Package 'Rdpack'

August 6, 2017

Type Package

Title Update and Manipulate Rd Documentation Objects

Version 0.4-21

Date 2017-08-04

Author Georgi N. Boshnakov

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

Description Functions for manipulation of Rd objects, including
function reprompt() for updating existing Rd
documentation for functions, methods and classes and
function rebib() for import of references from 'bibtex'
files. There is also a macro for importing 'bibtex'
references which an be used in Rd files and 'roxygen'
comments without importing this package.

Depends R (>= 2.15.0), methods, tools, gbRd

Imports bibtex (>= 0.4.0)

License GPL (>= 2)

LazyLoad yes

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2017-08-06 16:10:58 UTC

R topics documented:

Rdpack-package *Update and Manipulate Rd Documentation Objects*

Description

Functions for manipulation of Rd objects, including function reprompt() for updating existing Rd documentation for functions, methods and classes and function rebib() for import of references from 'bibtex' files. There is also a macro for importing 'bibtex' references which an be used in Rd files and 'roxygen' comments without importing this package.

Details

Package:	Rdpack
Type:	Package
Version:	0.4-21
Date:	2017-08-04
License:	GPL (>= 2)
LazyLoad:	yes
Built:	R 3.3.2; ; 2017-08-05 11:23:33 UTC; windows

Index:

Rdo_insert_element	Insert a new element in an Rd object
Rdo_is_newline	Check if an Rd fragment represents a newline character
Rdo_locate	Find positions of elements in an Rd object
Rdo_locate_leaves	Find leaves of an Rd object using a predicate
Rdo_macro	Format Rd fragments as macros (todo: a baffling title!)
Rdo_modify	Replace or modify parts of Rd objects
Rdo_modify_simple	Simple modification of Rd objects
Rdo_piecetag	Give information about Rd elements
Rdo_predefined_sections	Tables of predefined sections and types of pieces of Rd objects
Rdo_remove_srcref	Remove srcref attributes from Rd objects
Rdo_reparse	Reparse an Rd object
Rdo_sections	Locate the sections in Rd objects
Rdo_set_section	Replace a section in an Rd file
Rdo_show	Convert an Rd object to text and show it
Rdo_tag	Set the Rd_tag of an object
Rdo_tags	Give the Rd tags at the top level of an Rd object
Rdo_which	Find elements of Rd objects for which a condition is true
Rdpack-package	Update and Manipulate Rd Documentation Objects
Rdreplace_section	Replace the contents of a section in one or more Rd files
S4formals	Give the formal arguments of an S4 method
append_to_Rd_list	Add content to the element of an Rd object or fragment at a given position
bibentry_key	Give the key associated with a bibentry element
c_Rd	Concatenate Rd objects or pieces
char2Rdpiece	Convert a character vector to Rd piece
compare_usage1	Compare usage entries for a function
deparse_usage	Convert f_usage objects to text appropriate for usage sections in Rd files
format_funusage	Format the usage text of functions
get_bibentries	Get all references from a Bibtex file
get_sig_text	Produce the textual form of the signatures of available methods for an S4 generic function
get_usage_text	Get the text of the usage section of Rd documentation
insert_ref	Insert bibtex references in Rd and roxygen documentation
inspect_Rd	Inspect and update an Rd object or file
inspect_args	Inspect the argument section of an Rd object
inspect_signatures	Inspect signatures of S4 methods
inspect_slots	Inspect the slots of an S4 class
inspect_usage	Inspect the usage section in an Rd object

list_Rd	Combine Rd fragments
parse_Rdname	Parse the name section of an Rd object
parse_Rdpiece	Parse a piece of Rd source text
parse_Rdtext	Parse Rd source text as the contents of a section
parse_pairlist	Parse formal arguments of functions
parse_text	Parse expressions residing in character vectors
parse_usage_text	Parse usage text
promptPackageSexpr	Generates a shell of documentation for an installed package
promptUsage	Usage text for a function, S3 method or S4 method
rdo_text_restore	~~ Dummy title ~~
rebib	Work with bibtex references in Rd documentation
reprompt	Update the documentation of a topic
update_aliases_tmp	Update aliases for methods in Rd objects

Package Rdpack may help authors of R packages to keep their documentation up to date during development. 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.

Another set of functions is for management of bibliographic references.

The main functions provided by this package are reprompt, promptPackageSexpr, and rebib.

