

Package ‘roxygen’

November 12, 2009

Version 0.1-2

License GPL (>= 2)

Description A Doxygen-like in-source documentation system for Rd, collation, namespace and callgraphs.

Title Literate Programming in R

Author Peter Danenberg <pcd@roxygen.org>, Manuel Eugster
<Manuel.Eugster@stat.uni-muenchen.de> with contributions from Hadley
Wickham <hadley@rice.edu>

Maintainer Peter Danenberg <pcd@roxygen.org>

URL <http://roxygen.org>

Depends digest

Suggests Rgraphviz (>= 1.19.2), tools (>= 2.9.1)

Collate 'functional.R' 'list.R' 'roxygen.R' 'string.R' 'memoize.R' 'parse.R' 'parseS4.R' 'roclet.R'
'callgraph.R' 'description.R' 'collate.R' 'namespace.R' 'Rd.R' 'Rdmerge.R' 'Rdapi.R'
'Rdtank.R' 'Rd2.R' 'roxygenize.R'

Repository CRAN

Date/Publication 2009-11-12 11:39:59

R topics documented:

roxygen-package	4
assign.parent	5
caar	5
cadar	6
caddr	6
cadr	7
car	7
cat.description	8

cdddr	8
cddr	9
cdr	9
Compose	10
copy.dir	10
Curry	11
debug	11
description.dependencies	12
DESCRIPTION.FILE	12
DOC.DIR	12
expression.from.partitum	13
first.non.null	13
guess.name	14
Identity	14
include	15
INST.DIR	15
is.assignment	15
is.even	16
is.function.definition	16
is.nil	17
is.null.string	17
is.odd	18
LINE.DELIMITER	18
make.callgraph.roclet	18
make.collate.roclet	19
make.description.parser	20
make.namespace.roclet	21
make.Rd.roclet	22
make.Rd2.roclet	25
make.roclet	26
MAN.DIR	26
MATTER	27
NAMESPACE.FILE	27
Negate	27
nil	27
NIL.STRING	28
noop.description	28
nwords	28
pairwise	29
parse.assignee	29
parse.call	30
parse.default	30
parse.description	31
parse.description.file	31
parse.description.text	32
parse.element	32
parse.error	33
parse.file	33

parse.files	34
parse.formals	34
parse.message	35
parse.name	35
parse.name.description	36
parse.preref	36
parse.ref	37
parse.ref.list	37
parse.ref.preref	38
parse.ref.srcref	38
parse.refs	39
parse.srcref	39
parse.text	40
parse.toggle	40
parse.value	41
parse.warning	41
parser.default	42
parser.preref	42
parser.srcref	43
preorder.flatten.expression	43
preorder.walk.expression	44
preref.parsers	44
prerefs	45
R.DIR	45
Reduce.paste	45
register.parser	46
register.parsers	46
register.preref.parser	47
register.preref.parsers	47
register.srcref.parser	48
register.srcref.parsers	48
roxygen	49
ROXYGEN.DIR	49
roxygenize	49
SPACE	50
src.lines	50
srcref.parsers	50
strcar	51
strcdr	51
strcons	52
strmap	52
substr.regexpr	53
TAG.DELIMITER	53
trim	53
trim.left	54
trim.right	54
word.ref	55
zip	55

zip.c	56
zip.list	56

Index	57
--------------	-----------

roxygen-package *Literate Programming in R*

Description

Roxygen is a Doxygen-like documentation system for R; allowing in-source specification of Rd files, collation and namespace directives.

Details

Package:	Roxygen
Type:	Package
Version:	0.1-1
Date:	2008-08-25
License:	GPL (>= 2)
LazyLoad:	yes

Roxygen is run on a package (hereafter <package>) by R CMD `roxygen <package>` or Rcmd `roxygen.sh <package>` on Windows. By default, it creates a directory '`<package>.roxygen`' with the complete package cum populated Rd files, 'NAMESPACE', etc.; but can also operate destructively on the package itself with the '-d' option.

See the vignette ('`roxygen.pdf`') or manual ('`roxygen-manual.pdf`') for details.

Author(s)

Peter Danenberg <`pcd@roxygen.org`>, Manuel Eugster <`Manuel.Eugster@stat.uni-muenchen.de`>

Maintainer: Peter Danenberg <`pcd@roxygen.org`>

See Also

See `make.Rd.roclet`, `make.namespace.roclet`, `make.collate.roclet`, `make.callgraph.roclet` for an overview of roxygen tags.

