R and package methods have functions for creation of skeleton documentation, if a function gets a new argument or a generic gets a new method, then updating existing documentation
is somewhat inconvenient. This package provides functions that update parts of the Rd documentation that can be dealt with automatically and leave manual changes untouched. For example, usage sections for functions are updated and if there are undescribed arguments, additional items are put in the ‘arguments’ section.

A set of functions and macros support inclusion of references and citations from BibTeX files in R documentation (Rd and roxygen2). These tools use only facilities provided by base R and package \texttt{rbibutils} (Boshnakov and Putman 2020).

There are also convenience macros for inclusion of evaluated examples and graphs, which hide some of the hassle of doing this directly with the underlying \texttt{Sexpr}'s.

The subsections below give additional details, see also the vignettes.

**Creating and updating Rd files:**

The main function provided by this package is \texttt{reprompt}. There is also a function \texttt{promptPackageSexpr} for creating initial skeleton for overall package description such as this help page. \texttt{reprompt} produces a skeleton documentation for the requested object, similarly to functions like \texttt{prompt}, \texttt{promptMethods}, and \texttt{promptClass}. Unlike those functions, \texttt{reprompt} updates existing documentation (installed or in an Rd object or file) and produces a skeleton from scratch as a last resort only. If the documentation object describes more than one function, all descriptions are updated. Basically, \texttt{reprompt} updates things that are generated automatically, leaving manual editing untouched.

The typical use of \texttt{reprompt} is with one argument, as in

```r
reprompt(infile = "/Rdpack/man/reprompt.Rd")
reprompt(reprompt)
reprompt("reprompt")
```

\texttt{reprompt} updates the documentation of all objects described in the Rd object or file, and writes the updated Rd file in the current working directory, see \texttt{reprompt} for details. To describe a new function in an existing Rd file, just add something like \texttt{myfun()} and run \texttt{reprompt} to insert the correct signature, alias, etc. This works also for replacement functions, see \texttt{reprompt} for details.

\texttt{ereprompt} updates the Rd documentation in a file, overwrites it and opens it in an editor. It calls \texttt{reprompt} to do the actual job but has different defaults. Which editor is opened, is system dependent, see the \texttt{edit} and \texttt{ereprompt} for further details.

Users who work on Rd files in RStudio can use add-in “Reprompt” to invoke \texttt{reprompt} conveniently on an Rd file or on a selected object in an R source code file, see \texttt{RStudio_reprompt}. This add-in was contributed by Duncan Murdoch.

Users of Emacs/ESS have various options, depending on their workflow. One approach is to define a key to call \texttt{ereprompt} on the file being edited, see \texttt{georgisemacs} for an example setup. \texttt{promptPackageSexpr} creates a skeleton for a package overview in file name-package.Rd. Then the file can be edited as needed. This function needs to be called only once for a package since automatic generation of information in name-package.Rd is achieved with \texttt{Sexpr}'s at build time, not with verbatim strings (\texttt{promptPackage} used to insert verbatim strings but in recent versions of R it also uses macros.).

For example, the source of this help page is file 'Rdpack-package.Rd'. It was initially produced using

```r
promptPackageSexpr("Rdpack")
```
The factual information at the beginning of this help topic (the index above, the version and other stuff that can be determined automatically) is kept automatically up to date.

**References and Citations:**
Another set of functions is for management of bibliographic references in Rd files. The old approach based on function `rebib` is fully functional, see below, but the recommended way to insert references and citations is based on Rd macros.

The provided Rd macros are fully portable and, in particular, work in Rd files and roxygen2 comments, see `insertRef` and vignette `vignette("Inserting_bibtex_references","Rdpack")` for details and examples.

The BibTeX source for the references and citations produced by the Rd macros is file "REFERENCES.bib", which should be located in the root of the package installation directory. **Rdpack** needs also to be mentioned in two places in file ‘DESCRIPTION’. These one-off preparation steps are enumerated below:

1. Put the following line in file ‘DESCRIPTION’:

   ```
   RdMacros: Rdpack
   ```

   (If there is already a line starting with 'RdMacros:', add Rdpack to the list on that line.)

2. Add **Rdpack** to the list of imports (Imports: field) in file ‘DESCRIPTION’.

3. Add the following line to file ‘NAMESPACE’:

   ```
   importFrom(Rdpack,reprompt)
   ```

   Alternatively, if `devtools` is managing your NAMESPACE file, the equivalent `roxygen2` line is:

   ```
   #' @importFrom Rdpack reprompt
   ```

4. Create file "REFERENCES.bib" in subdirectory "inst/" of the package and put the BibTeX references in that file.

The Rd macro `\insertRef` takes two arguments: a BibTeX key and the name of a package. Thus, `\insertRef{key}{package}` inserts the reference whose key is `key` from "REFERENCES.bib" of the specified package (almost always the one being documented).

Citations can be done with Rd macro `\insertCite`, which inserts citation(s) for one or more BibTeX keys and records the keys. `\insertCiteOnly` is similar to `\insertCite` but does not record the keys. `\insertNoCite` records the keys but does not produce citations.

`\insertAllCited` creates a bibliography including all references recorded by `\insertCite` and `\insertNoCite`. It is usually put in section “References”, something like:

```
\references{
   \insertAllCited{}
}
```

in an Rd file. Don’t align the backslash with the second ‘e’ of @references, since roxygen2 may interpret it as verbatim text, not macro.

The analogous documentation chunk in roxygen2 might look like this:

```
#' @references
#' \insertAllCited{}
```
Bibliography styles for lists of references are supported as well. Currently the only alternative offered is to use long names (Georgi N. Boshnakov) in place of the default style (Boshnakov GN). More comprehensive alternatives can be included if needed or requested.

Convenience functions `makeVignetteReference` and `vigbib` generate Bibtex entries for vignettes.

**Previewing documentation pages:**
It is convenient during development to be able to view the rendered version of the document page being edited. The function `viewRd` renders a documentation file in a source package and displays it as text or in a browser. It renders references properly in any workflow, including `devtools` development mode (Wickham et al. 2018) in Emacs/ESS, Rstudio, Rgui. This function is a good candidate to be assigned to a key in editors which support this.

I created this function (in 2017) since the functions provided by `devtools` and Emacs/ESS are giving errors when processing pages containing Rd macros.

**Static Management of References:**
In the alternative approach, the function `rebib` updates the bibliographic references in an Rd file.
Rdpack uses a simple scheme for inclusion of bibliographic references. The key for each reference is in a TeX comment line, as in:

```latex
\references{
  ...
  % bibentry: key1
  % bibentry: key2
  ...
}
```

`rebib` puts each reference after the line containing its key. It does nothing if the reference has been put by a previous call of `rebib`. If the Bibtex entry for some references changes, it may be necessary to update them in the Rd file, as well. Call `rebib` with `force = TRUE` to get this effect. There is also a facility to include all references from the Bibtex file, see the documentation of `rebib` for details.

**Inserting evaluated examples:**
Sometimes the documentation of an object becomes more clear if accompanied by snippets of R code and their results. The standard Rd macro `\Sexpr` caters for a number of possibilities to evaluate R code and insert the results and the code in the documentation. The Rd macro `\printExample` provided by package `Rdpack` builds on it to print a snippet of R code and the results of its evaluation, similarly to console output but the code is not prefixed and the results are prefixed with comment symbols. For example, `\printExample{2+2; a <- 2*3; a}` produces the following in the rendered documentation:

```
2 + 2
##: [1] 4
a <- 2 * 3
a
##: [1] 6
```

The help page of `promptUsage` contains a number of examples created with `\printExample`. The corresponding Rd file can be obtained from the package tarball or from https://github.com/GeoBosh/Rdpack/blob/master/man/promptUsage.Rd.
The argument of \code{\texttt{printExample}} must be on a single line with versions of \proglang{R} before \proglang{R} 3.6.0. \code{\texttt{printExample}} is typically placed in section Details of an object's documentation, see section Details of \code{get_usage} for anumber of examples produced mostly with \code{\texttt{printExample}}.

The macro \code{runExamples} can be used as a replacement of section Examples. For example, if the following code is put at the top level in an Rd file (i.e. not in a section):

\begin{verbatim}
\runExamples{2+2; a <- 2*3; a}
\end{verbatim}

then it will be evaluated and replaced by a normal section examples:

\begin{verbatim}
\examples{
  2 + 2
  ###: 4
  a <- 2 * 3
  a
  ###: 6
}
\end{verbatim}

This generated examples section is processed by the standard \proglang{R} tools (almost) as if it was there from the outset. In particular, the examples are run by the \proglang{R}'s quality control tools and tangled along with examples in other documentation files.

In \proglang{R} versions before 3.6.0 \proglang{R CMD} check used to give a warning about unknown \Sexpr{\texttt{\$expr}} section at top level.

**Creating and including graphs:**

Figures can be inserted with the help of the standard Rd markup command \code{\texttt{\texttt{figure}}}. The Rd macro \code{\texttt{\texttt{insertFig}}} provided by package \pkg{Rdpack} takes a snipped of \proglang{R} code, evaluates it and inserts the plot produced by it (using \code{\texttt{\texttt{figure}}}). \code{\texttt{\texttt{insertFig}}} takes three arguments: a filename, the package name and the code to evaluate to produce the figure. For example,

\begin{verbatim}
\insertFig{cars.png}{mypackage}{x <- cars$speed; y <- cars$dist; plot(x,y)}
\end{verbatim}

will evaluate the code, save the graph in file "man/figures/cars.png" subdirectory of package "mypackage", and include the figure using \code{\texttt{\texttt{figure}}}. Subdirectory "\texttt{\texttt{figures}}" is created if it doesn’t exist. Currently the graphs are saved in "\texttt{\texttt{png}}" format only. In older versions of \proglang{R} the code should be on a single line for the reasons explained in the discussion of \code{\texttt{printExample}}.

The sister macro \code{\texttt{\texttt{makeFig}}} creates the graph in exactly the same way as \code{\texttt{\texttt{insertFig}}} but does not insert it. This can be done with a separate \code{\texttt{\texttt{figure}}} command. This can be used if additional options are desired for different output formats, see the description of \code{\texttt{\texttt{figure}}} in "Writing \proglang{R} extensions".

Other functions that may be useful are \code{\texttt{Rdo2Rdf}}, \code{\texttt{Rdapply}} and \code{\texttt{Rd_combo}}. Here is also brief information about some more technical functions that may be helpful in certain circumstances.

\code{\texttt{get_usage}} generates usage text for functions and methods. The functions can be located in environments or other objects. This may be useful for description of function elements of compound objects.

\code{\texttt{c_Rd}} concatenates Rd pieces, character strings and lists to create a larger Rd piece or a complete Rd object. \code{\texttt{list_Rd}} is similar to \code{\texttt{c_Rd}} but provides additional features for convenient assembling of Rd objects.

\code{\texttt{parse_Rdpiece}} is a technical function for parsing pieces of Rd source text but it has an argument to return formatted help text which may be useful when one wishes to show it to the user.

\code{\texttt{Rdo_set_section}} can be used to set a section, such as "\texttt{\texttt{author}}".

The remaining functions in the package are for programming with Rd objects.
**Note**

All processing is done on the parsed Rd objects, i.e. objects of class "Rd" or pieces of such objects (Murdoch 2010).

The following terminology is used (todo: probably not yet consistently) throughout the documentation.

"Rd object" - an object of class Rd, or part of such object.
"Rd piece" - part of an object of class Rd. Fragment is also used but note that parse_Rd defines fragment more restrictively.
"Rd text", "Rd source text", "Rd format" - these refer to the text of the Rd files.

**Author(s)**

Georgi N. Boshnakov [aut, cre], Duncan Murdoch [ctb]
Maintainer: Georgi N. Boshnakov <georgi.boshnakov@manchester.ac.uk>

**References**

**Note:** Reference ZZZ (2018) does not exist. It is a test that simple math in BibTeX entries works.

---


A. ZZZ (2018). “A relation between several fundamental constants: $e^{i\pi} = -1.$” A non-existent journal with the formula $L_2$ in its name. This reference does not exist. It is a test/demo that simple formulas in BibTeX files are OK. A formula in field 'note': $c^2 = a^2 + b^2$.

**See Also**

erepromt, reprompt, promptPackageSexpr, rebib,
ge_get_usage,
viewRd, vigbib, makeVignetteReference,
vignette("Inserting_bibtex_references",package = "Rdpack"),
vignette("Inserting_figures_and_evaluated_examples",package = "Rdpack")

**Examples**

```r
## The examples below show typical use but are not executable.
## For executable examples see the help pages of
## reprompt, promptPackageSexpr, and rebib.
```
append_to_Rd_list

Add content to the element of an Rd object or fragment at a given position

Description

Add content to the element of an Rd object or fragment at a given position.

Usage

append_to_Rd_list(rdo, x, pos)
Arguments

rdo    an Rd object
x      the content to append, an Rd object or a list of Rd objects.
pos    position at which to append x, typically an integer but may be anything accepted
         by the operator "[[".

Details

The element of rdo at position pos is replaced by its concatenation with x. The result keeps the
"Rd_tag" of rdo[[pos]].

Argument pos may specify a position at any depth of the Rd object.

This function is relatively low level and is mainly for use by other functions.

Value

the modified rdo object

Author(s)

Georgi N. Boshnakov

Examples

#rdoseq <- utils:::.getHelpFile(help("seq"))
rdoseq <- Rdo_fetch("seq", "base")
iusage <- which(tools:::RdTags(rdoseq) == "\usage")
iusage
attr(rdoseq[[iusage]], "Rd_tag")

## append a new line after the last usage line
rdoseq2 <- append_to_Rd_list(rdoseq, list(Rdo_newline()), iusage)

## Suppose that we wish to describe the function 'sequence' in the same Rd file.
## We append an usage statement for 'sequence()', without worrying about its
## actual signature.
rdoseq2 <- append_to_Rd_list(rdoseq2, list(Rdo_Rcode("sequence()")), iusage)
Rdo_show(rdoseq2)

## the two operations can be done in one step
rdoseq3 <- append_to_Rd_list(rdoseq, list(Rdo_newline(), Rdo_Rcode("sequence()")), iusage)
Rdo_show(rdoseq3)

## now run reprompt() to update rdoseq3, namely:
## (1) it corrects the signature of 'sequence' in section \usage.
## (2) reports new argument "nvec"
## (3) inserts \item for the new argument(s) in section \arguments.
reprompt(rdoseq3, filename=NA)
char2Rdpiece

Convert a character vector to Rd piece

Description

Convert a character vector to Rd piece.

Usage

char2Rdpiece(content, name, force.sec = FALSE)

Arguments

- content: a character vector.
- name: name of an Rd macro, a string.
- force.sec: TRUE or FALSE, see 'Details'.

Details

Argument content is converted to an Rd piece using name to determine the format of the result. The Rd tag of content is set as appropriate for name. More specifically, if name is the name of a macro (without the leading '\') whose content has a known "Rdtag", that tag is used. Otherwise the tag is set to "TEXT".

If force.sec is TRUE, name is treated as the name of a top level section of an Rd object. A top level section is exported as one argument macro if it is a standard section (detected with is_Rdsecname) and as the two argument macro "\section" otherwise.

If force.sec is FALSE, the content is exported as one argument macro without further checks.

Note

This function does not attempt to escape special symbols like ‘%’.

Author(s)

Georgi N. Boshnakov

Examples

## add a keyword section
a1 <- char2Rdpiece("graphics", "keyword")
a1
## "keyword" is a standard Rd top level section, so 'force.sec' is irrelevant
a2 <- char2Rdpiece("graphics", "keyword", force.sec = TRUE)
identical(a1, a2)

## an element suitable to be put in a "usage" section
char2Rdpiece("log(x, base = exp(1))", "usage")
## a user defined section "Todo"
char2Rdpiece("Give more examples for this function.", "Todo", force.sec = TRUE)

### Description

Compare usage entries for a function to its actual arguments.

### Usage

```r
compare_usage1(urdo, ucur)
```

### Arguments

- `urdo` usage text for a function or S3 method from an Rd object or file.
- `ucur` usage generated from the actual object.

### Details

Compares the usage statements for functions in the Rd object or file `urdo` to the usage inferred from the actual definitions of the functions. The comparison is symmetric but the interpretation assumes that `ucur` may be more recent.

Note: do not compare the return value to TRUE with `identical` or `isTRUE`. The attribute makes the returned value not identical to TRUE in any case.

### Value

TRUE if the usages are identical, FALSE otherwise. The return value has attribute "details", which is a list providing details of the comparison. The elements of this list should be referred by name, since if one of `urdo` or `ucur` is NULL or NA, the list contains only the fields "obj_removed", "obj_added", "rdo_usage", "cur_usage", and "alias".

- `identical_names` a logical value, TRUE if the ‘name’ is the same in both objects.
- `obj_removed` names present in `urdo` but not in `ucur`
- `obj_added` names present in `ucur` but not in `urdo`
- `identical_argnames` a logical value, TRUE if the argument names in both objects are the same.
- `identical_defaults` a logical value, TRUE if the defaults for the arguments in both objects are the same.
- `identical_formals` a logical value, TRUE if the formals are the same, i.e. fields `identical_argnames` and `identical_defaults` are both TRUE.
Description

Concatenates Rd objects or pieces

Usage

c_Rd(...)