NEW: References are most easily inserted with the help of the macro \insertRef{key}{package}. This facility works in Rd files and roxygen comments and needs only an entry in the DESCRIPTION file of the package, see insertRef and the vignette for details and examples.

reprompt produces a skeleton documentation for the requested object, similarly to functions like prompt, promptMethods, and promptClass. Unlike those functions, 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, reprompt updates things that are generated automatically, leaving manual editing untouched.

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

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

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 reprompt for details.

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 Sexpr's at build time, not with verbatim strings as promptPackage does.

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

```
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.

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:

```
\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.

The Bibtex source for the references is by default a file "REFERENCES.bib" located in the root of the package installation folder.

Note that there are other, more sophisticated, approaches to keeping documentation and code in synchron, for example package roxygen (todo: give references here).

It can hardly get simpler than using a single function, reprompt, with one argument (the doc file or the object to be updated) to update all sorts of Rd documentation files. Some may find that this is all they need and not bother with the rest of this documentation.

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

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

parse_Rdpiece is 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.

Rdo_set_section can be used to set a section, such as "\author".

The remaining functions in the package are for programming with Rd objects (and probably many of them should not be exported).

Note

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

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

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

References

Francois R (2014). *bibtex: bibtex parser*. R package version 0.4.0, https://CRAN.R-project.org/package=bibtex.

Murdoch D (2010). "Parsing Rd files." https://developer.r-project.org/parseRd.pdf.

See Also

`reprompt`, `promptPackageSexpr`, `rebib`

Examples

```
# The examples below show typical use.
# Simply insert the path to your Rd file, the name of the object or the
# object itself.
# For executable examples see main functions,
# reprompt and promptPackageSexpr

## Not run:
# update the doc. from the Rd source and save myfun.Rd
#     in the current directory (like prompt)
reprompt(infile="path/to/mypackage/man/myfun.Rd")

# update doc of myfun() from the installed doc (if any);
#     if none is found, create it like prompt
reprompt("myfun")
reprompt(myfun)      # same

# update doc. for S4 methods from Rd source
reprompt(infile="path/to/mypackage/man/myfun-methods.Rd")

# update doc. for S4 methods from installed doc (if any);
#     if none is found, create it like promptMethods
reprompt("myfun", type = "methods")
reprompt("myfun-methods")  # same
```

```
# update doc. for S4 class from Rd source
reprompt(infile="path/to/mypackage/man/myclass-class.Rd")

# update doc. of S4 class from installed doc.
#     if none is found, create it like promptClass
reprompt("myclass-class")
reprompt("myclass", type = "class")  # same


# create a skeleton "mypackage-package.Rd"
promptPackageSexpr("mypackage")

# update the references in "mypackage-package.Rd"
rebib(infile="path/to/mypackage/man/mypackage-package.Rd", force=TRUE)

## End(Not run)
```

`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"))
iusage <- which(tools:::RdTags(rdoseq) == "\\usage")

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

# then append a new usage statement, in this case for another function
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 the doc.
#     reports new argument "nvec" and updates the Rd object.
      # notice that the usage statement of sequence() is corrected
      # and an item for argument nvec is created.
reprompt(rdoseq3, filename=NA)
```

---

bibentry_key *Give the key associated with a bibentry element*

Description

Give the key associated with a bibentry element

Usage

```
bibentry_key(x)
```

Arguments

x	a single bibentry element.

Details

This is a convenience function to get the "key" attribute of a reference represented by a bibentry element.

There should be a better way to do this, I must be missing something here.

Value

a character string

Author(s)

Georgi N. Boshnakov

`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
char2Rdpiece("graphics","keyword")

# an element suitable to be put in a "usage" section
char2Rdpiece("log(x, base = exp(1))", "usage")

#
char2Rdpiece("Give more examples for this function.", "Todo", force.sec = TRUE)
```

`compare_usage1` *Compare usage entries for a function*

Description

Compare usage entries for a function.