See `roxygenize` for an alternative to 'R CMD roxygen'.

Examples

```
## To process a package in `pkg`, run `R CMD roxygen pkg`; or:
## Not run: roxygenize('pkg')
```

assign.parent	<i>Assign a variable in the parent environment when «-...</i>
---------------	---

Description

Assign a variable in the parent environment when <<- doesn't seem to work.

Usage

```
assign.parent(var, value, env)
```

Arguments

var	string of the variable to assign
value	value to be assigned
env	environment of the assignment (environment())

Value

NULL

caar	<i>Composite car/cdr...</i>
------	-----------------------------

Description

Composite car/cdr

Usage

```
caar(list)
```

Arguments

list	the list from which to extract
------	--------------------------------

Value

The extracted elements

cadar

Composite car/cdr...

Description

Composite `car/cdr`

Usage

`cadar(list)`

Arguments

`list` the list from which to extract

Value

The extracted elements

caddr

Composite car/cdr...

Description

Composite `car/cdr`

Usage

`caddr(list)`

Arguments

`list` the list from which to extract

Value

The extracted elements

`cadr`*Composite car/cdr...*

DescriptionComposite `car/cdr`**Usage**`cadr(list)`**Arguments**`list` the list from which to extract**Value**

The extracted elements

`car`*First element of a list...*

Description

First element of a list

Usage`car(list)`**Arguments**`list` the list to first**Value**

The first element

`cat.description` *Print the field-value pair to a given file or standard out.*

Description

Print the field-value pair to a given file or standard out.

Usage

```
cat.description(field, value, file="")
```

Arguments

<code>field</code>	the field to be printed
<code>value</code>	the value to be printed
<code>file</code>	the file whither to print (a blank string being standard out)

Value

NULL

`cdddd` *Composite car/cdr..*

Description

Composite `car/cdr`

Usage

```
cdddd(list)
```

Arguments

<code>list</code>	the list from which to extract
-------------------	--------------------------------

Value

The extracted elements

`cddr`*Composite car/cdr...*

DescriptionComposite `car/cdr`**Usage**`cddr(list)`**Arguments**`list` the list from which to extract**Value**

The extracted elements

`cdr`*Return elements after the first of a list.*

Description

Return elements after the first of a list.

Usage`cdr(list)`**Arguments**`list` the list from which to extract**Value**The elements after the first, or `nil` if only one

 Compose

Compose an arbitrary number of functions.

Description

Compose an arbitrary number of functions. My Happy Hacking keyboard gave out during the writing of this procedure; moment of silence, please.

Usage

```
Compose (...)
```

Arguments

```
...           the functions to be composed
```

Value

A composed function

 copy.dir

Recursively copy a directory thither; optionally unlinking...

Description

Recursively copy a directory thither; optionally unlinking the target first; optionally overwriting; optionally verbalizing.

Usage

```
copy.dir(source, target, unlink.target=FALSE, overwrite=FALSE,
         verbose=FALSE)
```

Arguments

```
source       the source directory
target       the target directory
unlink.target delete target directory first?
overwrite    overwrite target files?
verbose      verbalize transaction?
```

Value

NULL

Note

Not tested on non-linux platforms

Curry	<i>Pre-specify a procedures named parameters, returning a new procedure.</i>
-------	--

Description

Pre-specify a procedures named parameters, returning a new procedure.

Usage

```
Curry(FUN, ...)
```

Arguments

FUN	the function to be curried
...	the determining parameters

Details

Thanks, Byron Ellis. <https://stat.ethz.ch/pipermail/r-devel/2007-November/047318.html>

Value

A new function partially determined

debug	<i>Convenience function to print variable-value pairs.</i>
-------	--

Description

Convenience function to print variable-value pairs.

Usage

```
debug(...)
```

Arguments

...	named variable of the form a=b, ...
-----	-------------------------------------

Value

NULL

```
description.dependencies
```

Gather a DESCRIPTION's dependencies from the...

Description

Gather a 'DESCRIPTION's dependencies from the Package, Depends, Imports, Suggests, and Enhances fields.