Arguments

... objects to be concatenated, Rd objects or character strings, see ‘Details’.

Details

The arguments may be a mixture of lists and character strings. The lists are typically "Rd" objects or pieces. The character strings may also be elements of "Rd" objects carrying "Rd_tag" attributes. The "Rd_tag" attribute of character strings for which it is missing is set to "TEXT". Finally, each character element of "\dots" is enclosed in list.

Eventually all arguments become lists and they are concatenated using c(). If any of the arguments is of class "Rd", the class of the result is set to "Rd". Otherwise, the "Rd_tag" of the result is set to the first (if any) non-null "Rd_tag" in the arguments.

The structure of "Rd" objects is described by Murdoch (2010).

Value

An Rd object or a list whose attribute "Rd_tag" is set as described in ‘Details’
Author(s)
Georgi N. Boshnakov

References

See Also
list_Rd

Examples
a1 <- char2Rdpiece("Dummyname", "name")
a2 <- char2Rdpiece("Dummyallias1", "alias")
a3 <- char2Rdpiece("Dummy title", "title")
a4 <- char2Rdpiece("Dummy description", "description")

## The following are equivalent
## (gbRd::Rdo_empty() creates an empty list of class 'Rd')
if(requireNamespace("gbRd")){
b1 <- c_Rd(gbRd::Rdo_empty(), list(a1), list(a2), list(a3), list(a4))
c1 <- c_Rd(gbRd::Rdo_empty(), list(a1, a2, a3, a4))
d1 <- c_Rd(gbRd::Rdo_empty(), list(a1, a2), list(a3, a4))
identical(c1, b1)
identical(c1, d1)
Rdo_show(b1)
}

## insert a newline
d1n <- c_Rd(gbRd::Rdo_empty(), list(a1, a2), Rdo_newline(), list(a3, a4))
str(d1n)
}

## When most of the arguments are character strings
## the function 'list_Rd' may be more convenient.
u1 <- list_Rd(name = "Dummyname", alias = "Dummyallias1",
title = "Dummy title", description = "Dummy description",
Rd_class = TRUE )
Rdo_show(u1)

deparse_usage

Convert f_usage objects to text appropriate for usage sections in Rd files

Description
Converts f_usage objects to text appropriate for usage sections in Rd files. Handles S3 methods.
**Usage**

```r
deparse_usage(x)
deparse_usage1(x, width = 72)
```

## S3 method for class 'f_usage'

```r
as.character(x, ...)
```

**Arguments**

- `x`: an object from class "f_usage". For `depase_usage`, `x` can also be a list of "f_usage" objects.
- `width`: maximal width of text on a line.
- `...`: ignored.

**Details**

Both, `depase_usage1` and the `as.character` method for class "f_usage", convert an "f_usage" object to a character string suitable for Rd documentation. The `as.character` method is the user level function (it just calls `depase_usage1`). `depase_usage1` is internal function for programming. In the example below the first command creates an "f_usage" object, then the second converts it to character string.

```r
(a <- pairlist2f_usage1(formals(cor), "cor"))
#> name = cor
#> S3class =
#> S4sig =
#> infix = FALSE
#> fu = TRUE
#> argnames = x y use method
#> defaults : y = NULL
#> use = "everything"
#> method = c("pearson", "kendall", "spearman")

cat(as.character(a))
#> cor(x, y = NULL, use = "everything",
#>     method = c("pearson", "kendall", "spearman"))
```

Each usage entry is formatted and, if necessary, split over several lines. The width (number of characters) on a line can be changed with argument `width`.

`depase_usage` can be used when `x` is a list of "f_usage" objects. It calls `depase_usage1` with each of them and returns a character vector with one element for each component of `x`. When `x` is an object from class "f_usage", `depase_usage` is equivalent to `depase_usage1`.

**Value**

For `depase_usage1` and `as.character.f_usage`, a named character vector of length one (the name is the function name).

For `depase_usage`, a named character vector with one entry for the usage text for each function.
ereprompt

**Description**

Update an Rd file and open it in an editor. This is a wrapper for `reprompt` with different defaults for some parameters.

**Usage**

```r
ereprompt(..., edit = TRUE, filename = TRUE)
```
**format_funusage**

**Arguments**

- ... passed on to `reprompt`, see its documentation for details.
- edit if TRUE, the default, open an editor when finished.
- filename if TRUE, the default, replace and/or edit the original Rd file.

**Details**

erprompt calls reprompt to do the actual job but has different defaults for the arguments described on this page. By default, it replaces the original Rd file with the updated documentation and opens it in an editor.

**Value**

called for the side effect of updating Rd documentation file and opening it in an editor

**Author(s)**

Georgi N. Boshnakov

**See Also**

`reprompt` which does the actual work

**Examples**

```r
## this assumes that the current working directory is
## in any subdirectory of the development directory of Rdpack
## Not run:
ereprompt(infile = "reprompt.Rd")

## End(Not run)
```

---

**Description**

Formats the usage text of a function so that each line contains no more than a given number of characters.

**Usage**

`format_funusage(x, name = "", width = 72, realname)`
Arguments

- **x**: a character vector containing one element for each argument of the function, see 'Details'.
- **name**: the name of the function whose usage is described, a string.
- **width**: maximal width of each line of output.
- **realname**: the printed form of name, see ‘Details’, a string.

Details

format_funusage formats the usage text of a function for inclusion in Rd documentation files. If necessary, it splits the text into more lines in order to fit it within the requested width.

Each element of argument `x` contains the text for one argument of function `name` in the form `arg` or `arg = default`. format_funusage does not look into the content of `x`, it does the necessary pasting to form the complete usage text, inserting new lines and indentation to stay within the specified width. Elements of `x` are never split. If an argument (i.e., element of `x`) would cause the width to be exceeded, the entire argument is moved to the following line.

The text on the second and subsequent lines of each usage item starts in the column just after the opening parenthesis which follows the name of the function on the first line.

In descriptions of S3 methods and S4 methods, argument `name` may be a TeX macro like \method{print}{ts}. In that case the number of characters in `name` has little bearing on the actual number printed. In this case argument `realname` is used for counting both the number of characters on the first line of the usage message and the indentation for the subsequent lines.

Value

The formatted text as a length one character vector.

Note

Only the width of `realname` is used (for counting). The formatted text contains `name`. The width of strings is determined by calling `nchar` with argument `type` set to "width".

Author(s)

Georgi N. Boshnakov

See Also

- deparse_usage1

Examples

# this function is essentially internal,
# see deparse_usage1 and as.character.f_usage which use it.
get_bibentries

Get all references from a Bibtex file

Description

Get all references from a Bibtex file.

Usage

get_bibentries(..., package = NULL, bibfile = "REFERENCES.bib",
               url_only = FALSE, stop_on_error = TRUE)

Arguments

... arguments to be passed on to the file getting functions, character strings, see
     'Details'.
package name of a package, a character string or NULL.
bibfile name of a Bibtex file, a character string.
url_only if TRUE, restrict percent escaping to BibTeX field "URL".
stop_on_error if TRUE stop on error, otherwise issue a warning and return an empty bibentryRd
     object.

Details

get_bibentries parses the specified file.

and sets its names attribute to the keys of the bib elements (read.bib() does this since version
version 0.4.0 of bibtex, as well). Here is what get_bibentries does on top of read.bib (the
details are further below):

- get_bibentries deals with percent signs in URL's.
- if the current working directory is in the development directory of package, get_bibentries
  will first search for the bib file under that directory.

bibfile should normally be the base name of the Bibtex file. Calling get_bibentries without
any "\dots" arguments results in looking for the Bibtex file in the current directory if package is
NULL or missing, and in the installation directory of the specified package, otherwise. Argument
"..." may be used to specify directories. If package is missing or NULL, the complete path is
obtained with file.path(...,bibfile). Otherwise package must be a package name and the
file is taken from the installation directory of the package. Again, argument "..." can specify
subdirectory as in system.file.

If the current working directory is in the development directory of package, the bib file is first
sought there before resorting to the installation directory.

Although the base R packages do not have files REFERENCES.bib, argument package can be set to
one of them, e.g. "base". This works since package bibtex provides bib files for the core packages.
By default, `get_bibentries` escapes unescaped percent signs in all fields of bibtex objects. To restrict this only to field "url", set argument `url_only` to `FALSE`.

`get_bibentries` returns an object from class "bibentryRd", which inherits from bibentry. The printing method for "bibentryRd" unescapes percent signs in URLs for some styles where the escapes are undesirable.

**Value**

a `bibentryRd` object inheriting from `bibentry`

**Author(s)**

Georgi N. Boshnakov

**References**

There are no references for Rd macro \insertAllCites on this help page.

**Examples**

```r
r <- get_bibentries(package = "Rdpack")
r
print(r, style = "html")

## Bib from base R packages are disabled in Rdpack v2 (notify the
## maintainer of Rdpack or raise an issue on github if you wish this back).
##
## b <- get_bibentries(package = "stats")
## print(b[[1]], style = "R")
## print(b[[1]], style = "citation")

## here the url field contains percent encoding
fn_url <- system.file("examples", "url_with_percents.bib", package = "Rdpack")
u <- get_bibentries(bibfile = fn_url)

## the links produced by all of the following are valid
## and can be put in a browser
print(u, style = "html")
print(u, style = "bibtex")
print(u, style = "R")
print(u, style = "text")
print(u, style = "citation")

## The link here contains escapes but when put in a LaTeX document
## which uses the JSS style it generates correct clickable link,
## (see Details section)
print(u, style = "latex")

## here the journal field contains percent encoding
fn_other <- system.file("examples", "journal_with_percents.bib", package = "Rdpack")
j <- get_bibentries(bibfile = fn_url)
print(j, style = "html")
```
get_sig_text

Produce the textual form of the signatures of available methods for an S4 generic function.

Usage

get_sig_text(rdo, package = NULL)

Arguments

rdo  
an Rd object.

package  
if of class "character", give only methods defined by package, otherwise give all methods.

Details

Signatures are found using function findMethodSignatures from package "methods".

Here we find all methods for show() defined in package "methods" and print the first few of them:

fn <- utils::help("show-methods", package = "methods")
rdo <- utils:::.getHelpFile(fn)
head(get_sig_text(rdo))
##: [1] "signature(object = "ANY")"
##: [2] "signature(object = "MethodDefinition")"
##: [3] "signature(object = "MethodDefinitionWithTrace")"
##: [5] "signature(object = "MethodWithNext")"
##: [6] "signature(object = "MethodWithNextWithTrace")"

Value

A character vector with one element for each method.
Note

todo: It would be better to call promptMethods() to get the signatures but in version R-2.13.x I had trouble with argument ‘where’ (could not figure out how to use it to restrict to functions from a package; also, promptMethods() seemed to call the deprecated function getMethods()). Check how these things stand in current versions of R, there may be no problem any more (checked, in 2.14-0 it is the same).

Author(s)

Georgi N. Boshnakov

Examples

```r
## load another package with some S4 methods ("methods" is already loaded)
require("stats4")

rdo <- Rdo_fetch("show", package = "methods")
## alternatively:
#fn <- help("show-methods", package = "methods")
#rdo <- utils:::.getHelpFile(fn)

## this will find all methods for "show" in currently loaded packages
## (print only some of them)
head(get_sig_text(rdo))

## this will select only the ones from package "stats4"
get_sig_text(rdo, package = "stats4")

## this is also fine (interactively) but need to choose
## the appropriate element of "fn" if length(fn) > 1
#fn <- help("show-methods")

## this finds nothing
#fn <- help("logLik-methods", package = "methods")
#fn
Rdo_fetch("logLik-methods", package = "methods")

## this does
#fn <- help("logLik-methods", package = "stats4")
#rdo <- utils:::.getHelpFile(fn)
rdo2 <- Rdo_fetch("logLik-methods", package = "stats4")

get_sig_text(rdo2)
get_sig_text(rdo2, package = "stats4")

## only default method defined
## using this:
setGeneric("f1", function(x, y){NULL})
## since the following gives error in pkgdown:
#f1 <- function(x, y){NULL}
#setGeneric("f1")
```
get_usage_text

Get the text of the usage section of Rd documentation

Description
Get the text of the usage section of Rd documentation.

Usage
get_usage_text(rdo)

Arguments
rdo an Rd object or a character string

Details
If rdo is a string, it is parsed to obtain an Rd object.
The content of section "usage" is extracted and converted to string.

Value
a string
Note

todo: get_usage_text can be generalised to any Rd section but it is better to use a different approach since print.Rd() does not take care for some details (escaping %, for example). Also, the functions that use this one assume that it returns R code, which may not be the case if the usage section contains Rd comments.

Author(s)

Georgi N. Boshnakov

Examples

```r
## get the Rd object documenting Rdo_macro
#h <- utils::help("Rdo_macro", lib.loc = .libPaths())
#rdo <- utils:::.getHelpFile(h)
rdo <- Rdo_fetch("Rdo_macro", "Rdpack")
# extract the usage section and print it:
ut <- get_usage_text(rdo)
cat(ut, sep = "\n")
```

---

**insert_all_ref**  
*Insert references cited in packages*

Description

Insert references cited in packages.

Usage

```r
insert_all_ref(refs, style = "")
```

Arguments

- **refs**: a matrix specifying key-package pairs of the references to insert. Can also be a cached environment, see Details.
- **style**: a bibstyle, see Details.

Details

insert_all_ref is the workhorse behind several Rd macros for inclusion of references in Rd documentation.

Argument **refs** is a two-column character matrix. The first column specifies bibtex keys. To specify more than one key in a single element, separate them by commas. The second column specifies the package in which to look for the keys. A key equal to "*" requests all keys in the corresponding package.

insert_all_ref drops duplicated keys, collects the references, and converts them to Rd textual representation for inclusion in Rd documentation files.

refs can be a cached environment. This is for internal use and not documented.
Value

for `insert_all_ref`, a character string containing a textual representation of the references, suitable for inclusion in an Rd file.

Author(s)

Georgi N. Boshnakov

References

Currently there are no citations. Nevertheless, I have put `\insertAllCited{}` just after this paragraph to show the message that it prints when there are no citations. This seems better than printing nothing but it may be argued also that there should be a warning as well.

There are no references for Rd macro `\insertAllCites` on this help page.

Examples

```r
## a reference from package Rdpack
cat(insert_all_ref(matrix(c("Rpack:bibtex", "Rdpack"), ncol = 2)), "\n")

## more than one reference from package Rdpack, separate the keys with commas
cat(insert_all_ref(matrix(c("parseRd,Rpack:bibtex", "Rdpack"), ncol = 2)), "\n")

## all references from package Rdpack
cat(insert_all_ref(matrix(c("*", "Rdpack"), ncol = 2)), "\n")

## all references from package Rdpack and rbibutils
m <- matrix(c("*", "Rdpack", "*", "rbibutils"), ncol = 2, byrow = TRUE)
cat(insert_all_ref(m), "\n")
```

---

**insert_citeOnly**  
*Generate citations from bibtex files*

Description

Generate citations from bibtex files.

Usage

```r
insert_citeOnly(keys, package = NULL, before = NULL, after = NULL, 
bibpunct = NULL, ..., cached_env = NULL, 
cite_only = FALSE, dont_cite = FALSE)
```
Arguments

- `keys`: bibtex key(s), a character vector.
- `package`: name of an R package.
- `before`: see `citeNatbib`.
- `after`: see `citeNatbib`.
- `bibpunct`: see `citeNatbib`.
- `...`: further arguments.
- `cached_env`: for internal use.
- `cite_only`: for internal use.
- `dont_cite`: for internal use.

Details

This is the function behind `\insertCite` and related macros.

Value

A character vector containing the references with Rd markup.

Author(s)

Georgi N. Boshnakov

See Also

- `insert_ref` for description of all available Rd macros

Examples

```r
insert_citeOnly("@see also @Rpackage:rbibutils and @parseRd", package = "Rdpack")
## (see also Boshnakov and Putman 2020 and Murdoch 2010)

insert_citeOnly("@see also @Rpackage:rbibutils and @parseRd;nobrackets",
               package = "Rdpack")
## see also Boshnakov and Putman 2020 and Murdoch 2010
```

Description

Package Rdpack provides Rd macros for inserting references and citations from bibtex files into R documentation. Function `insert_ref()` is the workhorse behind this mechanism. The description given on this page should be sufficient, for more details see the vignette.
Usage

```r
insert_ref(key, package = NULL, ..., cached_env = NULL)
```

Arguments

- **key**: the bibtex key of the reference, a character string.
- **package**: the package in which to look for the bibtex file.
- **...**: further arguments to pass on to `get_bibentries()`. The only one of interest to users is `bibfile`, whose default is "REFERENCES.bib", see `get_bibentries`.
- **cached_env**: environment, used to avoid repeatedly parsing the bib file from scratch, mainly useful by the Rd macros.

Details

`insert_ref` extracts a bibliographic reference from a bibtex file, converts it to Rd format and returns a single string with embedded newline characters. It is the workhorse in the provided mechanism but most users do not even need to know about `insert_ref`.