Usage

```
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.

`added_argnames`
: names of arguments in `ucur` but not in `urdo`.

`removed_argnames`
: names of arguments in `urdo` but not in `ucur`.

`names_unchanged_defaults`
: names of arguments whose defaults are the same.

`rdo_usage` a copy of `urdo`.

`cur_usage` a copy of `ucur`.

`alias` alias of the name of the object, see 'Details'.

Author(s)

Georgi N. Boshnakov

See Also

`inspect_usage`

---

c_Rd *Concatenate Rd objects or pieces*

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 `...` 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.

Value

An Rd object or a list whose attribute "Rd_tag" is set as descibed in 'Details'

Author(s)

Georgi N. Boshnakov

Examples

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

# The following are equivalent
# TODO: replace Rdo_empty() below with a function from Rdpack
#       and uncomment
# b1 <- c_Rd(Rdo_empty(), list(a1), list(a2), list(a3), list(a4))
# c1 <- c_Rd(Rdo_empty(), list(a1, a2, a3, a4))
# d1 <- c_Rd(Rdo_empty(), list(a1, a2), list(a3, a4))
# identical(c1,b1)
# identical(c1,d1)
# Rdo_show(b1)

# insert a newline
# TODO: replace Rdo_empty() below with a function from Rdpack
#       and uncomment
# d1n <- c_Rd(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 )
```

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

```
deparse_usage(x)
deparse_usage1(x, width = 72)
## S3 method for class 'f_usage'
as.character(x, ... )
```

Arguments

x	an object from class f_usage
width	maximal width of text on a line
...	ignored; this argument is present for consistency with the generic as.character

Details

deparse_usage1 is internal function. For users as.character is more convenient.

Value

For deparse_usage1 and as.character.f_usage, a named character vector of length one (the name is the function name).

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

Author(s)

Georgi N. Boshnakov

See Also

pairlist2f_usage and pairlist2f_usage1

Examples

```
cur_wd <- getwd()
setwd(tempdir())

# as for prompt() the default to save in current dir as "seq.Rd".
fnseq <- reprompt(seq)

# let's parse the saved Rd file
```

```
rdoseq <- parse_Rd(fnseq)

# the usage of 'seq' has several entries, parse them all

ut <- get_usage_text(rdoseq)
cat(ut,"\n")

utp <- parse_usage_text(ut)

# format some of them
as.character(utp[[1]])
deparse_usage1(utp[[1]])  # same

cat(deparse_usage1(utp[[2]]))
cat(as.character(utp[[2]]))   # same


unlink(fnseq)
```

`format_funusage` *Format the usage text of functions*

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")
```

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.

Details

This function parses the specified file using `read.bib` from package "bibtex" and sets its `names` attribute to the keys of the bib elements.

`bibfile` should normally be the base name of the Bibtex file. Calling `get_bibentries` without any `...` 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`.

Value

a bibentry object

Author(s)

Georgi N. Boshnakov

`get_sig_text` *Produce the textual form of the signatures of available methods for an S4 generic function*

Description

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".

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 funtions 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

```
require("stats4") # to ensure the presence of S4 methods from
                  # at least one package other than "methods"

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

# this will find all methods for "show" in currently loaded packages
get_sig_text(rdo)

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

# this is also fine 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")

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

get_sig_text(rdo)
get_sig_text(rdo, package = "stats4")
```

---

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

```
# get the Rd object documenting Rdo_macro
h <- help("Rdo_macro")
rdo <- utils:::.getHelpFile(h)

ut <- get_usage_text(rdo)

# in this case rdo describes other objects
# and their usage entries are returned, as well.
cat(ut,sep = "\n")
```

---

insert_ref *Insert bibtex references in Rd and roxygen documentation*

Description

Include references from bibtex files into R documentation using a macro. Function insert_ref is called behind the scenes at package built time.

Usage

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

Arguments

key	the bibtex key of the reference, a character string.
package	the package in which to look for the the bibtex file.
...	further arguments to pass on to `bibtex::read.bib`

Details

`insert_ref` extracts a 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`.

To insert references from a bibtex file in your package:

1. add the following line to file 'DESCRIPTION':
   RdMacros: Rdpack
   (If the field 'RdMacros' is already present, add Rdpack to the list on that line.)
2. Create file `"REFERENCES.bib"` in subdirectory `"inst/"` of your package and put your bibtex references in it.

Then you can insert references in the documentation with \insertRef{key}{package}, where key is the bibtex key of the reference and package is an R package.

This works in manually written Rd files and in roxygen documentation chunks. The references will appear in the place where you put the macro, usually in a dedicated references section (\references in Rd files, @references in roxygen chunks).

Argument 'package' can be any installed R package, not necessarily the current one. This means that you don't need to copy references from other packages to your `"REFERENCES.bib"` file. This works for packages that have `"REFERENCES.bib"` in their installation directory and for the default packages.

For example, the references in the references section of this help page are generated by the following lines in the Rd file:

```
\insertRef{Rpack:bibtex}{Rdpack}

\insertRef{R}{bibtex}
```

A roxygen documentation chunk might look like this:

```
#' \@references
#' \insertRef{Rpack:bibtex}{Rdpack}
#'
#' \insertRef{R}{bibtex}
```

The references are processed when the package is built. So, there is *no need* to depend/import/suggest package "Rdpack", it only needs to be installed on your machine.

Value

for `insert_ref`, a character string

Author(s)

Georgi N. Boshnakov

References

Francois R (2014). *bibtex: bibtex parser*. R package version 0.4.0, https://CRAN.R-project.org/package=bibtex.

R Development Core Team (2009). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, https://www.R-project.org.

See Also

`rebib`

Examples

```
insert_ref("R", package = "bibtex")
```

---

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 depening on the type of the topic.

`inspect_Rdfun` 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_Rdfun` understands the syntax for S3 methods and S4 methods used in "usage" sections, as well.

`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 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 the object.

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 *Combine Rd fragments*

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 mixtue of character vactors 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",
```

```
              Rd_class=TRUE )

Rdo_show(u1)

# call reprompt to fill the arguments section
#    (and correct the usage)

fn <- tempfile("dummyfun", fileext="Rd")
reprompt(dummyfun, filename=fn)

# check that the result can be parsed and show it.
Rdo_show(parse_Rd(fn))

unlink(fn)
```

`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 or a list of pairlists, see 'Details'.
`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.

`x` is a single pairlist object for `parse_pairlist` and `pairlist2f_usage1`.

The pairlist object is parsed into a list whose 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 component (so, it may be of length zero).

`pairlist2f_usage1` adds components `name` (function name), `S3class`, `S4sig` and `infix`. `S3class` is set for S3 methods, `S4sig` is the signature of an S4 method (as used in Rd macro \S4method). `infix` is `TRUE` for the rare occations of usages of infix operators. The result is given class "f_usage". This class has a method for `as.character` which generates a text suitable for inclusion in Rd documentation.

Value

For `parse_pairlist`, a list with the following components:

`argnames`	names of arguments, a character vector
`defaults`	a named character vector containing the default values, converted to character strings.

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.
`S4sig`	S4 signature.
`infix`	a logical value, `TRUE` for infix operators.

Author(s)

Georgi N. Boshnakov

See Also

`promptUsage`

Examples

```
parse_pairlist(formals(lm))
```

---

parse_Rdname *Parse the name section of an Rd object*

Description

Parse the name section of an Rd object.

Usage

```
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

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

parse_Rdname(u1)

u2 <- list_Rd(name = "dummyclass-class", alias = "Dummyclass",
              title = "Class dummyclass",
              description = "Objects and methods for something.",
              Rd_class=TRUE )

parse_Rdname(u2)
```

`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 it when presenting help pages in text format.

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",
              author = "A. Author",
              Rd_class=TRUE )
fn <- tempfile("dummyfun", fileext="Rd")
reprompt(dummyfun, filename=fn)
rdo <- parse_Rd(fn)

# let's prepare a new item
rd <- "\\item{...}{further arguments to be passed on.}"
newarg <- parse_Rdtext(rd, section = "\\arguments")

# now append 'newarg' to the arguments section of rdo
iarg <- which(tools:::RdTags(rdo) == "\\arguments")
rdoa <- append_to_Rd_list(rdo, newarg, iarg)

Rdo_show(rdoa)

# for arguments and other frequent tasks there are
#     specialised functions
rdob <- Rdo_append_argument(rdo, "...", "further arguments to be passed on.")