Usage

```
description.dependencies (description.file)
```

Arguments

```
description.file
    the 'DESCRIPTION' to parse
```

Value

A list of dependencies

TODO

Test this!

```
DESCRIPTION.FILE    Whither to copy collate...
```

Description

Whither to copy collate

```
DOC.DIR            Whither to install docs...
```

Description

Whither to install docs

`expression.from.partitum`

Extract the expression from the parse tree.

Description

Extract the expression from the parse tree.

Usage

`expression.from.partitum(partitum)`

Arguments

`partitum` `partitum` the parsed elements

Value

the extracted expression

`first.non.null`

Find the first non-null argument.

Description

Find the first non-null argument.

Usage

`first.non.null(...)`

Arguments

`...` the arguments

Value

The first non-null argument

`guess.name` *Pluck name from a hierarchy of candidates; viz.*

Description

Pluck name from a hierarchy of candidates; viz. name, assignee, S4class, S4method, S4generic.

Usage

```
guess.name(partitum)
```

Arguments

`partitum` the parsed elements

Value

The guessed name (possibly NULL)

`Identity` *Identity function.*

Description

Identity function.

Usage

```
Identity(...)
```

Arguments

`...` tautological arguments

Details

Is concatenation benign?

Value

The tautologized arguments, concatenated

include	<i>Collate value parser..</i>
---------	-------------------------------

Description

Collate value parser

See Also

make.collate.roclet

INST.DIR	<i>Whither to copy installables...</i>
----------	--

Description

Whither to copy installables

is.assignment	<i>Whether the expression implies assignment by <-...</i>
---------------	--

Description

Whether the expression implies assignment by <- or =.

Usage

```
is.assignment (expression)
```

Arguments

`expression` the expression to check for assignment

Value

Whether or not the expression assigns by <- =

`is.even`*Is a number even?*

Description

Is a number even?

Usage

```
is.even(a)
```

Arguments

`a` the number to test

Value

Whether the number is even

`is.function.definition`*Whether the expression assigns function...*

Description

Whether the expression assigns function

Usage

```
is.function.definition(expression)
```

Arguments

`expression` the expression to check for assignment

Value

Whether the expression assigns a function

`is.nil` *Whether a list is empty.*

Description

Whether a list is empty.

Usage

```
is.nil(list)
```

Arguments

`list` the list to test

Value

Whether the list is empty

`is.null.string` *Does the string contain no matter, but very well [:space:]?*

Description

Does the string contain no matter, but very well [:space:]?

Usage

```
is.null.string(string)
```

Arguments

`string` the string to check

Value

TRUE if the string contains words, otherwise FALSE

```
is.odd
```

Is a number odd?

Description

Is a number odd?

Usage

```
is.odd(a)
```

Arguments

a the number to test

Value

Whether the number is odd

```
LINE.DELIMITER
```

Sequence that distinguishes roxygen comment from normal comment.

Description

Sequence that distinguishes roxygen comment from normal comment.

```
make.callgraph.roclet
```

Make a callgraph roclet which produces a static call graph...

Description

Make a callgraph roclet which produces a static call graph from a given function at a given depth with or without primitives.

Usage

```
make.callgraph.roclet(dependencies, dir=".", verbose=TRUE)
```

Arguments

dependencies packages required to evaluate interesting functions
 dir the directory to place the callgraphs in
 verbose announce what we're doing

Details

The callgraph roclet supports the following tags:

1. `@callGraphCreate` a call graph of the default depth, excluding primitive functions.
2. `@callGraphPrimitivesCreate` a call graph of the default depth, including primitive functions.
3. `@callGraphDepthChange` the depth of the callgraph from the default of 2.

The callgraph roclet is awkward in the sense that it requires a function's package to be loadable; which means, like calling LaTeX multiple times, one has to run roxygen on a package, install it, run roxygen again to get the callgraphs, and possibly install the package again.

TODO

- `index.html` 'index.html' in 'inst/doc' for callgraphs, possibly with thumbnails in png
- Text-only option `Option` for text-only callgraphs (which are clearer, in my opinion)

```
make.collate.roclet
```

Make collate roclet which parses the given files; topologically...

Description

Make collate roclet which parses the given files; topologically sorting `@includes`, and either merging the `Collate:` directive with a pre-existing 'DESCRIPTION' or writing to standard out.