Package `Rdpack` provides several Rd macros for inserting references and citations in the documentation of a package. All macros work in both, manually written Rd files and in `roxygen2` documentation chunks. The macros look for references in file "REFERENCES.bib" in the root of the installation directory of the package. When a package is in development mode under `devtools`, "inst/REFERENCES.bib" in the development directory of the package is checked first. See also `get_bibentries` but note that while direct calls to `insert_ref()` can specify a different file, the filename and the places where it is looked for are fixed for the Rd macros.

The description below assumes that `Rdpack` has been added to file DESCRIPTION, as described in `Rdpack-package` and vignette "Inserting bibtex references". The Rd macros are put in the text of documentation sources (‘Rd’ files or ‘roxygen2’ chunks).

**Rd macro insertRef:**

```r
\insertRef{key}{package}
```

Inserts the reference with bibtex key from file "REFERENCES.bib" in package package. Argument `package` can be any installed R package, not necessarily the one of the documentation object, as long as it contains file "REFERENCES.bib" in the root of its installation directory. The references are partially processed when the package is built.

References inserted with `\insertRef` appear in the place where the macro is put, usually in a dedicated references section (\references in Rd files, @references in roxygen chunks).

For example, section ‘References’ of this help page shows (among other things) the rendered results of the following lines in the Rd source file:

```r
\insertRef{Rpackage:rbibutils}{Rdpack}
\insertRef{parseRd}{Rdpack}
\insertRef{bibutils6.10}{rbibutils}
```

A roxygen2 documentation chunk might look like this:
The first reference has label Rpackage:rbibutils and is taken from file "REFERENCES.bib" in package Rdpack. The third reference is from the file "REFERENCES.bib" in package rbibutils.

For more details see vignette "Inserting_bibtex_references".

**Rd macro insertCite:**

\insertCite{key}{package} cites the key and records it for use by \insertAllCited, see below. key can contain more keys separated by commas.

\insertCite{parseRd,Rpackage:rbibutils}{Rdpack} (Murdoch 2010; Boshnakov and Putman 2020)
\insertCite{Rpackage:rbibutils}{Rdpack} (Boshnakov and Putman 2020)
\insertCite{bibutils6.10}{rbibutils} (Putnam 2020)

By default the citations are parenthesised (Murdoch 2010). To get textual citations, like Murdoch (2010), put the string ;textual at the end of the key. The references in the last two sentences were produced with \insertCite{parseRd}{Rdpack} and \insertCite{parseRd; textual}{Rdpack}, respectively. This also works with several citations, e.g. \insertCite{parseRd,Rpackage:rbibutils; textual}{Rdpack} produces: Murdoch (2010); Boshnakov and Putman (2020).

To mix the citations with other text, such as ‘see also’ and ‘chapter 3’, write the list of keys as free text, starting it with the symbol @ and prefixing each key with it. The @ symbol will not appear in the output. For example, the following code

\insertCite{@see also @parseRd and @Rpackage:rbibutils}{Rdpack},
\insertCite{@see also @parseRd; @Rpackage:rbibutils}{Rdpack},
\insertCite{@see also @parseRd and @Rpackage:rbibutils; textual}{Rdpack}.

produces:

(see also Murdoch 2010 and Boshnakov and Putman 2020),
(see also Murdoch 2010; Boshnakov and Putman 2020),
see also Murdoch (2010) and Boshnakov and Putman (2020).

In the parenthesised form, adding ;nobrackets at the end of the list of keys causes the enclosing parentheses to be dropped. This is useful if you wish to use markup for the text surrounding the references. For example,

\[\texttt{\textbackslash emph\{see also\} \textbackslash insertCite{@@parseRd; @Rpackage:rbibutils; nobrackets}{Rdpack}}\]

gives: (see also Murdoch 2010; Boshnakov and Putman 2020). Without ‘;nobrackets’ the result
would be (see also (Murdoch 2010; Boshnakov and Putman 2020)).

**Rd macro insertNoCite:**
The macro \insertNoCite{key}{package} records one or more references for \insertAllCited but does not cite it. Setting key to * will include all references from the specified package. For example, \insertNoCite{parseRd}{Rdpack} and \insertNoCite{*}{rbibutils} record the specified references for inclusion by \insertAllCited.

**Rd macro insertAllCited:**
\insertAllCited inserts all references cited with \insertCite and \insertNoCite. Putting this macro in the references section will keep the references up to date automatically. The Rd section may look something like:

```latex
\references{
  \insertAllCited{}
}
```

or in roxygen2, the references chunk might look like this:

```latex
# @references
# \insertAllCited{}
```

**Rd macro insertNoCite:**
\insertCiteOnly{key}{package} is as \insertCite but does not include the key in the list of references for \insertAllCited.

**Value**

for insert_ref, a character string

**Author(s)**

Georgi N. Boshnakov

**References**

For illustrative purposes there are two sets of citations below The first set of references is obtained with \insertRef for each reference:


The following references are obtained with a single \insertAllCited{}. The references are sorted automatically by the surnames of the authors:
inspect_args

Inspect the argument section of an Rd object

Description

Inspect the argument section of an Rd object.

Usage

inspect_args(rdo, i_usage)

Arguments

rdo an Rd object describing functions.

i_usage see Details.

Details

inspect_args checks if the arguments in the documentation object rdo match the (union of) the actual arguments of the functions it describes.

If i_usage is missing, it is computed by inspecting the current definitions of the functions described in rdo, see inspect_usage. This argument is likely to be supplied if the function calling inspect_args has already computed it for other purposes.
**Value**

TRUE if the arguments in the documentation match the (union of) the actual arguments of the described functions, FALSE otherwise.

The returned logical value has attribute ‘details’ which is a list with the following components.

- `rdo_argnames`: arguments described in the documentation object, rdo.
- `cur_argnames`: arguments in the current definitions of the described functions.
- `added_argnames`: new arguments
- `removed_argnames`: removed (dropped) arguments.

**Author(s)**

Georgi N. Boshnakov

---

**inspect_Rd**

*Inspect and update an Rd object or file*

**Description**

Inspect and update an Rd object or file.

**Usage**

```
inspect_Rd(rdo, package = NULL)
inspect_Rdfun(rdo, alias_update = TRUE)
inspect_Rdmethods(rdo, package = NULL)
inspect_Rdclass(rdo)
```

**Arguments**

- `rdo`: an Rd object or file name
- `package`: name of a package
- `alias_update`: if TRUE, add missing alias entries for functions with usage statements.
Details

These functions check if the descriptions of the objects in rdo are consistent with their current definitions and update them, if necessary. The details depend on the type of the documented topic. In general, the functions update entries that can be produced programmatically, possibly accompanied with a suggestion to the author to write some additional text.

inspect_Rd checks the 'name section of rdo and dispatches to one of the other inspect_XXX functions depending on the type of the topic.

inspect_Rdfunc processes documentation of functions. It checks the usage entries of all functions documented in rdo and updates them if necessary. It appends "alias" entries for functions that do not have them. Entries are created for any arguments that are missing from the "arguments" section. Warning is given for arguments in the "arguments" section that are not present in at least one usage entry. inspect_Rdfunc understands the syntax for S3 methods and S4 methods used in "usage" sections, as well. The S4 methods may also be in a section as produced by promptMethods.

inspect_Rdmethods checks and updates documentation of an S4 generic function.

inspect_Rdclass checks and updates documentation of an S4 class.

Since method signatures and descriptions may be present in documentation of a class, as well as in that of methods, the question arises where to put "alias" entries to avoid duplication. Currently, alias entries are put in method descriptions.

Author(s)

Georgi N. Boshnakov

inspect_signatures
Inspect signatures of S4 methods

Description
Inspect signatures of S4 methods.

Usage

inspect_clmethods(rdo, final = TRUE)

inspect_signatures(rdo, package = NULL, sec = "Methods")

Arguments

rdo an Rd object.
package the name of a package, a character string or NULL.
sec the name of a section to look into, a character string.
final If not TRUE insert text with suggestions, otherwise comment the suggestions out.
Details

Signatures in documentation of S4 classes and methods are stored somewhat differently. `inspect_signatures` inspects signatures in documentation of methods of a function. `inspect_clmethods` inspects signatures in documentation of a class.

`inspect_signatures` was written before `inspect_clmethods()` and was geared towards using existing code for ordinary functions (mainly `parse_usage_text()`).

If new methods are found, the functions add entries for them in the Rd object `rdo`.

If `rdo` documents methods that do not exist, a message inviting the user to remove them manually is printed but the offending entries remain in the object. At the time of writing, R CMD check does not warn about this.

Value

an Rd object

Note

todo: need consolidation.

Author(s)

Georgi N. Boshnakov

---

**inspect_slots**

*Inspect the slots of an S4 class*

Description

Inspect the slots of an S4 class.

Usage

`inspect_slots(rdo, final = TRUE)`

Arguments

- `rdo` an Rd object.
- `final` if not TRUE insert text with suggestions, otherwise comment the suggestions out.

Author(s)

Georgi N. Boshnakov
inspect_usage

Inspect the usage section in an Rd object

Description

Inspect the usage section in an Rd object.

Usage

inspect_usage(rdo)

Arguments

rdo an Rd object.

Details

The usage section in the Rd object, rdo, is extracted and parsed. The usage of each function described in rdo is obtained also from the actual installed function and compared to the one from rdo.

The return value is a list, with one element for each function usage as returned by compare_usage1. One of the consequences of this is that an easy way to add a usage description of a function, say fu to an existing Rd file is to simply add a line fu() to the usage section of that file and run reprompt on it.

Value

a list of comparison results as described in ‘Details’ (todo: give more details here)

Author(s)

Georgi N. Boshnakov

See Also

inspect_args
list_Rd

Description

Combine Rd fragments and strings into one object.

Usage

list_Rd(..., Rd_tag = NULL, Rd_class = FALSE)

Arguments

... named list of objects to combine, see ‘Details’.
Rd_tag if non-null, a value for the Rd_tag of the result.
Rd_class logical; if TRUE, the result will be of class "Rd".

Details

The names of named arguments specify tags for the corresponding elements (not arbitrary tags, ones that are converted to macro names by prepending backslash to them). This is a convenient way to specify sections, items, etc, in cases when the arguments have not being tagged by previous processing. Character string arguments are converted to the appropriate Rd pieces. Argument ... may contain a mixture of character vectors and Rd pieces.

Value

an Rd object or list with Rd_tag attribute, as specified by the arguments.

Author(s)

Georgi N. Boshnakov

See Also

c_Rd

Examples

## see also the examples for c_Rd
dummyfun <- function(x, ...) x

u1 <- list_Rd(name = "Dummyname", alias = "dummyfun",
title = "Dummy title", description = "Dummy description",
usage = "dummyfun(x)",
value = "numeric vector",
author = "A. Author",

makeVignetteReference

Make bibtex references for vignettes

Description

Make bibtex references for vignettes

Usage

makeVignetteReference(package, vig = 1, verbose = TRUE, title, author,
  type = "pdf", bibtype = "Article", key = NULL)

vigbib(package, verbose = TRUE, ..., vig = NULL)

Arguments

package a character string, the name of an installed package.

vig an integer number or a character string identifying a vignette.

verbose if TRUE, print the references in Bibtex format.

title a character string, title of the vignette, see Details.

author a character string, title of the vignette, see Details.

type a character string, type of the vignette, such as "pdf" or "html". Currently ignored.

bibtype a character string, Bibtex type for the reference, defaults to "Article".

key a character string specifying a key for the Bibtex entry. If missing, suitable key is generated automatically.

... arguments passed by vigbib() to makeVignetteReference().
makeVignetteReference

Details

vigbib() generates Bibtex references for all vignettes in a package. makeVignetteReference() produces a BibTex entry for a particular vignette.

There seems to be no standard way to cite vignettes in R packages. For vignettes that are complete journal papers (most notably in the Journal of Statistical Software), the authors would usually prefer the papers to be cited, rather than the vignette. In any case, consulting the output of citation("a_package") is the best starting point. If the vignette has been extended substantially after the paper was published, cite both.

In many cases it is sufficient to give the command that opens the vignette, e.g.:

vignette("Inserting_bibtex_references", package = "Rdpack").

makeVignetteReference() makes a BibTex entry for one vignette. It looks for the available vignettes using vignette(package=package). Argument vig can be a character string identifying the vignette by the name that would be used in a call to vignette(). It can also be an integer, identifying the vignette by the index (in the order in which the vignettes are returned by vignette()). By default the first vignette is returned. If vig is not suitable, a suitable list of alternatives is printed.

For vigbib() it is sufficient to give the name of a package. It accepts all arguments of makeVignetteReference() except vig (actually, supplying vig is equivalent to calling makeVignetteReference() directly).

The remaining arguments can be used to overwrite some automatically generated entries. For example, the vignette authors may not be the same as the package authors.

Value

a bibentry object containing the generated references (the BibTex entries are also printed, so that they can be copied to a bib file)

Author(s)

Georgi N. Boshnakov

Examples

## NOTE (2020-01-21): the following examples work fine, but are not
## rendered correctly by pkgdown::build_site(), so there may be errors
## on the site produced by it, https://geobosh.github.io/Rdpack/.

vigbib("Rdpack")
makeVignetteReference("Rdpack", vig = 1)
makeVignetteReference("Rdpack", vig = "Inserting_bibtex_references")
## the first few characters of the name suffice:
makeVignetteReference("Rdpack", vig = "Inserting_bib")

## this gives an error but also prints the available vignettes:  
## makeVignetteReference("Matrix", vig = "NoSuchVignette")

vigbib("utils")
makeVignetteReference("utils", vig = 1)
## commented out since can be slow:
## high<-installed.packages(priority = "high")
## highbib<-lapply(rownames(high), function(x) try(Rdpack:::vigbib(x, verbose = FALSE)))
parse_pairlist

Parse formal arguments of functions

Description
Parse formal arguments of functions and convert them to f_usage objects.

Usage
parse_pairlist(x)

pairlist2f_usage1(x, name, S3class = "", S4sig = "", infix = FALSE,
                  fu = TRUE)

Arguments
x       a pairlist representing the arguments of a function.
name    function name.
S3class S3 class, see ‘Details’
S4sig   S4 signature, see Details.
infix   if TRUE the function usage is in infix form, see Details.
fu      if TRUE the object is a function, otherwise it is something else (e.g. a variable
          or a constant like pi and Inf).

Details
These functions are mostly internal.
parse_pairlist parses the pairlist object, x, into a list with two components. The first component
contains the names of the arguments. The second component is a named list containing the default
values, converted to strings. Only arguments with default values have entries in the second compo-
nent (so, it may be of length zero). If x is empty or NULL, both components have length zero. An
example:

parse_pairlist(formals(system.file))
##: $argnames
##: [1] "..." "package" "lib.loc" "mustWork"

##: $defaults
##: package   lib.loc   mustWork
##: "\"base\"" "NULL" "FALSE"

pairlist2f_usage1() creates an object of S3 class "f_usage". The object contains the result of
parsing x with parse_pairlist(x) and a number of additional components which are copies of
the remaining arguments to the function (without any processing). The components are listed in
section Values. S3class is set to an S3 class for for S3 methods, S4sig is the signature of an S4 method (as used in Rd macro \S4method). infix is TRUE for the rare occasions when the function is primarily used in infix form.

Class "f_usage" has a method for as.character() which generates a text suitable for inclusion in Rd documentation.

pairlist2f_usage1(formals(summary.lm), "summary", S3class = "lm")
### name = summary
### S3class = lm
### S4sig =
### infix = FALSE
### fu = TRUE
### argnames = object correlation symbolic.cor ...
### defaults : correlation = FALSE
### symbolic.cor = FALSE

Value

For parse_pairlist, a list with the following components:

argnames the names of all arguments, a character vector
defaults a named character vector containing the default values, converted to character strings. Only arguments with defaults have entries in this vector.

For pairlist2f_usage1, an object with S3 class "f_usage". This is a list as for parse_pairlist and the following additional components:

name function name, a character string.
S3class S3 class, a character string; "" if not an S3 method.
S4sig S4 signature; "" if not an S4 method.
infix a logical value, TRUE for infix operators.
fu indicates the type of the object, usually TRUE, see Details.

Author(s)

Georgi N. Boshnakov

See Also

promptUsage accepts f_usage objects, get_usage

Examples

parse_pairlist(formals(lm))
parse_pairlist(formals(system.file))
s_lm <- pairlist2f_usage1(formals(summary.lm), "summary", S3class = "lm")
s_lm
as.character(s_lm)
**parse_Rdname**

*Parse the name section of an Rd object*

**Description**

Parse the name section of an Rd object.

**Usage**

```r
call parse_Rdname(rdo)
```

**Arguments**

- `rdo` an Rd object

**Details**

The content of section "\name" is extracted. If it contains a hyphen, '-', the part before the hyphen is taken to be the topic (usually a function name), while the part after the hyphen is the type. If the name does not contain hyphens, the type is set to the empty string.

**Value**

A list with two components:

- `fname` name of the topic, usually a function
- `type` type of the topic, such as "method"

**Author(s)**

Georgi N. Boshnakov

**Examples**

```r
test <- list_Rd(name = "Dummyname", alias = "Dummyallias1",
               title = "Dummy title", description = "Dummy description",
               Rd_class=TRUE )

parse_Rdname(test)

test2 <- list_Rd(name = "dummyclass-class", alias = "Dummyclass",
               title = "Class dummyclass",
               description = "Objects and methods for something.",
               Rd_class=TRUE )

parse_Rdname(test2)
```
parse_Rdpiece

Parse a piece of Rd source text

Description

Parse a piece of Rd source text.