Rdo_show(reprompt(rdob))
```

```
# todo: Rdo_show(rdob) for some reason does not show the arguments.
#       investigate! Rdo_show uses Rd2txt. Is it possible that the
#       latter needs srcref's in the Rd object? They are only refreshed
#       Rd_parse is called.


unlink(fn)
```

`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 `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{" is inserted at the begiinning of `text` and a closing "}" 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

`parse_text` *Parse expressions residing in character vectors*

Description

Parse expressions residing in character vectors.

Usage

```
parse_text(text, ..., keep = TRUE)
```

Arguments

`text`	the text to parse, normally a character vector but can be anything that `parse` accepts for this artgument.
`...`	additional arguments to be passed on to `parse`.
`keep`	required setting for option keep.source, see details.

Details

This is like `parse(text=text,...)` with the additional feature that if the setting of option "keep.source" is not as requested by argument `keep`, it is set to `keep` before calling `parse` and restored afterwards.

Value

an expression representing the parsed text, see `link{parse}` for details

Note

The usual setting of option "keep.source" in interactive sessions is TRUE. However, in 'R CMD check' it is FALSE.

As a consequence, examples from the documentation may run fine when copied and pasted in an R session but (rightly) fail 'R CMD check', when they depend on option "keep.source" being `TRUE`.

Author(s)

Georgi N. Boshnakov

See Also

parse

parse_usage_text *Parse usage text*

Description

Parse usage text.

Usage

```
parse_1usage_text(text)
parse_usage_text(text)
```

Arguments

text	content of the usage section of an Rd object, a character vector.

Details

parse_usage_text does some preprocessing of text then calls parse_1usage_text for each usage statement.

The preprocessing changes "..." to "..." and converts S3 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.

Value

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

Author(s)

Georgi N. Boshnakov

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

Description

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

Usage

```
Rdo_predefined_sections

Rdo_piece_types

rdo_top_tags
```

Details

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

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

These 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

The format of `Rdo_predefined_sections` is:

name	VERB	description	TEXT
alias	VERB	examples	RCODE
concept	TEXT	usage	RCODE
docType	TEXT	Rdversion	VERB
title	TEXT	synopsis	VERB
name	VERB	section	TEXT
alias	VERB	arguments	TEXT
concept	TEXT	keyword	TEXT
docType	TEXT	note	TEXT
title	TEXT	format	TEXT
name	VERB	source	TEXT
alias	VERB	details	TEXT
concept	TEXT	value	TEXT
docType	TEXT	references	TEXT
title	TEXT	author	TEXT
name	VERB	seealso	TEXT

name	VERB	|	description	TEXT
alias	VERB	|	examples	RCODE
concept	TEXT	|	usage	RCODE
docType	TEXT	|	Rdversion	VERB
title	TEXT	|	synopsis	VERB
name	VERB	|	section	TEXT
alias	VERB	|	arguments	TEXT
concept	TEXT	|	keyword	TEXT
docType	TEXT	|	note	TEXT
title	TEXT	|	format	TEXT
name	VERB	|	source	TEXT
alias	VERB	|	details	TEXT
concept	TEXT	|	value	TEXT
docType	TEXT	|	references	TEXT
title	TEXT	|	author	TEXT
name	VERB	|	seealso	TEXT

The format of `Rdo_piece_types` is:

name	VERB	alias	VERB	concept	TEXT
docType	TEXT	title	TEXT	description	TEXT
examples	RCODE	usage	RCODE	Rdversion	VERB
synopsis	VERB	Sexpr	RCODE	RdOpts	VERB
code	RCODE	dontshow	RCODE	donttest	RCODE
testonly	RCODE	dontrun	VERB	env	VERB
kbd	VERB	option	VERB	out	VERB
preformatted	VERB	samp	VERB	special	VERB
url	VERB	verb	VERB	deqn	VERB
eqn	VERB	renewcommand	VERB	newcommand	VERB

name	VERB	| alias	VERB	| concept	TEXT
docType	TEXT	| title	TEXT	| description	TEXT
examples	RCODE	| usage	RCODE	| Rdversion	VERB
synopsis	VERB	| Sexpr	RCODE	| RdOpts	VERB
code	RCODE	| dontshow	RCODE	| donttest	RCODE
testonly	RCODE	| dontrun	VERB	| env	VERB
kbd	VERB	| option	VERB	| out	VERB
preformatted	VERB	| samp	VERB	| special	VERB
url	VERB	| verb	VERB	| deqn	VERB
eqn	VERB	| renewcommand	VERB	| newcommand	VERB

The value of `rdo_top_tags` is:

name	Rdversion	docType	alias	encoding
concept	title	description	usage	format
source	arguments	details	value	references
section	note	author	seealso	examples
keyword	ifdef	ifndef	newcommand	renewcommand
COMMENT	TEXT			

`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 'promptPackage' 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 refernces 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 *Usage text for a function, S3 method or S4 method*

Description

Obtains the usage text for a function, S3 method or S4 method for inclusion in the usage section of Rd documentation.

Usage

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

promptUsage(..., usage)
```

Arguments

`...`	for `promptUsage`, arguments to be passed on to `get_usage`; for `get_usage`, currently not used.
`usage`	an usage object, see Details.
`object`	a function object or a character name of one.
`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`).