Usage

```
make.collate.roclet (merge.file, target.file="", verbose=TRUE)
```

Arguments

<code>merge.file</code>	'DESCRIPTION' file with which to merge directive; or NULL for none
<code>target.file</code>	whither to cat directive (whether merged or not); blank line is standard out
<code>verbose</code>	whether to describe what we're doing with the target.file

Details

Each `@include` tag should specify the filename of one intrapackage dependency; multiple `@include` tags may be given.

Contains the member function `parse` which parses an arbitrary number of files, and `parse.dir` which recursively parses a directory tree.

Value

Rd roclet

See Also

[make.roclet](#)

Examples

```
#' `example-a.R`, `example-b.R` and `example-c.R` reside
#' in the `example` directory, with dependencies
#' a -> {b, c}. This is `example-a.R`.
#' @include example-b.R
#' @include example-c.R
roxxygen()

roclet <- make.collate.roclet()
## Not run: roclet$parse.dir('example')
```

```
make.description.parser
```

Make a parser to parse DESCRIPTION files.

Description

Make a parser to parse ‘DESCRIPTION’ files.

Usage

```
make.description.parser(parse.default=cat.description,
  pre.parse=noop.description, post.parse=noop.description)
```

Arguments

<code>parse.default</code>	the default parser receiving a field and value
<code>pre.parse</code>	a function receiving the parsed fields before individual parsing
<code>post.parse</code>	a function receiving the parsed fields after individual parsing

Details

Contains the member functions `register.parser`, taking a field and parser; and `parse`, taking the parsed fields from [parse.description.file](#) or similar.

Value

NULL

```
make.namespace.roclet
```

Make a namespace roclet which parses the given files and writes a list of...

Description

Make a namespace roclet which parses the given files and writes a list of namespace directives to a given file or standard out; see *Writing R Extensions* (<http://cran.r-project.org/doc/manuals/R-exts.pdf>) for details.

Usage

```
make.namespace.roclet (outfile="", verbose=TRUE)
```

Arguments

outfile	whither to send output; blank string means standard out
verbose	whether to announce what we're doing with the <i>outfile</i>

Details

The namespace roclet supports the following tags:

Roxygen tag	'NAMESPACE' equivalent
@export	export
@exportClass	exportClasses
@exportMethod	exportMethod
@exportPattern	exportPattern
@S3method	S3method
@import	import
@importFrom	importFrom
@importClassesFrom	importClassesFrom
@importMethodsFrom	importMethodsFrom

1. @exportMay be specified with or without value; if unadorned, roxygen will try to guess the exported value by assignee, setMethod, setClass, etc. Otherwise, @export f g ... translates to export (f, g, ...).
2. @exportClassOverrides setClass.
3. @exportMethodOverrides setMethod or setGeneric.
4. @exportPatternSee “1.6.2 Registering S3 methods” from *Writing R Extensions*.
5. @S3methodOverrides the export of an S3 method.
6. @importSee “1.6.1 Specifying imports and exports” from *Writing R Extensions*.
7. @importFromSee “1.6.1 Specifying imports and exports” from *Writing R Extensions*.

8. @importClassesFrom See “1.6.6 Name spaces with formal classes and methods” from *Writing R Extensions*.
9. @importMethodsFrom See “1.6.6 Name spaces with formal classes and methods” from *Writing R Extensions*.

Value

Namespace roclet

Examples

```
#' An example file, example.R, which imports
#' packages foo and bar
#' @import foo bar
roxygen()

#' An exportable function
#' @export
fun <- function() {}

roclet <- make.namespace.roclet()
## Not run: roclet$parse('example.R')
```

make.Rd.roclet	<i>Make an Rd roclet which parses the given files and, if specified, populates...</i>
----------------	---

Description

Make an Rd roclet which parses the given files and, if specified, populates the given subdirectory with Rd files; or writes to standard out. See *Writing R Extensions* (<http://cran.r-project.org/doc/manuals/R-exts.pdf>) for details.