Usage

parse_Rdpiece(x, result = "")

Arguments

x the piece of Rd text, a character vector.

result if "text", converts the result to printable text (e.g. to be shown to the user), otherwise returns an Rd object.

Details

parse_Rdpiece parses a piece of source Rd text. The text may be an almost arbitrary piece that may be inserted in an Rd source file, except that it should not be a top level section (use parse_Rdtext for sections). Todo: it probably can be also a parsed piece, check!

This is somewhat tricky since parse_Rd does not accept arbitrary piece of Rd text. It handles either a complete Rd source or a fragment, defined (as I understand it) as a top level section. To circumvent this limitation, this function constructs a minimal complete Rd source putting argument x in a section (currently "Note") which does not have special formatting on its own. After parsing, it extracts only the part corresponding to x.

parse_Rdpiece by default returns the parsed Rd piece. However, if result="text", then the text is formatted as the help system would do when presenting help pages in text format.

TODO: add an argument for macros?

Value

a parsed Rd piece or its textual representation as described in Details

Author(s)

Georgi N. Boshnakov

Examples

# the following creates Rd object rdo
dummyfun <- function(x) x
u1 <- list_Rd(name = "Dummyname", alias = "dummyfun",
title = "Dummy title", description = "Dummy description",
usage = "dummyfun(x,y)",
value = "numeric vector",
}


parse_Rdtext

Parse Rd source text as the contents of a section

Description
Parse Rd source text as the contents of a given section.

Usage
parse_Rdtext(text, section = NA)

Arguments
- text: Rd source text, a character vector.
- section: the section name, a string.

Details
If section is given, then parse_Rdtext parses text as appropriate for the content of section. This is achieved by inserting text as an argument to the TeX macro section. For example, if section is "usage", then a line \usage{\dots} is inserted at the beginning of text and a closing "\end{\usage}" at its end.

If section is NA then parse_Rdtext parses it without preprocessing. In this case text itself will normally be a complete section fragment.
Value

an Rd fragment

Note

The text is saved to a temporary file and parsed using parse_Rd. This is done for at least two reasons. Firstly, parse_Rd works most reliably (at the time of writing this) from a file. Secondly, the saved file may be slightly different (escaped backslashes being the primary example). It would be a nightmare to ensure that all concerned functions know if some Rd text is read from a file or not.

The (currently internal) function .parse_Rdlines takes a character vector, writes it to a file (using cat) and calls parse_Rd to parse it.

Author(s)

Georgi N. Boshnakov

See Also

parse_Rdpiece

Description

Parse usage text.

Usage

parse_1usage_text(text)
parse_usage_text(text)

Arguments

text conceptually, the content of the usage section of one or more Rd objects, a character vector, see Details.

Details

For parse_usage_text, text is a character vector representing the contents of the usage section of an Rd object. parse_usage_text does some preprocessing of text then calls parse_1usage_text for each usage statement.

The preprocessing changes "\dots" to "..." and converts S3- and S4-method descriptions to a form suitable for parse(). The text is then parsed (with parse) and "srcref" attribute removed from the parsed object.

todo: currently no checks is made for Rd comments in text.
parse_1usage_text processes the usage statement of one object and calls `pairlist2f_usage1` to convert it to an object from S3 class "f_usage".

**Value**

for `parse_1usage_text`, an object from S3 class "f_usage", see `pairlist2f_usage1` for its structure.

for `parse_usage_text`, a list containing one element for each usage entry, as prepared by `parse_1usage_text`

**Author(s)**

Georgi N. Boshnakov

<table>
<thead>
<tr>
<th>predefined</th>
<th>Tables of predefined sections and types of pieces of Rd objects</th>
</tr>
</thead>
</table>

**Description**

Tables of predefined sections and types of pieces of Rd objects.

**Usage**

Rdo_predefined_sections

Rdo_piece_types

rdo_top_tags

**Details**

The Rd syntax defines several tables (Murdoch 2010). Rdpack stores them in the variables described here.

Rdo_predefined_sections is a named character vector of types of the top level sections of an Rd object.

Rdo_piece_types is a named character vector giving the types of the core (all possible?) Rd macros.

**NOTE:** These objects are hard coded and need to be updated if the specifications of the Rd format are updated.

todo: write functions that go through existing Rd documentation to discover missing or wrong items.
Value

For Rdo_predefined_sections, the name-value pairs are given in the following table. For example, \code{Rdo_predefined_sections["examples"]} is RCODE.

<table>
<thead>
<tr>
<th>name</th>
<th>verb</th>
<th>description</th>
<th>verb</th>
<th>TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>verb</td>
<td>examples</td>
<td>verb</td>
<td>RCODE</td>
</tr>
<tr>
<td>concept</td>
<td>text</td>
<td>usage</td>
<td>verb</td>
<td>RCODE</td>
</tr>
<tr>
<td>docType</td>
<td>text</td>
<td>Rdversion</td>
<td>verb</td>
<td>VERB</td>
</tr>
<tr>
<td>title</td>
<td>text</td>
<td>synopsis</td>
<td>verb</td>
<td>VERB</td>
</tr>
<tr>
<td>name</td>
<td>verb</td>
<td>section</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>alias</td>
<td>verb</td>
<td>arguments</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>concept</td>
<td>text</td>
<td>keyword</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>docType</td>
<td>text</td>
<td>note</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>title</td>
<td>text</td>
<td>format</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>verb</td>
<td>source</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>alias</td>
<td>verb</td>
<td>details</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>concept</td>
<td>text</td>
<td>value</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>docType</td>
<td>text</td>
<td>references</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>title</td>
<td>text</td>
<td>author</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>verb</td>
<td>seealso</td>
<td>text</td>
<td></td>
</tr>
</tbody>
</table>

For Rdo_piece_types, the name-value pairs are:

<table>
<thead>
<tr>
<th>name</th>
<th>verb</th>
<th>alias</th>
<th>verb</th>
<th>concept</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>docType</td>
<td>text</td>
<td>title</td>
<td>text</td>
<td>description</td>
<td>text</td>
</tr>
<tr>
<td>examples</td>
<td>RCODE</td>
<td>usage</td>
<td>RCODE</td>
<td>Rdversion</td>
<td>VERB</td>
</tr>
<tr>
<td>synopsis</td>
<td>VERB</td>
<td>Sexpr</td>
<td>RCODE</td>
<td>RdOpts</td>
<td>VERB</td>
</tr>
<tr>
<td>code</td>
<td>RCODE</td>
<td>dontshow</td>
<td>RCODE</td>
<td>donttest</td>
<td>RCODE</td>
</tr>
<tr>
<td>testonly</td>
<td>RCODE</td>
<td>dontrun</td>
<td>VERB</td>
<td>env</td>
<td>VERB</td>
</tr>
<tr>
<td>kbd</td>
<td>VERB</td>
<td>option</td>
<td>VERB</td>
<td>out</td>
<td>VERB</td>
</tr>
<tr>
<td>preformatted</td>
<td>VERB</td>
<td>samp</td>
<td>VERB</td>
<td>special</td>
<td>VERB</td>
</tr>
<tr>
<td>url</td>
<td>VERB</td>
<td>verb</td>
<td>VERB</td>
<td>deqn</td>
<td>VERB</td>
</tr>
<tr>
<td>eqn</td>
<td>VERB</td>
<td>renewcommand</td>
<td>VERB</td>
<td>newcommand</td>
<td>VERB</td>
</tr>
</tbody>
</table>

For rdo_top_tags, the values are:

<table>
<thead>
<tr>
<th>\name</th>
<th>\Rdversion</th>
<th>\docType</th>
<th>\alias</th>
<th>\encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>\concept</td>
<td>\title</td>
<td>\description</td>
<td>usage</td>
<td>\format</td>
</tr>
<tr>
<td>\source</td>
<td>\arguments</td>
<td>\details</td>
<td>\value</td>
<td>\references</td>
</tr>
<tr>
<td>\section</td>
<td>\note</td>
<td>\author</td>
<td>\seealso</td>
<td>\examples</td>
</tr>
<tr>
<td>\keyword</td>
<td>\ifdef</td>
<td>\ifndef</td>
<td>\newcommand</td>
<td>\renewcommand</td>
</tr>
</tbody>
</table>

COMMENT | TEXT

Note that most, but not all, are prefixed with backslash.

References

promptPackageSexpr  
Generates a shell of documentation for an installed package 

Description

Generates a shell of documentation for an installed package. The content is similar to ‘prompt-Package’ but information that can be computed is produced with Sexpr’s so that it is always up to date.

Usage

promptPackageSexpr(package, filename = NULL, final = TRUE, overview = FALSE, bib = TRUE)

Arguments

package  name of a package, a string
filename  name of a file where to write the generated Rd content, a string. The default should be sufficient in most cases.
final    logical; if TRUE the content should be usable without manual editing.
overview logical; if TRUE creates sections with hints what to put in them, otherwise such sections are written to the file but are commented out.
bib     If TRUE, create a comment line in the references section that will cause rebib to import all references from the default bib file.

Details

The generated skeleton is functionally (almost) equivalent to that produced by promptPackage. The difference is that while promptPackage computes some information and inserts it verbatim in the skeleton, promptPackageSexpr inserts Sexpr’s for the computation of the same information at package build time.

In this way there is no need to manually update information like the version of the package. The index of functions (which contains their descriptions) does not need manual updating, as well.

promptPackageSexpr needs to be called only once to create the initial skeleton. Then the Rd file can be edited as needed.

If the Rd file is generated with the option bib = TRUE (or the appropriate lines are added to the references section manually) the references can be updated at any time by a call of rebib.

todo: At the moment final=FALSE has the effect described for overview. At the time of writing this (2011-11-18) I do not remember if this is intentional or the corresponding ‘if’ clause contains | by mistake.

Value

the name of the file (invisibly)
Note
The automatically generated information is that of the installed (or at least built) package. Usually this is not a problem (and this is the idea of the function) but it means that if a developer is adding documentation for previously undocumented functions, they will appear in the 'Index' section only after the package is installed again. Similarly, if the description file of the package is changed, the package needs to be installed again for the changes to appear in the overview. Since the documentation is installed together with the package this is no surprise, of course. This may only cause a problem if documentation is produced with `R CMD Rd2pdf` before the updated version is installed.

This function is not called `repromptXXX` since the idea is that it is called only once and then the Rd file can be edited freely, see also 'Details'.

Author(s)
Georgi N. Boshnakov

promptUsage  Generate usage text for functions and methods

Description
Generates usage text for a function, S3 method or S4 method. The text is suitably formatted for inclusion in the usage section of Rd documentation.

Usage

```r
get_usage(object, name = NULL, force.function = FALSE, ..., S3class = "", S4sig = "", infix = FALSE, fu = TRUE, out.format = "text")
promptUsage(..., usage)
```

Arguments

- **object**: a function object or a character name of one, can be anonymous function.
- **name**: the name of a function, a string.
- **force.function**: enforce looking for a function.
- **S3class**: the S3 class of the function, a character vector.
- **out.format**: if "text", return the result as a character vector.
- **S4sig**: (the signature of an S4 method, as used in Rd macro \S4method).
- **infix**: if TRUE the function is an infix operator.
- **fu**: if TRUE the object is a function, otherwise it is something else (e.g. a variable or a constant like `pi` and `Inf`).
- **usage**: an usage object, see Details.
- **...**: for `promptUsage`, arguments to be passed on to `get_usage`; for `get_usage`, currently not used.
Details

get_usage() takes a function object, or the name of one, and creates text suitable for inclusion in the usage section of Rd documentation. The usage is generated from the function object. When in interactive R session, use cat() to print the result for copying and pasting into Rd documentation or saving to a file (otherwise, if the usage text contains backslashes, they may appear duplicated). Long text is wrapped on two or more lines. For example,

```r
cat(get_usage(lm))
##: lm(formula, data, subset, weights, na.action, method = "qr",
##: model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
##: contrasts = NULL, offset, \dots)
```

Argument "name" can be used to specify a print name for the function. This is most often needed for S3 methods. Compare

```r
cat(get_usage(summary.lm))
##: summary.lm(object, correlation = FALSE, symbolic.cor = FALSE, \dots)
```

and

```r
cat(get_usage(summary.lm, name = "summary"))
##: summary(object, correlation = FALSE, symbolic.cor = FALSE, \dots)
```

The call is just summary() in the latter. This fairly reflects the fact that S3 methods are normally called via the generic, but adding some explanatory text around the command is usually a good idea. For programatically generated usage sections in help pages, argument S3class can be used to get the standard Rd markup for S3 methods.

```r
cat(get_usage(summary.lm, "summary", S3class = "lm"))
##: \method{summary}{lm}(object, correlation = FALSE, symbolic.cor = FALSE, \dots)
```

(Note that \method can only be used in Usage sections.)

When object is an anonymous function, argument name is compulsory. For example,

```r
cat(get_usage(function(x = 3, y = "a"){}, "f"))
```

get_usage() can also be used to insert dynamically signatures of functions located in other objects, such as environments and lists, see the examples.

If a function is used as an infix operator, set infix = TRUE.

```r
get_usage("+", infix = TRUE)
##: [1] "e1 + e2"
get_usage("%in%", infix = TRUE)
##: [1] "x %in% table"
```

The name of the operator may be in a variable:

```r
op <- "+
get_usage(op, infix = TRUE)
##: [1] "e1 + e2"
```
Backticks are ok, as well,

```r
get_usage(`+`, infix = TRUE)
##: [1] "e1 + e2"
```

But if a backtickBed operator is in a variable, surprise springs:

```r
op <- `+`
get_usage(op, infix = TRUE)
##: [1] "e1 op e2"
```

In this case, don’t use backticks or, if you must, evaluate the argument:

```r
op <- `+`
get_usage(eval(op), name = `+`, infix = TRUE)
##: [1] "e1 + e2"
```

`promptUsage()` is mostly for internal use. It is like `get_usage()` with an additional argument, `usage`, used to pass a specially parsed argument list of class "f_usage", produced by other functions in `Rdpack`. In particular it could have been generated by a previous call to `get_usage()`.

**Value**

A character string or an object of S3 class "f_usage", see `pairlist2f_usage1` for its format.

**Note**

For an S3 or S4 generic, use the name of the function, not the object, see the examples.

These functions are for usage descriptions as they appear in the "usage" section of Rd files. Descriptions of S4 methods for "Methods" sections are dealt with by other functions.