Details

Argument `usage` could probably only be useful in programming when the usage text has been obtained (or generated) programmatically. `usage` may be an "f_usage" object obtained e.g. from `get_usage()`.

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.

Value

a character string or an object of S3 class "f_usage", see `pairlist2f_usage` 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

```
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, ...)")

# but this gives the args of the generic:  "\method{droplevels}{data.frame}(x, ...)"
get_usage(droplevels,S3class="data.frame")
```

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 rapply is assumed to have identical structure to the original object. This means that argument how of rapply must not be "unlist" and normally will be "replace". Rdtagapply gives sensible default values for classes and how. See the documentation of rapply for details and the possible choices for classes, how or other arguments passed to it via ....

Rdtagapply is a convenience variant of Rdapply for the common task of modifying or examining only elements with a given Rd_tag attribute. Since the Rd equation macros \eqn and \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 \eqn and \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.

rattr is an auxilliary 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.

Value

For Rdapply, an Rd object with some of its leaves replaced as specified above.

For rattr, the object x with attributes of any list elements of it set to the corresponding attributes of y.

Note

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

Author(s)

Georgi N. Boshnakov

See Also

rapply

Examples

```
# 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)
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(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

```
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 backslahes 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 the `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 backslahes (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

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

# # this output "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

```
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

Examples

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

dottext <- "further arguments to be passed on."

# rdo2 <- Rdo_append_argument(rdo, "...", dottext, create = TRUE)
rdo2 <- Rdo_append_argument(rdo, "...", dottext, create = TRUE)
rdo2 <- Rdo_append_argument(rdo2, "z", "a numeric vector")
Rdo_show(reprompt(rdo2))

# todo: Rdo_show(rdob) for some reason does not show the arguments.
#       investigate! Rdo_show uses Rd2txt. Is it possible that the
#       latter needs srcref's in the Rd object? They are only refreshed
#       Rd_parse is called.


unlink(fn)
```

Rdo_collect_metadata *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 which collect the metadata from sections 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".

Value

a named character vector, as described in details.

Author(s)

Georgi N. Boshnakov

See Also

tools:::.Rd_get_metadata

Examples

```
## Not run:
# this needs "timezones.Rd".

infile <- file.path(R.home(), "src/library/base/man/timezones.Rd")
rd <- parse_Rd(infile)

# The functions described here handle also "ifdef" and similar
# directives. This contains an element windows = "TZDIR"
Rdo_collect_aliases(rd)

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

## End(Not run)
```

Rdo_empty_sections *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)
```

Arguments

rdo	an Rd object or Rd source text.
with_bs	if TRUE return the section names with the leading backslash.
sec	not used

Details

The function checkRd is used to determine which sections are empty.

Value

For Rdo_empty_sections, the names of the empty sections as a character vector.

For Rdo_drop_empty, the Rd object stripped from empty sections.

Author(s)

Georgi N. Boshnakov

Examples

```
dummyfun <- function(x) x
rdo8 <- list_Rd(name = "Dummyname", alias = "dummyfun",
              title = "Dummy title", description = "Dummy description",
              usage = "dummyfun(x,y)",
              value = "numeric vector",
              author = "",
              details = "",
              note = "",
              Rd_class=TRUE )

Rdo_empty_sections(rdo8)        # "details" "note"    "author"

rdo8a <- Rdo_drop_empty(rdo8)
Rdo_empty_sections(rdo8a)       # character(0)
```

`Rdo_flatinsert` *Insert content in an Rd fragment*

Description

Insert content in an Rd fragment.

Usage

```
Rdo_flatinsert(rdo, val, pos, before = TRUE)

Rdo_flatremove(rdo, from, to)
```

Arguments

`rdo`	an Rd object.
`val`	the value to insert.2
`pos`	poistion.
`before`	if TRUE insert pushing the elemnt at pos forward.
`from`	beginning of the region to remove.
`to`	end of the region to remove.

Details

`Rdo_flatinsert` inserts `val` at position `pos`, effectively by concatenation.

`Rdo_flatremove` 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 Rdo 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