Usage

```
make.Rd.roclet(subdir, verbose=TRUE)
```

Arguments

subdir	directory into which to place the Rd files; if NULL, standard out.
verbose	whether to declare what we’re doing in the <i>subdir</i>

Details

The first paragraph of a roxygen block constitutes its description, the subsequent paragraphs its details; moreover, the Rd roclet supports these tags:

Roxygen tag	Rd analogue
@author	\author
@aliases	\alias, ...
@concept	\concept
@example	<i>n/a</i>
@examples	\examples
@format	\format
@keywords	\keyword, ...
@method	\method
@name	\name
@note	\note
@param	\arguments{\item, ...}
@references	\references
@return	\value
@seealso	\seealso
@source	\source
@title	\title
@TODO	<i>n/a</i>
@usage	\usage

1. @author See “2.1.1 Documenting functions” from *Writing R Extensions*.
2. @aliases A default alias is plucked from the @name or assignee; otherwise, @alias a b ... translates to \alias{a}, \alias{b}, &c. If you specify one alias, however, specify them all.
3. @concept See “2.8 Indices” from *Writing R Extensions*.
4. @example Each @example tag specifies an example file relative to the package head; if the file resides in ‘tests’, for instance, it will be checked with R CMD check. The contents of the file will be concatenated under \examples{...}.
5. @examples Verbatim examples; see “2.1.1 Documenting functions” from *Writing R Extensions*.
6. @format See “2.1.2 Documenting data sets” from *Writing R Extensions*.
7. @keywords @keywords a b ... translates to \keyword{a}, \keyword{b}, &c.
8. @method Use @method <generic> <class> to document S3 functions.
9. @name In the absense of an explicit @name tag, the name of an assignment is plucked from the assignee.
10. @note See “2.1.1 Documenting functions” from *Writing R Extensions*.
11. @param Each function variable should have a @param <variable> <description> specified.
12. @references See “2.1.1 Documenting functions” from *Writing R Extensions*.

13. `@return`The return value of the function, or `NULL`.
14. `@seealso`See “2.1.1 Documenting functions” from *Writing R Extensions*.
15. `@source`See “2.1.2 Documenting data sets” from *Writing R Extensions*.
16. `@title`A default title is plucked from the first sentence of the description; that is, the first phrase ending with a period, question mark or newline. In the absence of a description, the title becomes the `@name` or assignee; lastly, it can be overridden with `@title`.
17. `@TODO`Note to developers to get off their asses.
18. `@usage`A default usage is construed from a function’s formals, but can be overridden with `@usage` (e.g. in the case of multiple functions in one Rd unit).

Value

Rd roclet

TODO

param method setClass setGeneric setMethod make.Rd.roclet

Examples

```
#' This sentence describes the function.
#'
#' Here are the details (notice the preceding blank
#' line); the name, title, usage and alias will be
#' automatically generated.
#'
#' @param a a parameter
#' @return NULL
f <- function(a=1) NULL

#' S3 functions require a @method tag for
#' the time being.
#'
#' @method specialize foo
#' @param f a generic foo
#' @param ... ignored
#' @return The specialized foo
specialize.foo <- function(f, ...)
  actually.specialize(f)

roclet <- make.Rd.roclet('man')
## Not run: roclet$parse('example.R')
```

make.Rd2.roclet	<i>New implementation of the Rd roclet; same functionality as the original...</i>
-----------------	---

Description

New implementation of the Rd roclet; same functionality as the original implementation plus basic S4 handling.

Usage

```
make.Rd2.roclet(subdir, verbose=TRUE, exportonly=FALSE,  
               documentedonly=TRUE)
```

Arguments

subdir	directory into which to place the Rd files; if NULL, standard out.
verbose	whether to declare what we're doing in the <i>subdir</i>
exportonly	create Rd files only for exported "things"
documentedonly	create Rd files only for "things" which are documented with Roxygen

Details

See [make.Rd.roclet](#) for description and available tags; new tags are:

1. @nordSuppress Rd creation.
2. @rdnameDefinition of the Rd name; blocks with the same @rdname are merged into one Rd file.
3. @slotEach S4 class slot should have a @slot <name> <description> specified.

Value

Rd roclet

<code>make.roclet</code>	<i>Abstract roclet that serves as a rudimentary API.</i>
--------------------------	--

Description

Abstract roclet that serves as a rudimentary API.