**Author(s)**

Georgi N. Boshnakov

**See Also**

`parse_pairlist`

**Examples**

```r
u <- get_usage(lm)  # a long usage text
cat(u)
```

# if there are additional arguments in S3 methods,
# use names of the functions, not the objects, e.g.
get_usage("droplevels", S3class = "data.frame")
get_usage(name = "droplevels", S3class = "data.frame")
# (both give \method{droplevels}{data.frame}(x, except = NULL, \dots)"
```
# but this gives the args of the generic: "$\text{method}(\text{droplevels})(data.frame)(x, \ldots)"
get_usage(droplevels, S3class = "data.frame")

## a list containing some functions
sumaries <- list(xbar = function(x) mean(x), rho = function(x, y) cor(x, y))
get_usage(sumaries$xbar, name = "xbar")
get_usage(sumaries$rho, name = "rho")

## functions in an environment
esummaries <- list2env(summaries)
get_usage(esummaries$xbar, name = "xbar")
get_usage(esummaries$rho, name = "rho")

---

**Rdapply**

*Apply a function over an Rd object*

**Description**

Apply a function recursively over an Rd object, similarly to `rapply` but keeping attributes.

**Usage**

```
Rdapply(x, ...)
```

```
Rdtagapply(object, FUN, rdtag, classes = "character", how = "replace", ...
```

```
rattr(x, y)
```

**Arguments**

- `x`: the Rd object on which to apply a function.
- `object`: the Rd object on which to apply a function.
- `FUN`: The function to apply, see details.
- `rdtag`: apply FUN only to elements whose Rd_tag attribute is rdtag.
- `y`: an Rd object with the same structure as `x`, see ‘Details’.
- `...`: arguments to pass to `rapply`, see ‘Details’.
- `classes`: a character vector, passed on to `rapply`, see ‘Details’.
- `how`: a character string, passed on to `rapply`, see ‘Details’.

**Details**

`Rdapply` works like `rapply` but preserves the attributes of `x` and (recursively) any sublists of it. `Rdapply` first calls `rapply`, passing all arguments to it. Then it restores recursively the attributes by calling `rattr`. 
Note that the object returned by \texttt{rapply} is assumed to have identical structure to the original object. This means that argument \texttt{how} of \texttt{rapply} must not be "unlist" and normally will be "replace". \texttt{Rdtagapply} gives sensible default values for classes and how. See the documentation of \texttt{rapply} for details and the possible choices for classes, how or other arguments passed to it via "\dots".

\texttt{Rdtagapply} is a convenience variant of \texttt{Rdapply} for the common task of modifying or examining only elements with a given \texttt{Rd} tag attribute. Since the Rd equation macros \texttt{\eqn} and \texttt{\deqn} are assigned Rd tag "VERB" but are processed differently from other "VERB" pieces, pseudo-tags "mathVERB" and "nonmathVERB" are provided, such that "mathVERB" is for actions on the first argument of the mathematical macros \texttt{\eqn} and \texttt{\deqn}, while "nonmathVERB" is for actions on "VERB" macros in all other contexts. There is also a pseudo-tag "nonmath" for anything that is not math.

\texttt{rattr} is an auxiliary function which takes two Rd objects (with identical structure) and recursively examines them. It makes the attributes of any lists in the first argument identical to the corresponding attributes in the second.

\textbf{Value}

For \texttt{Rdapply} and \texttt{Rdtagapply}, an Rd object with some of its leaves replaced as specified above.

For \texttt{rattr}, the object \texttt{x} with attributes of any list elements of \texttt{y} set to the corresponding attributes of \texttt{y}.

\textbf{Note}

todo: may be it is better to rename the argument \texttt{FUN} of \texttt{Rdtagapply} to \texttt{f}, which is its name in \texttt{rapply}.

\textbf{Author(s)}

Georgi N. Boshnakov

\textbf{See Also}

\texttt{rapply}

\textbf{Examples}

\begin{verbatim}
# create an Rd object for the sake of example
u1 <- list_Rd(name = "Dummyname", alias = "dummyfun",
              title = "Dummy title", description = "Dummy description",
              usage = "dummyfun(x)",
              value = "numeric vector",
              author = "A. Author",
              examples = "\na <- matrix(1:6,nrow=2)\na %*% t(a)\nt(a) %*% a",
              Rd_class=TRUE )

# correct R code for examples but wrong for saving in Rd files
Rdo_show(u1)

# escape percents everywhere except in comments
# (actually, .anypercent escapes only unescaped percents)
\end{verbatim}
```r
rdo <- Rdapply(u1, Rdpack:::.anypercent, classes = "character", how = "replace")

# syntactically wrong R code for examples but ok for saving in Rd files
Rdo_show(rdo)

# Rdo2Rdf does this by default for examples and other R code,
# so code can be kept syntactically correct while processing.
# (reprompt() takes care of this too as it uses Rdo2Rdf for saving)
fn <- tempfile("u1", fileext="Rd")
Rdo2Rdf(u1, file = fn)

# the saved file contains escaped percents but they disappear in parsing:
file.show(fn)
Rdo_show(tools::parse_Rd(fn))

# if you think that sections should start on new lines,
# the following makes the file a little more human-friendly
# (by inserting new lines).
u2 <- Rdpack:::.Rd_tidy(u1)
Rdo2Rdf(u2, file = fn)
file.show(fn)
unlink(fn)
```

---

**Rdo2Rdf**

**Convert an Rd object to Rd file format**

**Description**

Converts an Rd object to Rd format and saves it to a file or returns it as a character vector. It escapes percents where necessary and (optionally) backslashes in the examples section.

**Usage**

```r
Rdo2Rdf(rdo, deparse = FALSE, ex_restore = FALSE, file = NULL,
        rcode = TRUE, srcfile = NULL)
```

**Arguments**

- **rdo**: an Rd object or a character vector, see ‘Details’.
- **deparse**: logical, passed to the print method for Rd objects, see ‘Details’.
- **ex_restore**: logical, if TRUE escapes backslashes where necessary.
- **file**: a filename where to store the result. If NULL or "missing", the result is returned as a character vector.
- **rcode**: if TRUE, duplicate backslashes in RCODE elements, see Details.
- **srcfile**: NULL or a file name, see ‘Details’.
Details

The description here is rather technical and incomplete. In any case it concerns almost exclusively Rd files which use escape sequences containing multiple consecutive backslashes or escaped curly braces (such things appear in regular expressions, for example).

In principle, this function should be redundant, since the print and as.character methods for objects of class "Rd" would be expected to do the job. I was not able to get the desired result that way (the deparse option to print did not work completely for me either).

Arguments ex_restore and rcode were added on an ad-hoc basis. rcode is more recent and causes Rdo2Rdf to duplicate backslashes found in any element Rd_tag-ed with "RCODE". ex_restore does the same but only for the examples section. In effect, if rcode is TRUE, ex_restore is ignored.

The initial intent of this function (and the package Rdpack as a whole was not to refer to the Rd source file. However, there is some flexibility in the Rd syntax that does not allow the source file to be restored identically from the parsed object. This concerns mainly backslashes (and to some extent curly braces) which in certain contexts may or may not be escaped and the parsed object is the same. Although this does not affect functionality, it may be annoying if the escapes in sections not examined by reprompt were changed.

If srcfile is the name of a file, the file is parsed and the Rd text of sections of rdo that are identical to sections from srcfile is taken directly from srcfile, ensuring that they will be identical to the original.

Value

NULL, if file is not NULL. Otherwise the Rd formatted text as a character vector.

Note

Here is an example when the author’s Rd source cannot be restored exactly from the parsed object.

In the Rd source "author" has two backslashes here: \author.

In the Rd source "author" has one backslash here: \author.

Both sentences are correct and the parsed file contains only one backslash in both cases. If reprompt looks only at the parsed object it will export one backslash in both cases. So, further reprompt()-ing will not change them again. This is if reprompt is called with sec_copy = FALSE. With the default sec_copy = TRUE, reprompt calls Rdo2Rdf with argument srcfile set to the name of the Rd file and since reprompt does not modify section "Note", its text is copied from the file and the author’s original preserved.

However, the arguments of \eqn are parse_Rd-ed differently (or so it seems) even though they are also in verbatim.

Author(s)

Georgi N. Boshnakov
Examples

```r
# # this keeps the backslashes in "author" (see Note above)
# reprompt(infile="/man/Rdo2Rdf.Rd")

# # this outputs "author" preceded by one backslash only.
# reprompt(infile="/man/Rdo2Rdf.Rd", sec_copy = FALSE)
```

### Rdo_append_argument

**Append an item for a new argument to an Rd object**

#### Description

Append an item for a new argument to an Rd object.

#### Usage

```r
Rdo_append_argument(rdo, argname, description = NA, indent = " ", create = FALSE)
```

#### Arguments

- `rdo` an Rd object.
- `argname` name of the argument, a character vector.
- `description` description of the argument, a character vector.
- `indent` a string, typically whitespace.
- `create` not used (todo: remove?)

#### Details

Appends one or more items to the section describing arguments of functions in an Rd object. The section is created if not present.

- If `description` is missing or NA, a "todo" text is inserted.
- The inserted text is indented using the string `indent`.
- The lengths of `argname` and `description` should normally be equal but if `description` is of length one, it is repeated to achieve this when needed.

#### Value

an Rd object

#### Author(s)

Georgi N. Boshnakov
Rdo_collectAliases

Examples

```
## the following creates Rd object rdo
dummyfun <- function(x) x
fn <- tempfile("dummyfun", fileext = ".Rd")
reprompt(dummyfun, filename = fn)
rdo <- tools::parse_Rd(fn)

## add documentation for arguments
## that are not in the signature of 'dummyfun()

dottext <- "further arguments to be passed on."
dots <- paste0("\\", "dots")
rdo2 <- Rdo_append_argument(rdo, dots, dottext, create = TRUE)
rdo2 <- Rdo_append_argument(rdo2, "z", "a numeric vector")

## reprompt() warns to remove documentation for non-existing arguments:
Rdo_show(reprompt(rdo2, filename = fn))

unlink(fn)
```

Rdo_collect_aliases

Collect aliases or other metadata from an Rd object

Description

Collect aliases or other metadata from an Rd object.

Usage

```
Rdo_collect_aliases(rdo)
Rdo_collect_metadata(rdo, sec)
```

Arguments

```
rdo an Rd object
sec the kind of metadata to collect, a character string, such as "alias" and "keyword".
```

Details

`Rdo_collect_aliases` finds all aliases in `rdo` and returns them as a named character vector. The name of an alias is usually the empty string, "", but it may also be "windows" or "unix" if the alias is wrapped in a `#ifdef` directive with the corresponding first argument.

`Rdo_collect_metadata` is a generalisation of the above. It collects the metadata from section(s) `sec`, where `sec` is the name of a section without the leading backslash. `sec` is assumed to be a section containing a single word, such as "keyword", "alias", "name".

Currently `Rdo_collect_metadata` is not exported.
Value

a named character vector, as described in Details.

Author(s)

Georgi N. Boshnakov

See Also

tools::.Rd_get_metadata

Examples

## this example originally (circa 2012) was:
##    infile <- file.path(R.home(), "src/library/base/man/timezones.Rd")
## but the OS conditional alias in that file has been removed.
## So, create an artificial example:
##    infile <- system.file("examples", "tz.Rd", package = "Rdpack")

## file.show(infile)
rd <- tools::parse_Rd(infile)

## The functions described here handle "ifdef" and similar directives.
## This detects OS specific aliases (windows = "onlywin" and unix = "onlyunix"):
Rdo_collect_aliases(rd)
Rdpack:::Rdo_collect_metadata(rd, "alias") # same

## In contrast, the following do not find "onlywin" and "onlyunix":
sapply(rd[which(tools::.RdTags(rd) == "\alias")], as.character)
tools::.Rd_get_metadata(rd, "alias")

Rdpack:::Rdo_collect_metadata(rd, "name")
Rdpack:::Rdo_collect_metadata(rd, "keyword")

Rdo_empty_sections rdo, with_bs = FALSE)
Rdo_drop_empty(rdo, sec = TRUE)

Find or remove empty sections in Rd objects

Description

Find or remove empty sections in Rd objects

Usage

Rdo_empty_sections(rdo, with_bs = FALSE)
Rdo_drop_empty(rdo, sec = TRUE)
Rdo_fetch

Get help pages as Rd objects

Description

Get a help page as an Rd object from an installed or source package.

Usage

Rdo_fetch(Rd_name = character(0), package, dir = ".", installed = TRUE)
Arguments

Rd_name names of one or more Rd help pages. name here is the name of an Rd file stripped from the extension.
package the package from which to get the Rd object, a character string.
dir a character string giving the root directory of a source package. Used only if package is missing.
installed if TRUE, the default, the Rd object is taken unconditionally from the installed package. If FALSE, the help page may be taken from a source tree, if appropriate (typically if package is in 'developer's mode under devtools, see Details).

Details

If Rd_name is a character string (typical use case), the corresponding help page is returned as an object from class "Rd". If the length of Rd_name is greater than one, the result is a Rd_named list containing the corresponding "Rd" objects. The default Rd_name = character(0) requests all Rd pages in the package.

Note that Rd_name does not contain the extension ".Rd" but the names in the returned list do.

Argument package names the package from which to fetch the documentation object. With the default installed = TRUE the object is taken unconditionally from the installed package. To get it from the source tree of a package, use argument "dir" instead. The default, ",", for dir is suitable for workflows where the working directory is the root of the desired package.

Argument installed concerns primarily development under package "devtools". "devtools" intercepts and modifies several base R commands, concerning access to system files and getting help, with the aim of rerouting them to the source trees of packages under developer's mode. If argument installed is TRUE, the default, the requested pages are taken from the installed package, even if it is in development mode. If argument installed is FALSE, the Rd objects are taken from the corresponding source tree, if the specified package is under developer's mode, and from the installed package otherwise.

Argument Rd_name is the name used in the \name section of Rd files.

When working off the source tree of a package, Rdo_fetch processes the Rd files, so roxygen2 users need to update them if necessary.

Value

if length(Rd_name) = 1, an object of class "Rd", otherwise a list of "Rd" objects.

Author(s)

Georgi N. Boshnakov

Examples

```r
## get a single help page
rdo <- Rdo_fetch("viewRd", package = "Rdpack")
Rdo_show(rdo)

## get a list of help pages
```
Rdo_platinsert

\begin{verbatim}
  rdo <- Rdo_fetch(c("viewRd", "reprompt"), package = "Rdpack")
  names(rdo)
\end{verbatim}

\begin{verbatim}
Rdo_platinsert  Insert or remove content in an Rd fragment
\end{verbatim}

Description

Insert or remove content in an Rd fragment.

Usage

\begin{verbatim}
  Rdo_platinsert(rdo, val, pos, before = TRUE)
  Rdo_platremove(rdo, from, to)
\end{verbatim}

Arguments

\begin{verbatim}
  rdo      an Rd object.
  val      the value to insert.
  pos      position.
  before   if TRUE, insert the new content at pos, pushing the element at pos forward.
  from     beginning of the region to remove.
  to       end of the region to remove.
\end{verbatim}

Details

Rdo_platinsert inserts val at position pos, effectively by concatenation.
Rdo_platremove removes elements from from to to.

Value

the modified rdo

Author(s)

Georgi N. Boshnakov
Rdo_get_argument_names

Get the names of arguments in usage sections of Rd objects

Description
Get the names of arguments in usage sections of Rd objects.

Usage
Rdo_get_argument_names(rdo)

Arguments
rdo an Rd object.

Details
All arguments names in the "arguments" section of rdo are extracted. If there is no such section, the results is a character vector of length zero.

Arguments which have different descriptions for different OS’es are included and not duplicated.
Arguments which have descriptions for a particular OS are included, irrespectively of the OS of the running R process. (todo: introduce argument to control this?)

Value
a character vector

Author(s)
Georgi N. Boshnakov

See Also
Rdo_get_item_labels

Examples
# Should be DIRECTLY executable !! ----
**Rdo_get_insert_pos**  
*Find the position of an "Rd_tag"*

**Description**
Find the position of an "Rd_tag".

**Usage**

\[
\text{Rdo_get_insert_pos}(\text{rdo}, \text{tag})
\]

**Arguments**

- `rdo` an Rd object
- `tag` the "Rd_tag" to search for, a string

**Details**
This function returns a position in `rdo`, where the next element carrying "Rd_tag" `tag` should be inserted. The position is determined as follows.

- If one or more elements of `rdo` have "Rd_tag" `tag`, then the position is one plus the position of the last such element.

- If there are no elements with "Rd_tag" `tag`, the position is one plus the length of `rdo`, unless `tag` is a known top level Rd section. In that case, the position is such that the standard ordering of sections in an Rd object is followed. This is set in the internal variable `.rd_sections`.

**Value**
an integer

**Author(s)**
Georgi N. Boshnakov

**Examples**

```r
# \h <- help("Rdo_macro")
# \rdo <- utils:::getHelpFile(h)
# \rdo <- Rdo_fetch("Rdo_macro", "Rdpack")

# ialias <- which(tools:::RdTags(rdo) == "\alias")
# ialias
# next_pos <- Rdo_get_insert_pos(rdo, "\alias") # 1 + max(ialias)
# next_pos
# stopifnot(next_pos == max(ialias) + 1)

# ikeyword <- which(tools:::RdTags(rdo) == "\keyword")
ikeyword
```
next_pos <- Rdo_get_insert_pos(rdo, "\keyword") # 1 + max(ikeyword)
next_pos
stopifnot(next_pos == max(ikeyword) + 1)

Rdo_get_item_labels Get the labels of items in an Rd object

Description
Get the labels of items in an Rd object.

Usage
Rdo_get_item_labels(rdo)

Arguments
rdo an Rd object.

Details
Rdo_get_item_labels(rdo) gives the labels of all "\item"s in rdo. Argument rdo is often a section or other Rd object fragment, see the examples.

Value
a character vector

Author(s)
Georgi N. Boshnakov

Examples
infile <- system.file("examples", "tz.Rd", package = "Rdpack")
rd <- tools::parse_Rd(infile)

## get item labels found anywhere in the Rd object
(items <- Rdo_get_item_labels(rd))

## search only in section "arguments" (i.e., get argument names)
## (note [[1]] - there is only one arguments section)
pos.args <- Rdo_locate_core_section(rd, "\arguments"[[1]])
(ARGS <- Rdo_get_item_labels(rd[[pos.args$pos]]))

## search only in section "value"
pos.val <- Rdo_locate_core_section(rd, "\value"[[1]])
(vals <- Rdo_get_item_labels(rd[[pos.val$pos]]))

## There are no other items in 'rd', so this gives TRUE:
all.equal(items, c(ARGS, vals)) # TRUE
**Rdo_insert**

Insert a new element in an Rd object possibly surrounding it with new lines.

**Description**

Insert a new element in an Rd object possibly surrounding it with new lines.

**Usage**

```r
date(d, format = format, tz = tz)
```

**Arguments**

- `rdo`: an Rd object
- `val`: the content to insert, an Rd object.
- `newline`: a logical value, controls the insertion of new lines before and after `val`, see 'Details'.

**Details**

Argument `val` is inserted in `rdo` at an appropriate position, see `Rdo_get_insert_pos` for detailed explanation.

If `newline` is TRUE, newline elements are inserted before and after `val` but only if they are not already there.

Typically, `val` is a section of an Rd object and `rdo` is an Rd object which is being constructed or modified.

**Value**

an Rd object

**Author(s)**

Georgi N. Boshnakov
### Rdo_insert_element

**Insert a new element in an Rd object**

**Description**

Insert a new element at a given position in an Rd object.

**Usage**