```
Rdo_get_insert_pos(rdo, 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

```
h <- help("Rdo_macro")
rdo <- utils:::.getHelpFile(h)

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

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

`Rdo_get_item_labels` *~~ Dummy title ~~*

Description

~~ Dummy description ~~

Usage

```
Rdo_get_item_labels(rdo)
```

Arguments

`rdo`	an Rd object.

Value

a character vector

Author(s)

Georgi N. Boshnakov

`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

```
Rdo_insert(rdo, val, newline = TRUE)
```

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

```
Rdo_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

```
Rdo_is_newline(rdo)
```

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

`Rdo_locate` *Find positions of elements in an Rd object*

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="..." 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 charater 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 <- 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.
newline	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

```
Rdo_modify(rdo, val, create = FALSE, replace = FALSE, top = TRUE)

Rdo_replace_section(rdo, val, create = FALSE, replace = 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 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 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 `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

```
# a <- 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")
```

`Rdo_modify_simple` *Simple modification of Rd objects*

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`.

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 charater vectors `Rdo_piece_types` and `Rdo_predefined_sections`.

Author(s)

Georgi N. Boshnakov

See Also

`Rdo_piece_types` and `Rdo_predefined_sections`

Examples

```
Rdo_piecetag("eqn")  # ==> "VERB"
Rdo_piecetag("code") # ==> "RCODE"

Rdo_sectype("usage") # ==> "RCODE"
Rdo_sectype("title") # ==> "TEXT"

Rdo_sectype("arguments")
```

Rdo_remove_srcref *Remove srcref attributes from Rd objects*

Description

Removes srcref attributes from Rd objects.

Usage

```
Rdo_remove_srcref(rdo)
```

Arguments

rdo	an Rd object

Details

`srcref` attrbutes (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 removes traverses he 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*

Description

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 athere is a problem if the tempfile disappears.)

Examples

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

dottext <- "further arguments to be passed on."

rdo2 <- Rdo_append_argument(rdo, "...", 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)
```

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.

Details

Rdo_sections locates all sections allowed 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. **todo:** maybe create a convenience function for this case or (better) introduce an argument to request dropping the outer list?

Value

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

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

Note

I wrote Rdo_sections after most of the core functionality was tested. Currently this function is underused—it can replace a number of internal functions,

Author(s)

Georgi N. Boshnakov

See Also

Rdo_locate

`Rdo_set_section` *Replace a section in an Rd file*

Description

Replace a section in an Rd file.

Usage

```
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:

```
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.

Note

This is probably the only function in the package that changes an Rd file in-place.

Author(s)

Georgi N. Boshnakov

Examples

```
dummyfun <- function(x) x

fn <- tempfile("dummyfun", fileext="Rd")
reprompt(dummyfun, filename=fn)
Rdo_show(parse_Rd(fn))

Rdo_set_section("Georgi N. Boshnakov", "author", fn)
Rdo_show(parse_Rd(fn))

unlink(fn)
```

Rdo_show *Convert an Rd object to text and show it*

Description

Convert an Rd object to text and show it with file.show.

Usage

```
Rdo_show(rdo)
```

Arguments

rdo	an Rd object

Value

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

Author(s)

Georgi N. Boshnakov

Rdo_tag *Set the Rd_tag of an object*

Description

Set the Rd_tag of an object.

Usage

```
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.

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 *~~ Dummy title ~~*

Description

~~ Dummy description ~~

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 an auxilliary function.

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

This 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. (Note that, in general, it is not possible to restore the original Rd file from the Rd object due to the specifications of the Rd format. However, equivalent things for the computer are not necessarily equally pleasant for humans.

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

`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

```
##---- Should be DIRECTLY executable !! ----
```

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

```
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 mpdofying 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)
```

Rd_combo *Manipulate a number of Rd files*

Description

Manipulate a number of Rd files.

Usage

```
Rd_combo(rd, f, ..., .MORE)
```

Arguments

rd	names of Rd files, a character vector.
f	function to apply, see Details.
...	further arguments to pass on to f.
.MORE	another function to be applied for each file to the result of f.

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

```
## Not run:
rdnames <- dir(path = "./man", pattern=".*[.]Rd$", full.names=TRUE)
Rd_combo(rdnames, reprompt)
for(nam in rdnames) try(reprompt(nam))
for(nam in rdnames) try(reprompt(nam, sec_copy=FALSE))

## End(Not run)
```

rebib *Work with bibtex references in Rd documentation*

Description

Work with bibtex references in Rd documentation.

Usage

```
rebib(infile, outfile, ...)

inspect_Rdbib(rdo, force = FALSE, ...)
```

Arguments

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 is used by package Rdpack for incorporation of bibliographic references from BibTeX files.

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 exmple 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 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

Examples

```
## Not run:
# update references in all Rd files in the package's 'man' directory
#
rdnames <- dir(path = "./man", pattern=".*[.]Rd$", full.names=TRUE)
lapply(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, methods or slots, as appropriate. This is an extention 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, ...)
```

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, see 'Details'.
type	type of topic, e.g. "methods", see 'Details'.
package	package name; document only objects defined in this package.
methods	used for documentation of S4 methods only, syntax is as the corresponding argument in promptMethods, see Details.
verbose	if TRUE print messages on the screen.
filename	name of the file for the generated Rd content.
...	currently not used
sec_copy	if TRUE copy Rd contents of unchanged sections in the result, see Details.

Details

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

```
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 of the prompt family.

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

```
reprompt("myfun-methods")

reprompt("[<--methods", package = "mypackage")             # or
reprompt("[<-", type = "methods", package = "mypackage")

reprompt("myclass", type = "class", package = "mypackage") # or
reprompt("myclass-class", package = "mypackage")
```

For "myfun" above the "package" argument may not matter but for the replacement method it almost certainly will.

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.

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 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.

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, not literal strings.

If argument sec_copy is TRUE, reprompt will, effectively, copy the contents of (some) unchanged sections, thus ensuring that they are exactly the same as in the original.

In general, if an Rd object is obtained by parsing an Rd file, then converting the object back to the Rd format may not reproduce the original file exactly (some escape sequences may change). Even though the new version should be functionally equivalent to the original, this may not be desirable. For example, if such changes creep into the Details section (which reprompt never changes) the user may be annoyed or worried. Notice that this is not really a cause of concern. Moreover, once you replace the old file with the one one produced by reprompt, it will not change again.

Value

if filename is a character string or NULL, the name of the file to which the updated shell was written.

Otherwise, the 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 were you expect them to be. (This may be addressed in a future revision.)

A common error when supplying a file name for reprompt is to forget to name the argument, e.g. reprompt(infile="./Rdpack/man/reprompt.Rd"). Otherwise the argument is taken to be the name of a function. Currently a convenience convention 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. However, this feature should not be relied upon as it may be withdrawn since it is dubious and only partially solves the problem with forgetting the name of the argument.

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...).

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.

Todo: add an S3class argument (or equivalent) to allow specification of S3 methods for `reprompt`. Note that S3 methods present in usage statements are already processed. Also, giving the full name of an S3 method as `fun.class` works, as well (e.g. `reprompt(predict.lm)`).

Author(s)

Georgi N. Boshnakov

Examples

```
# note: usage of reprompt() is simple.
#       the examples below are bulky because they
#       simulate various usage scenarios.
#
# change to a temporary directory to avoid clogging up user's
cur_wd <- getwd()
setwd(tempdir())

# as for prompt() the default 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
rdoseq <- parse_Rd(fnseq)

# replace usage statements with wrong ones for illustration.
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)  # wrong usage

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

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

# add a description of  sequence()
rdo2 <- Rdo_modify_simple(rdoseq, "myseq()", "usage")

reprompt(rdo2, file = "seqB.Rd")  # updates usage of myseq
Rdo_show(parse_Rd("seqB.Rd"))     # and add an entry for 'x' in 'arguments'




# as for prompt() the default is to save in current dir as "seq.Rd".
fnseq <- reprompt(seq)
```

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

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 <- Rdpack:::.Rd_tidy(rdo)          # tidy up (e.g. insert new lines
                                       #          for human readers)

reprompt(rdo) # same as rdoseq;



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

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

`S4formals` *Give the formal arguments of an S4 method*

Description

Give the formal arguments of an S4 method.

Usage

```
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 as 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 ... 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 ... 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

```
require(stats4)
S4formals("plot",c(x="profile.mle", y="missing"))

m1 <- getMethod("plot",c(x="profile.mle", y="missing"))
S4formals(m1)
```

---

update_aliases_tmp *Update aliases for methods in Rd objects*

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

Index