```r
ddo_insert_element(rdo, val, pos)
```

**Arguments**

- `rdo` an Rd object
- `val` the content to insert.
- `pos` position at which to insert `val`, typically an integer but may be anything accepted by the operator "[[]."

**Details**

`val` is inserted at position `pos`, between the elements at positions `pos-1` and `pos`. If `pos` is equal to 1, `val` is prepended to `rdo`. If `pos` is missing or equal to the length of `rdo`, `val` is appended to `rdo`.  

todo: allow vector `pos` to insert deeper into the object.  
todo: character `pos` to insert at a position specified by "tag" for example?  
todo: more guarded copying of attributes?

**Value**

an Rd object

**Author(s)**

Georgi N. Boshnakov

### Rdo_is_newline

**Check if an Rd fragment represents a newline character**

**Description**

Check if an Rd fragment represents a newline character

**Usage**

```r
ddo_is_newline(rdo)
```
Rdo_locate

Arguments

rdo an Rd object

Details

This is a utility function that may be used to tidy up Rd objects.

It returns TRUE if the Rd object represents a newline character, i.e. it is a character vector of length one containing the string "\n". Attributes are ignored.

Otherwise it returns FALSE.

Value

TRUE or FALSE

Author(s)

Georgi N. Boshnakov

Description

Find positions of elements for which a function returns TRUE. Optionally, apply another function to the selected elements and return the results along with the positions.

Usage

Rdo_locate(object, f = function(x) TRUE, pos_only = TRUE, lists = FALSE, fpos = NULL, nested = TRUE)

Arguments

object an Rd object
f a function returning TRUE if an element is desired and FALSE otherwise.
pos_only if TRUE, return only the positions; if this argument is a function, return also the result of applying the function to the selected element, see Details.
lists if FALSE, examine only leaves, if TRUE, examine also lists, see Details.
fpos a function with two arguments, object and position, it is called and the value is returned along with the position, see Details.
nested a logical value, it has effect only when lists is TRUE, see ‘Details’.
Details

With the default setting of `lists = FALSE`, the function `f` is applied to each leave (a character string) of the Rd object. If `lists = TRUE` the function `f` is applied also to each branch (a list). In this case, argument `nested` controls what happens when `f` returns TRUE. If `nested` is TRUE, each element of the list is also inspected recursively, otherwise this is not done and, effectively, the list is considered a leaf. If `f` does not return TRUE, the value of `nested` has no effect and the elements of the list are inspected.

The position of each object for which `f` returns TRUE is recorded as a numeric vector.

`fpos` and `pos_only` provide two ways to do something with the selected elements. Argument `fpos` is more powerful than `pos_only` but the latter should be sufficient and simpler to use in most cases.

If `fpos` is a function, it is applied to each selected element with two arguments, `object` and the position, and the result returned along with the position. In this case argument `pos_only` is ignored. If `fpos` is NULL the action depends on `pos_only`.

If `pos_only = TRUE`, `Rdo_locate` returns a list of such vectors (not a matrix since the positions of the leaves are, in general, at different depths).

If `pos_only` is a function, it is applied to each selected element and the result returned along with the position.

Value

a list with one entry for each selected element. Each entry is a numeric vector or a list with two elements:

- `pos` the position, a vector of positive integers,
- `value` the result of applying the function to the element at `pos`.

Note

The following needs additional thought.

In some circumstances an empty list, tagged with `Rd_tag` may turn up, e.g. `list()` with `Rd_tag="dots"` in an `arguments` section.

On the one hand this is a list. On the other hand it may be considered a leaf. It is not clear if any attempt to recurse into such a list should be made at all.

Author(s)

Georgi N. Boshnakov

See Also

- `Rdo_sections` and `Rdo_locate_core_section` which locate top level sections

Examples

# todo: put examples here!
Rdo_locate_leaves  Find leaves of an Rd object using a predicate

Description

Apply a function to the character leaves of an Rd object and return the indices of those for which the result is TRUE.

Usage

Rdo_locate_leaves(object, f = function(x) TRUE)

Arguments

- `object`: the object to be examined, usually a list.
- `f`: a function (predicate) returning TRUE for elements with the desired property.

Details

- `object` can be any list whose elements are character strings or lists. The structure is examined recursively. If `object` is a character vector, it is enclosed in a list.
- This function provides a convenient way to locate leaves of an Rd object with a particular content. The function is not limited to Rd objects but it assumes that the elements of `object` are either lists or character vectors and currently does not check if this is the case.
- **todo**: describe the case of `list()` (Rd_tag`ed.)

Value

- a list of the positions of the leaves for which the predicate gives TRUE. Each position is an integer vector suitable for the `[[` operator.

Author(s)

Georgi N. Boshnakov

Examples

dummyfun <- function(x) x

fn <- tempfile("dummyfun", fileext = "Rd")
reprompt(dummyfun, filename = fn)
rdo <- tools::parse_Rd(fn)

f <- function(x) Rdo_is_newline(x)
nl <- Rdo_locate_leaves(rdo, f)

length(nl) # there are quite a few newline leaves!
Rdo_macro

Format Rd fragments as macros (todo: a baffling title!)

Description
Format Rd fragments as macros, generally by putting them in a list and setting the "Rd_tag" as needed.

Usage
Rdo_macro(x, name)
Rdo_macro1(x, name)
Rdo_macro2(x, y, name)
Rdo_item(x, y)
Rdo_sigitem(x, y, newline = TRUE)

Arguments
x an object.
y an object.
name the "Rd_tag", a string.
nnewline currently ignored.

Details
Rdo_macro1 wraps x in a list with "Rd_tag" name. This is the representation of Rd macros with one argument.
Rdo_macro2 basically wraps a possibly transformed x and y in a list with "Rd_tag" name. More specifically, if x has a non-NULL "Rd_tag", x is wrapped in list. Otherwise x is left as is, unless x is a character string, when it is converted to a text Rd element and wrapped in list. y is processed in the same way. This is the representation of Rd macros with two arguments.
Rdo_macro returns an object with "Rd_tag" name, constructed as follows. If x is not of class "character", its attribute "Rd_tag" is set to name and the result returned without further processing. Otherwise, if it is of class "character", x is tagged as an Rd "TEXT" element. It is then wrapped in a list but only if name is one of "\eqn" or "\deqn". Finally, Rdo_macro1 is called on the transformed object.
Rdo_item is equivalent to Rdo_macro2 with name set to "\item".
Rdo_sigitem is for items which have the syntax used in description of signatures. In that case the first argument of "\item" is wrapped in "\code". If y is missing, a text inviting the author to provide a description of the function for this signature is inserted.
**Value**

An Rd element with appropriately set Rd_tag, as described in ‘Details’.

**Author(s)**

Georgi N. Boshnakov

---

**Rdo_modify**

*Replace or modify parts of Rd objects*

---

**Description**

Replace or modify parts of Rd objects.

**Usage**

\[
\text{Rdo\_modify}(rdo, val, create = \text{FALSE}, replace = \text{FALSE}, top = \text{TRUE})
\]

\[
\text{Rdo\_replace\_section}(rdo, val, create = \text{FALSE}, replace = \text{TRUE})
\]

**Arguments**

- **rdo**
  - an Rd object.
- **val**
  - an Rd fragment.
- **create**
  - if TRUE, create a new section, see ‘Details’.
- **replace**
  - a logical, if TRUE val replaces the old content, otherwise val is concatenated with it, see ‘Details’.
- **top**
  - a logical, if TRUE examine also the "Rd_tag" of rdo, see ‘Details’.

**Details**

Argument rdo is an Rd object (complete or a fragment) to be modified. val is an Rd fragment to use for modification.

Basically, val is appended to (if replace is FALSE) or replaces (if replace is TRUE) the content of an element of rdo which has the same "Rd_tag" as val.

Argument top specifies whether to check the "Rd_tag" of rdo itself, see below.

Here are the details.

If top is \text{TRUE} and rdo and val have the same (non-NULL) "Rd_tag", then the action depends on replace (argument create is ignored in this case). If replace is \text{TRUE}, val is returned. Otherwise rdo and val are, effectively, concatenated. For example, rdo may be the "arguments" section of an Rd object and val may also be an "arguments" section containing new arguments.

Otherwise, an element with the "Rd_tag" of val is searched in rdo using \text{tools::RdTags()}. If such elements are found, the action again depends on replace.
1. If `replace` is a character string, then the first element of `rdo` that is a list whose only element is identical to the value of `replace` is replaced by `val`. If such an element is not present and `create` is `TRUE`, `val` is inserted in `rdo`. If `create` is `FALSE`, `rdo` is not changed.

2. If `replace` is `TRUE`, the first element found is replaced with `val`.

3. If `replace` is `FALSE`, `val` is appended to the first element found.

If no element with the "Rd_tag" of `val` is found the action depends on `create`. If `create` is `TRUE`, then `val` is inserted in `rdo`, effectively creating a new section. If `create` is `FALSE`, an error is thrown.

Rdo_replace_section is like Rdo_modify with argument `top` fixed to `TRUE` and the default for argument `replace` set to `TRUE`. It hardly makes sense to call Rdo_replace_section with `replace = FALSE` but a character value for it may be handy in some cases, see the examples.

### Value

an Rd object or fragment, as described in 'Details'

### Author(s)

Georgi N. Boshnakov

### Examples

```r
# a <- tools::parse_Rd("./man/promptUsage.Rd")
# char2Rdpiece("documentation", "keyword")

# this changes a keyword from Rd to documentation
# Rdo_replace_section(a, char2Rdpiece("documentation", "keyword"), replace = "Rd")
```

### Description

Simple modification of Rd objects.

### Usage

`Rdo_modify_simple(rdo, text, section, ...)`

### Arguments

- `rdo` an Rd object.
- `text` a character vector
- `section` name of an Rd section, a string.
- `...` additional arguments to be passed to `Rdo_modify`. 
### Rdo_piecetag

**Details**

Argument `text` is used to modify (as a replacement of or addition to) the content of section `section` of `rdo`.

This function can be used for simple modifications of an Rd object using character content without converting it separately to Rd. `text` is converted to Rd with `char2Rdpiece(text, section)`. The result is passed to `Rdo_modify`, together with the remaining arguments.

**Value**

an Rd object

**Author(s)**

Georgi N. Boshnakov

**See Also**

`Rdo_modify`

---

| Rdo_piecetag | Give information about Rd elements |

---

### Description

Give information about Rd elements.

### Usage

- `Rdo_piecetag(name)`
- `Rdo_sectype(x)`
- `is_Rdsecname(name)`

### Arguments

- `name` the name of an Rd macro, a string.
- `x` the name of an Rd macro, a string.

### Details

- `Rdo_piecetag` gives the "Rd_tag" of the Rd macro `name`.
- `Rdo_sectype` gives the "Rd_tag" of the Rd section `x`.
- `is_Rdsecname(name)` returns TRUE if `name` is the name of a top level Rd section.

The information returned by these functions is obtained from the character vectors `Rdo_piece_types` and `Rdo_predefined_sections`.
**Author(s)**
Georgi N. Boshnakov

**See Also**
Rdo_piece_types and Rdo_predefined_sections

**Examples**

```r
do_piece(tag("eqn")) # ==> "VERB"
do_piece("code") # ==> "RCODE"

do_sectype("usage") # ==> "RCODE"
do_sectype("title") # ==> "TEXT"
do_sectype("arguments")
```

**Description**
Removes srcref attributes from Rd objects.

**Usage**

```r
do_remove_srcref(d)
```

**Arguments**

- `d` an Rd object

**Details**

srcref attributes (set by parse_Rd) may be getting in the way during manipulation of Rd objects, such as comparisons, addition and replacement of elements. This function traverses the argument and removes the srcref attribute from all of its elements.

**Value**

an Rd object with no srcref attributes.

**Author(s)**
Georgi N. Boshnakov
**Rdo_reparse**

Reparse an Rd object.

**Usage**

Rdo_reparse(rdo)

**Arguments**

rdo an Rd object

**Details**

Rdo_reparse saves rdo to a temporary file and parses it with parse_Rd. This ensures that the Rd object is a "canonical" one, since one and the same Rd file can be produced by different (but equivalent) Rd objects.

Also, the functions in this package do not attend to attribute "srcref" (and do not use it) and reparsing takes care of this. (todo: check if there is a problem if the tempfile disappears.)

(Murdoch 2010; Francois 2014)

**References**

Romain Francois (2014). *bibtex: bibtex parser*. R package version 0.4.0.


**Examples**

```r
# the following creates Rd object rdo
dummyfun <- function(x) x
fn <- tempfile("dummyfun", fileext = "Rd")
reprompt(dummyfun, filename = fn)
rdo <- tools::parse_Rd(fn)
dottext <- "further arguments to be passed on."

dots <- paste0("\\", "dots")
rdo2 <- Rdo_append_argument(rdo, dots, dottext, create = TRUE)
rdo2 <- Rdo_append_argument(rdo2, "z", "a numeric vector")
Rdo_show(Rdo_reparse(rdo2))
```
# the following does ot show the arguments. (todo: why?)
#   (see also examples in Rdo_append_argument)
Rdo_show(rdo2)

unlink(fn)

---

**Rdo_sections**  *Locate the sections in Rd objects*

**Description**

Locate the Rd sections in an Rd object and return a list of their positions and names.

**Usage**

```
Rdo_sections(rdo)
Rdo_locate_core_section(rdo, sec)
```

**Arguments**

- **rdo**: an Rd object.
- **sec**: the name of a section, a character string. For builtin sections the leading backslash should be included.

**Details**

*Rdo_sections* locates all sections at the top level in an Rd object. This includes the predefined sections and the user defined sections. Sections wrapped in #ifdef directives are also found. 

*Rdo_sections* returns a list with one entry for each section in *rdo*. This entry is a list with components "pos" and "title" giving the position (suitable for use in "[[")) and the title of the section. For user defined sections the actual name is returned, not "section".

The names of the sections are returned as single strings without attributes. The titles of predefined sections are single words but user defined sections may have longer titles and sometimes contain basic markup.

*Rdo_locate_core_section* works similarly but returns only the results for section *sec*. Currently it simply calls *Rdo_sections* and returns only the results for *sec*.

Note that for consistency *Rdo_locate_core_section* does not attempt to simplify the result in the common case when there is only one instance of the requested section—it is put in a list of length one.

(Murdoch 2010) (Francois 2014)
Value

A list giving the positions and titles of the sections in rdo as described in 'Details'. The format is essentially that of \texttt{Rdo_locate}, the difference being that field "value" from that function is renamed to "title" here.

- \texttt{pos} the position, a vector of positive integers,
- \texttt{title} a standard section name, such as "\name" or, in the case of "\section", the actual title of the section.

Note

I wrote \texttt{Rdo_sections} and \texttt{Rdo_locate_core_section} after most of the core functionality was tested. Currently these functions are underused—they can replace a number of internal and exported functions.

Author(s)

Georgi N. Boshnakov

References

Romain Francois (2014). \textit{bibtex: bibtex parser}. R package version 0.4.0.


See Also

\texttt{Rdo_locate}

Examples

```r
infile <- system.file("examples", "tz.Rd", package = "Rdpack")
rd <- tools::parse_Rd(infile)

## Locate all top level sections in rd
sections <- Rdo_sections(rd)
## How many sections there are in rd?
length(sections)
## What are their titles?
sapply(sections, function(x) x$title)

## The names of builtin sections include the backslash
Rdo_locate_core_section(rd, "\\title")

## Locate a user defined section
Rdo_locate_core_section(rd, "Further information")

## The names of builtin sections include the backslash
Rdo_locate_core_section(rd, "\\details")
```
Replace a section in an Rd file

## Description

Replace a section in an Rd file.

## Usage

```r
Rdo_set_section(text, sec, file, ...)
```

## Arguments

- `text`: the new text of the section, a character vector.
- `sec`: name of the section.
- `file`: name of the file.
- `...`: arguments to be passed on to `Rdo_modify`.

## Details

Parses the file, replaces the specified section with the new content and writes the file back. The text is processed as appropriate for the particular section (`sec`).

For example:

```r
Rdo_set_section("Georgi N. Boshnakov","author","./man/Rdo2Rdf.Rd")
```

(Some care is needed with the author field for "xxx-package.Rd" files, such as "Rdpack-package.Rd", where the Author(s) field has somewhat different layout.)

By default `Rdo_set_section` does not create the section if it does not exist, since this may not be desirable for some Rd files. The "..." arguments can be used to change this, they are passed on to `Rdo_modify`, see its documentation for details.

## Value

This function is used mainly for the side effect of changing file. It returns the Rd formatted text as a character vector.

## Author(s)

Georgi N. Boshnakov

## See Also

`Rdo_modify`
Rdo_show

Examples

fnA <- tempfile("dummyfun", fileext = "Rd")
dummyfun <- function(x) x
reprompt(dummyfun, filename = fnA)
Rdo_show(tools::parse_Rd(fnA))

## set the author section, create it if necessary.
Rdo_set_section("A.A. Author", "author", fnA, create = TRUE)
Rdo_show(tools::parse_Rd(fnA))

## replace the author section
Rdo_set_section("Georgi N. Boshnakov", "author", fnA)
Rdo_show(tools::parse_Rd(fnA))

unlink(fnA)

---

Rdo_show  Convert an Rd object to text and show it

Description

Render an Rd object as text and show it.

Usage

Rdo_show(rdo)

Arguments

rdo an Rd object

Details

Rdo_show renders the help page represented by rdo as text and shows it with file.show().
Rdo_show is a simplified front end to utils::Rd2txt. See viewRd for more complete rendering,
including of references and citations.

Value

Invisible NULL. The function is used for the side effect of showing the text representation of rdo.

Author(s)

Georgi N. Boshnakov

See Also

viewRd
## create a minimal Rd object

```r
u1 <- list_Rd(name = "Dummyname", alias = "Dummyallias1",
               title = "Dummy title", description = "Dummy description",
               Rd_class = TRUE )
```

```r
## Not run:
## run this interactively:
Rdo_show(u1)
```

## End(Not run)

### Rdo_tag

Set the Rd_tag of an object

#### Description

Set the Rd_tag of an object.

#### Usage

```r
Rdo_comment(x = "%%")
Rdo_tag(x, name)
Rdo_verb(x)
Rdo_Rcode(x)
Rdo_text(x)
Rdo_newline()
```

#### Arguments

- `x`  an object, appropriate for the requested Rd_tag.
- `name`  the tag name, a string.

#### Details

These functions simply set attribute "Rd_tag" of `x`, effectively assuming that the caller has prepared it as needed.

- `Rdo_tag` sets the "Rd_tag" attribute of `x` to name. The other functions are shorthands with a fixed name and no second argument.
- `Rdo_comment` tags an Rd element as comment.
- `Rdo_newline` gives an Rd element representing an empty line.
Rdo_tags

Value
x with its "Rd_tag" set to name (Rdo_tag), "TEXT" (Rdo_text), "VERB" (Rdo_verb) or "RCODE" (Rdo_Rcode).

Author(s)
Georgi N. Boshnakov

Rdo_tags 

Give the Rd tags at the top level of an Rd object

Description
Give the Rd tags at the top level of an Rd object.

Usage
Rdo_tags(rdo, nulltag = "")

Arguments
rdo an Rd object.
nulltag a value to use when Rd_tag is missing or NULL.

Details
The "Rd_tag" attributes of the top level elements of rdo are collected in a character vector. Argument nulltag is used for elements without that attribute. This guarantees that the result is a character vector.
Rdo_tags is similar to the internal function tools:::RdTags. Note that tools:::RdTags may return a list in the rare cases when attribute Rd_tags is not present in all elements of rdo.

Value
a character vector

Author(s)
Georgi N. Boshnakov

See Also
Rdo_which, Rdo_which_tag_eq, Rdo_which_tag_in

Examples
##---- Should be DIRECTLY executable !! ----
rdo_text_restore  Ensure exported fragments of Rd are as the original

Description

For an Rd object imported from a file, this function ensures that fragments that were not not changed during the editing of the object remain unchanged in the exported file. This function is used by reprompt() to ensure exactly that.

Usage

rdo_text_restore(cur, orig, pos_list, file)

Arguments

- **cur**: an Rd object
- **orig**: an Rd object
- **pos_list**: a list of srcref objects specifying portions of files to replace, see 'Details'.
- **file**: a file name, essentially a text representation of cur.

Details

This is essentially internal function. It exists because, in general, it is not possible to restore the original Rd file from the Rd object due to the specifications of the Rd format. The file exported from the parsed Rd file is functionally equivalent to the original but equivalent things for the computer are not necessarily equally pleasant for humans.

This function is used by reprompt when the source is from a file and the option to keep the source of unchanged sections as in the original.

todo: needs clean up, there are unnecessary arguments in particular.

Value

the main result is the side effect of replacing sections in file not changed by reprompt with the original ones.

Author(s)

Georgi N. Boshnakov

See Also

reprompt
Rdo_which

Find elements of Rd objects for which a condition is true

Description

Find elements of Rd objects for which a condition is true.

Usage

Rdo_which(rdo, fun)
Rdo_which_tag_eq(rdo, tag)
Rdo_which_tag_in(rdo, tags)

Arguments

rdo       an Rd object.
fun       a function to evaluate with each element of rdo.
tag      a character string.
tags      a character vector.

Details

These functions return the indices of the (top level) elements of rdo which satisfy a condition.
Rdo_which finds elements of rdo for which the function fun gives TRUE.
Rdo_which_tag_eq finds elements with a specific Rd_tag.
Rdo_which_tag_in finds elements whose Rd_tag’s are among the ones specified by tags.

Value

a vector of positive integers

Author(s)

Georgi N. Boshnakov

See Also

Rdo_locate which searches recursively the Rd object.
Examples

```r
## get the help page for topoc seq()
rdo_seq <- tools::Rd_db("base")[["seq.Rd"]]
## find location of aliases in the topic
( ind <- Rdo_which_tag_eq(rdo_seq, "\alias") )
## extract the first alias
rdo_seq[[ ind[1] ]]
## Not run:
## extract all aliases
rdo_seq[ind]

## db_bibtex <- tools::Rd_db("bibtex")
## names(db_bibtex)
## ## Rdo object for read.bib()
## rdo_read.bib <- db_bibtex["read.bib.Rd"]
## Rdo_tags(rdo_read.bib)
##
## ## which elements of read.bib are aliases?
## Rdo_which_tag_eq(rdo_read.bib, "\alias")
## rdo_read.bib[[3]]
##
## ## which elements of read.bib contain R code?
## Rdo_which(rdo_read.bib, function(x) any(Rdo_tags(x) == "RCODE") )
## rdo_read.bib[[5]]
##
## ## which contain prose?
## Rdo_which(rdo_read.bib, function(x) any(Rdo_tags(x) == "TEXT") )
```

---

Rdpack_bibstyles

**Set up a custom style for references in help pages**

Description

Set up a custom style for references in help pages.

Usage

```r
Rdpack_bibstyles(package, authors)
```

Arguments

- **package**: the name of a package, a character string.
- **authors**: if equal to "LongNames", use full names of authors in reference lists, see Details.
Details

This is the initial implementation of support for styles for lists of bibliography references. Currently setting "authors" to "LongNames" will cause the references to appear with full names, eg John Smith rather than in the default Smith J style.

Package authors can request this feature by adding the following line to their .onLoad function (if their package has one):

```r
Rdpack::Rdpack_bibstyles(package = pkg, authors = "LongNames")
```

of just copy the following definition in a package that does not have .onLoad:

```r
.onLoad <- function(lib, pkg){
    Rdpack::Rdpack_bibstyles(package = pkg, authors = "LongNames")
    invisible(NULL)
}
```

After building and installing the package the references should be using long names.

Value

in .onLoad(), the function is used purely to set up a bibstyle as discussed in Details.

Internally, **Rdpack** uses it to extract styles set up by packages:
- if called with argument package only, the style requested by that package;
- if called with no arguments, a list of all styles.

Author(s)

Georgi N. Boshnakov

---

**Rdreplace_section**  
*Replace the contents of a section in one or more Rd files*

Description

Replace the contents of a section in one or more Rd files.

Usage

```r
Rdreplace_section(text, sec, pattern, path = "/man", exclude = NULL, ...)```

Arguments

text  
the replacement text, a character string.

sec  
the name of the section without the leading backslash, as for Rdo_set_section.

pattern  
regular expression for R files to process, see Details.

path  
the directory were to look for the Rd files.

exclude  
regular expression for R files to exclude, see Details.

...  
not used.
Details

Rdreplace_section looks in the directory specified by path for files whose names match pat and
drops those whose names match exclude. Then it replaces section sec in the files selected in this
way.

Rdreplace_section is a convenience function to replace a section (such as a keyword or author) in several files in one go. It calls Rdo_set_section to do the work.

Value

A vector giving the full names of the processed Rd files, but the function is used for the side effect of modifying them as described in section Details.

Author(s)

Georgi N. Boshnakov

See Also

Rdo_set_section

Examples

## Not run:
# replace the author in all Rd files except pkgname-package
Rdreplace_section("A. Author", "author", ".*\[.\]Rd$", exclude = "-package[.]Rd$")

## End(Not run)
Details

Rd_combo parses each file in rd, applies f to the Rd object, and applies the function .MORE (if supplied) on the results of f.

A typical value for .MORE is reprompt or another function that saves the resulting Rd object.
todo: Rd_combo is already useful but needs further work.

Examples

```r
## Not run:
rdnames <- dir(path = ".man", pattern=".*[.]Rd$", full.names=TRUE)

## which Rd files don't have a value section?
counts <- unlist(Rd_combo(rdnames, function(x) length(Rdo_locate_core_section(x, "\value"))))
rdnames[counts == 0]

## reprompt all files
Rd_combo(rdnames, reprompt)
for(nam in rdnames) try(reprompt(nam))
for(nam in rdnames) try(reprompt(nam, sec_copy=FALSE))
```

rebib

Work with bibtex references in Rd documentation

Description

Work with bibtex references in Rd documentation.

Usage

```r
rebib(infile, outfile, 

inspect_Rdbib(rdo, force = FALSE, 
```

Arguments

```r
infile          name of the Rd file to update, a character string.
outfile         a filename for the updated Rd object.
...             further arguments to be passed to get_bibentries, see ‘Details’.
rdo             an Rd object.
force           if TRUE, re-insert previously imported references. Otherwise do not change such references.
```
Details

`inspect_Rdbib` takes an Rd object and processes the references as specified below.

The user level function is `rebib`. It parses the Rd file `infile`, calls `inspect_Rdbib` to process the references, and writes the modified Rd object to file `outfile`. If `outfile` is missing it is set to the basename of `infile`. If `outfile` is the empty string, "", then `outfile` is set to `infile`.

The default Bibtex file is "REFERENCES.bib" in the current working directory. Arguments ". . . " can be used to change the name of the bib file and its location. Argument `bibfile` can be used to overwrite the default name of the bib file. Argument `package` can be used to specify that the bib file should be taken from the root of the installation directory of package `package`, see `get_bibentries` for details.

The following scheme can be used for incorporation of bibliographic references from BibTeX files. Note however, that usually it is more convenient to use the approach based on Rd macros like `\insertRef`, see `insert_ref` and the vignette(s).

The Bibtex key for each reference is put in a comment line in the references section, as in

```
\references{
  % bibentry: key1

  % bibentry: key2

  ...
}
```

At least one space after the colon, ":", is required, spaces before "bibentry:" are ignored.

Then run `rebib()` on the file, see the example section for a way to run `rebib()` on all files in one go.

Each reference is inserted immediately after the comment line specifying it and a matching comment line marking its end is inserted.

Before inserting a reference, a check for the matching ending line is made, and if such a line is found, the reference is not inserted. This means that to add new references it is sufficient to give their keys, as described above and run the function again. References that are already there will not be duplicated.

The inserted reference may also be edited, if necessary. As long as the two comment lines enclosing it are not removed, the reference will not be overwritten by subsequent calls of the functions described here. Any text outside the markers delineating references inserted by this mechanism is left unchanged, including references inserted by other means.

To include all references from the bib file, the following line can be used:

```
% bibentry:all
```

Notice that there is no space after the colon in this case. In this case a marker is put after the last reference and the whole thing is considered one block. So, if the end marker is present and `force` is FALSE, none will be changed. Otherwise, if `force` is TRUE, the whole block of references will be removed and all references currently in the bib file will be inserted.

The main purpose of `bibentry:all` is for use in a package overview file. The reference section in the file "package-package" generated by `promptPackageSexpr` uses this feature (but the user still needs to call `rebib` to insert the references).
Value

for inspect_Rdbib, the modified Rd object.

rebib is used mainly for the side effect of creating a file with the references updated. It returns the Rd object created by parsing and modifying the Rd file.

Author(s)

Georgi N. Boshnakov

See Also

insert_ref and the vignette(s) for the recommended way to import BibTeX references and citations.

Examples

## Not run:
# update references in all Rd files in the package's 'man' directory
#
rdnames <- dir(path = "./man", pattern=".*[.]Rd$", full.names=TRUE)
llapply(rdnames, function(x) rebib(x, package="Rdpack"))
## End(Not run)

---

**reprompt**

Update the documentation of a topic

Description

Examine the documentation of functions, methods or classes and update it with additional arguments, aliases, methods or slots, as appropriate. This is an extension of the promptXXX() family of functions.

Usage

reprompt(object, infile = NULL, Rdtext = NULL, final = TRUE,
  type = NULL, package = NULL, methods = NULL, verbose = TRUE,
  filename = NULL, sec_copy = TRUE, edit = FALSE, ...)

Arguments

- `object`: the object whose documentation is to be updated, such as a string, a function, a help topic, a parsed Rd object, see ‘Details’.
- `infile`: a file name containing Rd documentation, see ‘Details’.
- `Rdtext`: a character string with Rd formatted text, see ‘Details’.
- `final`: logical, if TRUE modifies the output of prompt so that the package can be built.
type  type of topic, such as "methods" or "class", see ‘Details’.

package  package name; document only objects defined in this package, especially useful for methods, see ‘Details’.

methods  used for documentation of S4 methods only, rarely needed even for them. This argument is passed on to `promptMethods`, see its documentation for details.

verbose  if TRUE print messages on the screen.

filename  name of the file for the generated Rd content; if NULL, a suitable file name is generated, if TRUE it will be set to `infile`, if FALSE the Rd text is returned, see ‘Details’.

...  currently not used.

sec_copy  if TRUE copy Rd contents of unchanged sections in the result, see Details.

edit  if TRUE call file.edit just before returning. This argument is ignored if filename is FALSE.

Details

The typical use of `reprompt` is with one argument, as in

```r
reprompt(infile = './Rdpack/man/reprompt.Rd')
reprompt(reprompt)
reprompt("reprompt")
```

`reprompt` updates the requested documentation and writes the new Rd file in the current working directory. When argument `infile` is used, the descriptions of all objects described in the input file are updated. When an object or its name is given, `reprompt` looks first for installed documentation and processes it in the same way as in the case of `infile`. If there is no documentation for the object, `reprompt` creates a skeleton Rd file similar to the one produced by the base R functions from the `prompt` family (if `final = TRUE`, the default, it modifies the output of `prompt()`, so that the package can be built).

To document a function, say `myfun`, in an existing Rd file, just add `myfun()` to the usage section in the file and call `reprompt()` on that file. Put quotes around the function name if it is non-syntactic. For replacement functions (functions with names ending in `<-`) `reprompt()` will insert the proper usage statement. For example, if the signature of `xxx<-` is `(x,...,value)` then both, "xxx<-()" and `xxx() <-value` will be replaced by `xxx(x,...) <-value`.

For S4 methods and classes the argument "package" is often needed to restrict the output to methods defined in the package of interest.

```r
reprompt("myfun-methods")
reprompt("[<-methods", package = "mypackage")
reprompt("[<-", type = "methods", package = "mypackage") # same
reprompt("myclass", type = "class", package = "mypackage")
reprompt("myclass-class", package = "mypackage") # same
```
Without the "package" argument the reprompt for "<-" would give all methods defined by loaded packages at the time of the call.

Currently reprompt functionality is not implemented for topic "package" but if object has the form "name-package" (or the equivalent with argument topic) and there is no documentation for package?name, reprompt calls promptPackageSexpr to create the required shell. Note that the shell produced by promptPackageSexpr does not need ‘reprompting’ since the automatically generated information is included by \Sexpr’s, not literal strings.

Below are the details.

Typically, only one of object, infile, and Rdtext is supplied. Warning messages are issued if this is not the case.

The object must have been made available by the time when reprompt() is issued. If the object is in a package this is typically achieved by a library() command.

object may be a function or a name, as accepted by the ? operator. If it has the form "name-class" and "name-methods" a documentation shell for class "name" or the methods for generic function "name" will be examined/created. Alternatively, argument type may be set to "class" or "methods" to achieve the same effect.

infile specifies a documentation file to be updated. If it contains the documentation for one or more functions, reprompt examines their usage statements and updates them if they have changed. It also adds arguments to the "arguments" section if not all arguments in the usage statements have entries there. If infile describes the methods of a function or a class, the checks made are as appropriate for them. For example, new methods and/or slots are added to the corresponding sections.

It is all too easy in interactive use to forget to name the infile argument, compare

```
reprompt("./man/reprompt.Rd") vs. reprompt(infile = "/man/reprompt.Rd")
```

A convenience feature is that if infile is missing and object is a character string ending in ".Rd" and containing a forward slash (i.e. it looks like Rd file in a directory), then it is taken to be infile.

Rdtext is similar to infile but the Rd content is taken from a character vector.

If Rd content is supplied by infile or Rdtext, reprompt uses it as a basis for comparison. Otherwise it tries to find installed documentation for the object or topic requested. If that fails, it resorts to one of the promptXXX functions to generate a documentation shell. If that also fails, the requested object or topic does not exist.

If the above succeeds, the function examines the current definition of the requested object(s), methods, or class and amends the documentation with any additional items it finds.

For Rd objects describing one or more functions, the usage expressions are checked and replaced, if they have changed. Arguments appearing in one or more usage expressions and missing from section "Arguments" are amended to it with content "Describe ..." and a message is printed. Arguments no longer in the usage statements are NOT removed but a message is issued to alert the user. Alias sections are inserted for any functions with "usage" but without "alias" section.

If filename is a character string, it is used as is, so any path should be included in the string. Argument filename usuallly is omitted since the automatically generated file name is suitable in most cases. Exceptions are functions with non-standard names (such as replacement functions whose names end in "<-") for which the generated file names may not be acceptable on some operating systems.
If `filename` is missing or `NULL`, a suitable name is generated as follows. If `infile` is supplied, `filename` is set to a file with the same name in the current working directory (i.e. any path information in `infile` is dropped). Otherwise, `filename` is obtained by appending the name tag of the Rd object with ".Rd".

If `filename` is `TRUE`, it is set to `infile` or, if `infile` is missing or `NULL`, a suitable name is generated as above. This can be used to change `infile` in place.

If `filename` is `FALSE`, the Rd text is returned as a character vector and not written to a file.

If `edit` is `TRUE`, the reprompted file is opened in an editor, see also `ereprompt` (‘e’ for ‘edit’) which is like `reprompt` but has as default `edit = TRUE` and some other related settings.

`file.edit()` is used to call the editor. Which editor is opened depends on the OS and on the user configuration. RStudio users will probably prefer the ‘Reprompt’ add-in or the underlying function `RStudio_reprompt`. Emacs users would normally have set up `emacsclient` as their editor and this is automatically done by EMACS/ESS (even on Windows).

If argument `sec_copy` is `TRUE` (the default), `reprompt` will, effectively, copy the contents of (some) unchanged sections, thus ensuring that they are exactly the same as in the original. This needs additional work, since parsing an Rd file and then exporting the created Rd object to an Rd file does not necessarily produce an identical Rd file (some escape sequences may be changed in the process, for example). Even though the new version should be functionally equivalent to the original, such changes are usually not desirable. For example, if such changes creep into the Details section (which `reprompt` never changes) the user may be annoyed or worried.

**Value**

if `filename` is a character string or `NULL`, the name of the file to which the updated shell was written. Otherwise, the Rd text is returned as a character vector.

**Note**

The arguments of `reprompt` are similar to `prompt`, with some additions. As in `prompt`, `filename` specifies the output file. In `reprompt` a new argument, `infile`, specifies the input file containing the Rd source.

When `reprompt` is used to update sources of Rd documentation for a package, it is best to supply the original Rd file in argument `infile`. Otherwise, if the original Rd file contains `\Sexpr` commands, `reprompt` may not be able to recover the original Rd content from the installed documentation. Also, the fields (e.g. the keywords) in the installed documentation may not be where you expect them to be. (This may be addressed in a future revision.)

While `reprompt` adds new items to the documentation, it never removes existing content. It only issues a suggestion to remove an item, if it does not seem necessary any more (e.g. a removed argument from a function definition).

`reprompt` handles usage statements for S3 and S4 methods introduced with any of the macros `\method`, `\S3method` and `\S4method`, as in `\method{fun}{class}(args...)`. `reprompt` understands also subsetting ans subassignment operators. For example, suppose that the `\arguments` section of file "bracket.Rd" contains these directives (or any other wrong signatures):

```r
\method[[]{ts}()
\method[[][]{Date}()
```
Then reprompt("./bracket.Rd") will change them to

\method{\texttt{\[\}}{\texttt{ts}}(x, i, j, \texttt{drop = TRUE})
\method{\texttt{\[\[\}}{\texttt{Date}}(x, \texttt{\\dots}, \texttt{drop = TRUE})

This works for the assignment operators and functions, as well. For example, any of these

\method{\texttt{\}[\<-\}}{\texttt{POSIXlt}}()
\method{\texttt{\[\}}{\texttt{POSIXlt}}(x, j) <- value

will be converted by reprompt to the standard form

\method{\texttt{\[\}}{\texttt{POSIXlt}}(x, i, j) <- value

Note that the quotes in `\[<-\]` above.

Usage statements for functions are split over two or more lines if necessary. The user may edit them afterwards if the automatic splitting is not appropriate, see below.

The usage section of Rd objects describing functions is left intact if the usage has not changed. To force reprompt to recreate the usage section (e.g. to reformat long lines), invalidate the usage of one of the described functions by removing an argument from its usage expression. Currently the usage section is recreated completely if the usage of any of the described functions has changed.

Manual formatting may be lost in such cases.

Author(s)

Georgi N. Boshnakov

See Also

eresponse which by default calls the editor on the original file

Examples

```r
## note: usage of reprompt() is simple. the examples below are bulky
## because they simulate various usage scenarios with commands,
## while in normal usage they would be due to editing.

## change to a temporary directory to avoid clogging up user's
cur_wd <- getwd()
tmpdir <- tempdir()
setwd(tmpdir)

## as for prompt() the default is to save in current dir as "seq.Rd".
## the argument here is a function, reprompt finds its doc and
## updates all objects described along with 'seq'.
## (In this case there is nothing to update, we have not changed 'seq'.)
fnseq <- reprompt(seq)
```
## let's parse the saved Rd file (the filename is returned by reprompt)

rdoseq <- tools::parse_Rd(fnseq)  # parse fnseq to see the result.
Rdo_show(rdoseq)

## we replace usage statements with wrong ones for illustration.
## (note there is an S3 method along with the functions)

dummy_usage <- char2Rdpiece(paste("seq()", "\method{seq}{default}()", "seq.int()", "seq.along()", "seq.len()", sep="\n", "usage")

rdoseq_dummy <- Rdo_replace_section(rdoseq, dummy_usage)
Rdo_show(rdoseq_dummy)  # usage statements are wrong

reprompt(rdoseq_dummy, file = "seqA.Rd")
Rdo_show(tools::parse_Rd("seqA.Rd"))  # usage ok after reprompt

## define function myseq()

myseq <- function(from, to, x){
  if(to < 0) {
    seq(from = from, to = length(x) + to)
  } else seq(from, to)
}

## we wish to describe myseq() along with seq();
## it is sufficient to put myseq() in the usage section
## and let reprompt() do the rest

rdo2 <- Rdo_modify_simple(rdoseq, "myseq()", "usage")
reprompt(rdo2, file = "seqB.Rd")  # updates usage of myseq

## show the rendered result:
Rdo_show(tools::parse_Rd("seqB.Rd"))

## Run this if you wish to see the Rd file:
## file.show("seqB.Rd")

reprompt(infile = "seq.Rd", filename = "seq2.Rd")
reprompt(infile = "seq2.Rd", filename = "seq3.Rd")

## Rd objects for installed help may need some tidying for human editing.
#hseq_inst <- help("seq")
#rdo <- utils::.getHelpFile(hseq_inst)
rdo <- Rdo_fetch("seq", "base")
rdo <- Rdpack::.Rd_tidy(rdo)  # tidy up (e.g. insert new lines for human readers)

reprompt(rdo)  # rdo and rdoseq are equivalent
all.equal(reprompt(rdo), reprompt(rdoseq))  # TRUE

## clean up
unlink("seq.Rd")  # remove temporary files
unlink("seq2.Rd")
unlink("seq3.Rd")
unlink("seqA.Rd")
unlink("seqB.Rd")
RStudio_reprompt

```r
setwd(cur_wd)  # restore user's working directory
unlink(tmpdir)
```

---

**RStudio_reprompt**  
*Call reprompt based on RStudio editor contents*

**Description**

This function uses the RStudio API to call `reprompt` on either the current help file in the editor, or if a name is highlighted in a `.R` file, on that object.

**Usage**

```r
RStudio_reprompt(verbos$ = TRUE)
```

**Arguments**

- `verbose`  
  If TRUE print progress to console.

**Details**

This function depends on being run in RStudio; it will generate an error if run in other contexts.

It depends on code being in a package that has already been built, installed, and attached. In RStudio, this means you should run “Install and Restart” before running this function.

It is automatically installed into RStudio as an add-in called “Reprompt”. Whether invoked directly or through the add-in, it looks at the file currently being edited in the code editor. If it is an `.Rd` file, it will run `reprompt` on that file.

If it is an R source file, it will look for a selected object name. It queries the help system to find if there is already a help page for that name, and if so, works on that. If not, it will try to create one.

**Value**

NULL, invisibly.

**Author(s)**

Duncan Murdoch

**See Also**

`reprompt`, `prompt`
**S4formals**

*Give the formal arguments of an S4 method*

**Description**

Give the formal arguments of an S4 method.

**Usage**

```r
S4formals(fun, ...)
```

**Arguments**

- `fun` name of an S4 generic, a string, or the method, see Details.
- `...` further arguments to be passed to `getMethod`, see Details.

**Details**

`S4formals` gives the formal arguments of the requested method. If `fun` is not of class `methodDefinition`, it calls `getMethods`, passing on all arguments.

Typically, `fun` is the name of a generic function and the second argument is the signature of the method as a character vector. Alternatively, `fun` may be the method itself (e.g. obtained previously from `getMethod`) and in that case the "dots" arguments are ignored. See `getMethod` for full details and other acceptable arguments.

**Value**

a pairlist, like `formals`

**Note**

Arguments of a method after those used for dispatch may be different from the arguments of the generic. The latter may simply have a "dots" argument there.

todo: there should be a similar function in the "methods" package, or at least use a documented feature to extract it.

**Author(s)**

Georgi N. Boshnakov

**Examples**

```r
require(stats4) # makes plot() S4 generic
S4formals("plot", c(x = "profile.mle", y = "missing"))
m1 <- getMethod("plot", c(x = "profile.mle", y = "missing"))
S4formals(m1)
```
update_aliases_tmp  

Description
Update aliases for methods in Rd objects

Usage
update_aliases_tmp(rdo, package = NULL)

Arguments
rdo an Rd object
package the name of a package, a character string.

Details
This is a quick fix. todo: complete it!

Value
the updated Rd object

Author(s)
Georgi N. Boshnakov

viewRd  

Description
View Rd files in a source package.

Usage
viewRd(infile, type = getOption("help_type"), stages = NULL)

Arguments
infile name of an Rd file, a character string.
type one of "text" or "html"
stages a character vector specifying which stages of the R installation process to imitate. The default, c("build", "install", "render"), should be fine in most cases.
Details

This function can be used to view Rd files from the source directory of a package. The page is presented in text format or in html browser, according to the setting of argument type. The default is `getOption("help_type")`.

Value

the function is used for the side effect of showing the help page in a text help window or a web browser.

Note

Developers with "devtools" can use `viewRd()` instead of `help()` for documentation objects that contain Rd macros, such as `insertRef`, see vignette:

`vignette("Inserting_bibtex_references",package = "Rdpack").`

Author(s)

Georgi N. Boshnakov
Index

* **RdoBuild**
  append_to_Rd_list, 9
  c_Rd, 13
  char2Rdpiece, 11
  list_Rd, 35
  parse_pairlist, 38
  Rdo_append_argument, 54
  Rdo_empty_sections, 56
  Rdo_insert, 63
  Rdo_insert_element, 64
  Rdo_is_newline, 64
  Rdo_macro, 68
  Rdo_modify, 69
  Rdo_modify_simple, 70
  Rdo_set_section, 76
  Rdo_tag, 78
  Rdreplace_section, 83

* **RdoProgramming**
  inspect_Rd, 31
  parse_Rdname, 40
  parse_Rdpiece, 41
  parse_Rdtext, 42
  Rdapply, 50
  Rdo_flatinsert, 59
  Rdo_get_insert_pos, 61
  Rdo_get_item_labels, 62
  Rdo_piecetag, 71
  Rdo_remove_srcref, 72
  rdo_text_restore, 80

* **RdoS4**
  get_sig_text, 21
  inspect_signatures, 32
  inspect_slots, 33
  update_aliases_tmp, 95

* **RdoUsage**
  compare_usage1, 12
  deparse_usage, 14
  format_funusage, 17
  get_usage_text, 23
  inspect_args, 30
  inspect_usage, 34
  parse_usage_text, 43

* **Rd**
  insert_ref, 26
  predefined, 44
  promptPackageSexpr, 46
  promptUsage, 47
  Rd_combo, 84
  Rdo2Rdf, 52
  Rdo_collect_aliases, 55
  Rdo_locate, 65
  Rdo_locate_leaves, 67
  Rdo_reparse, 73
  Rdo_sections, 74
  Rdo_show, 77
  Rdo_tags, 79
  Rdo_which, 81
  rebib, 85
  reprompt, 87

* **bibtex**
  get_bibentries, 19
  makeVignetteReference, 36
  Rdpack_bibstyles, 82
  rebib, 85

* **documentation**
  eredeprompt, 16
  insert_all_ref, 24
  insert_citeOnly, 25
  insert_ref, 26
  promptUsage, 47
  Rdo_fetch, 57
  Rdpack_bibstyles, 82
  viewRd, 95

* **methods**
  S4formals, 94

* **package**
  Rdpack-package, 3

* **programming**
INDEX

get_bibentries, 19
append_to_Rd_list, 9
as.character.f_usage (deparse_usage), 14
c_Rd, 7, 13, 35
char2Rdpiece, 11
citeNatbib, 26
compare_usage1, 18
deparse_usage, 14
deparse_usage1, 18
deparse_usage1 (deparse_usage), 14
eedit, 4
ereprompt, 4, 8, 16, 90, 91
formals, 94
format_funusage, 17
get_bibentries, 19, 27, 85, 86
get_sig_text, 21
get_usage, 7, 8, 39
get_usage (promptUsage), 47
get_usage_text, 23
getMethod, 94
insert_all_ref, 24
insert_citeOnly, 25, 30
insertRef, 26, 26, 86, 87
insertRef (insert_ref), 26
inspect_args, 30, 34
inspect_clmethods (inspect_signatures), 32
inspect_Rd, 31
inspect_Rdbib (rebib), 85
inspect_Rdclass (inspect_Rd), 31
inspect_Rdfun (inspect_Rd), 31
inspect_Rdmethods (inspect_Rd), 31
inspect_signatures, 32
inspect_slots, 33
inspect_usage, 13, 34
is_Rdsecname, 11
is_Rdsecname (Rdo_piecetag), 71
list_Rd, 7, 14, 35
makeVignetteReference, 6, 8, 36
pairlist2f_usage1, 16, 44, 49
pairlist2f_usage1 (parse_pairlist), 38
parse_usage_text (parse_usage_text), 43
parse_pairlist, 38, 49
parse_Rdname, 40
parse_Rdpiece, 7, 41, 43
parse_Rdtext, 41, 42
parse_usage_text, 43
predefined, 44
prompt, 93
promptMethods, 88
promptPackageSexpr, 4, 8, 46, 86, 89
promptUsage, 6, 39, 47
rapply, 50, 51
rattr (Rdapply), 50
Rd_combo, 84
Rdapply, 50
Rdo2Rdf, 52
Rdo_append_argument, 54
Rdo_collect_aliases, 55
Rdo_collect_metadata
(Rdo_collect_aliases), 55
Rdo_comment (Rdo_tag), 78
Rdo_drop_empty (Rdo_empty_sections), 56
Rdo_empty_sections, 56
Rdo_fetch, 57
Rdo_flatinsert, 59
Rdo_flatremove (Rdo_flatinsert), 59
Rdo_get_argument_names, 60
Rdo_get_insert_pos, 61, 63
Rdo_get_item_labels, 60, 62
Rdo_insert, 63
Rdo_insert_element, 64
Rdo_is_newline, 64
Rdo_item (Rdo_macro), 68
Rdo_locate, 65, 75, 81
Rdo_locate_core_section, 66
Rdo_locate_core_section (Rdo_sections), 74
Rdo_locate_leaves, 67
Rdo_macro, 68
Rdo_macro1 (Rdo_macro), 68
Rdo_macro2 (Rdo_macro), 68
Rdo_modify, 69, 71, 76
Rdo_modify_simple, 70
Rdo_newline (Rdo_tag), 78
Rdo_piece_types, 72
Rdo_piece_types (predefined), 44
Rdo_piecetag, 71
INDEX

Rdo_predefined_sections, 72
Rdo_predefined_sections (predefined), 44
Rdo_Rcode (Rdo_tag), 78
Rdo_remove_srcref, 72
Rdo_reparse, 73
Rdo_replace_section (Rdo_modify), 69
Rdo_sections, 66, 74
Rdo_sectype (Rdo_piecetag), 71
Rdo_set_section, 76, 84
Rdo_show, 77
Rdo_sigitem (Rdo_macro), 68
Rdo_tag, 78
Rdo_tags, 79
Rdo_text (Rdo_tag), 78
rdo_text_restore, 80
rdo_top_tags (predefined), 44
Rdo_verb (Rdo_tag), 78
Rdo_which, 79, 81
Rdo_which_tag_eq, 79
Rdo_which_tag_eq (Rdo_which), 81
Rdo_which_tag_in, 79
Rdo_which_tag_in (Rdo_which), 81
Rdpack (Rdpack-package), 3
Rdpack-package, 3
Rdpack_bibstyles, 82
Rdreplace_section, 83
Rdtagapply (Rdapply), 50
rebib, 5, 6, 8, 46, 85
reprompt, 4, 8, 17, 80, 87, 93
RStudio_reprompt, 4, 90, 93
S4formals, 94
update_aliases_tmp, 95

viewRd, 6, 8, 77, 95
vigbib, 6, 8
vigbib (makeVignetteReference), 36