Usage

```
make.roclet(parse.default, pre.parse, post.parse, pre.files,
            post.files)
```

Arguments

<code>parse.default</code>	the default parser taking key and value
<code>pre.parse</code>	a callback function taking a list of parsed elements; called before processing a file
<code>post.parse</code>	a callback function taking a list of parsed elements; called after processing a file
<code>pre.files</code>	a callback function with no arguments; called before any file has been parsed
<code>post.files</code>	a callback function with no arguments; called after every file has been parsed

Details

Contains the following member functions:

- `register.parsertakes` key and parser
- `register.parserstakes` parser and keys
- `register.default.parsertakes` a key
- `register.default.parserstake` parsers
- `parseparses` material contained in files

MAN.DIR	<i>Whither to copy Rds...</i>
---------	-------------------------------

Description

Whither to copy Rds

MATTER	<i>Anti-anti-words...</i>
--------	---------------------------

Description

Anti-anti-words

NAMESPACE.FILE	<i>Whither to copy namespace...</i>
----------------	-------------------------------------

Description

Whither to copy namespace

Negate	<i>Negate a function; borrowed from src/library/base/R/funprog...</i>
--------	---

Description

Negate a function; borrowed from src/library/base/R/funprog.R for pre-2.7 Rs.

Usage

Negate(f)

Arguments

f the function to be negated

Value

The negated function

nil	<i>The empty list...</i>
-----	--------------------------

Description

The empty list

`NIL.STRING` *Analogue to the empty list...*

Description

Analogue to the empty list

`noop.description` *Description parser that does nothing..*

Description

Description parser that does nothing

Usage

`noop.description(field, value)`

Arguments

<code>field</code>	the field to be parsed
<code>value</code>	the value to be parsed

Value

`NULL`

`nwords` *Number of words a string contains.*

Description

Number of words a string contains.

Usage

`nwords(string)`

Arguments

<code>string</code>	the string whose words to count
---------------------	---------------------------------

Value

Number of words in the string

`pairwise`*Combine a list into pairwise elements; lists should...*

Description

Combine a list into pairwise elements; lists should be of the same length. In case of odd numbers of members, the last will be removed.

Usage

```
pairwise(list)
```

Arguments

`list` the list to be pairwise decomposed

Value

A list of pairwise elements

`parse.assignee`*Find the assignee of the expression...*

Description

Find the assignee of the expression

Usage

```
parse.assignee(expression)
```

Arguments

`expression` the expression in which to find the assignee

Value

The expression's assignee

<code>parse.call</code>	<i>Parse a function call, paying special attention to...</i>
-------------------------	--

Description

Parse a function call, paying special attention to assignments by `<-` or `=`.

Usage

```
parse.call(expressions)
```

Arguments

`expressions` the expression to search through

Value

List of formals and assignee in case of assignment, the processed expression in case of non-assigning function calls (see `parse.srcref`).

<code>parse.default</code>	<i>Default parser which simply emits the key and expression;...</i>
----------------------------	---

Description

Default parser which simply emits the key and expression; used for elements with optional values (like `@export`) where roclets can do more sophisticated things with `NULL`.

Usage

```
parse.default(key, rest)
```

Arguments

`key` the parsing key
`rest` the expression to be parsed

Value

A list containing the key and expression (possibly null)

`parse.description` *Parse description: the premier part of a roxygen block...*

Description

Parse description: the premier part of a roxygen block containing description and option details separated by a blank roxygen line.

Usage

`parse.description(expression)`

Arguments

`expression` the description to be parsed

Value

A list containing the parsed description

`parse.description.file`
Convenience function to call...

Description

Convenience function to call `parse.description.text` with the given 'DESCRIPTION' file.

Usage

`parse.description.file(description.file)`

Arguments

`description.file`
the 'DESCRIPTION' file to be parsed

Value

NULL

```
parse.description.text
```

Parse lines of text corresponding to a package DESCRIPTION file.

Description

Parse lines of text corresponding to a package DESCRIPTION file.

Usage

```
parse.description.text (description)
```

Arguments

description the lines of text

Value

A list of values indexed by field

```
parse.element
```

Parse a raw string containing key and expressions.

Description

Parse a raw string containing key and expressions.

Usage

```
parse.element (element)
```

Arguments

element the string containing key and expressions

Value

A list containing the parsed constituents

parse.error	<i>Centrally formatted error; stopping execution...</i>
-------------	---

Description

Centrally formatted error; stopping execution

Usage

```
parse.error(key, message)
```

Arguments

key	the offending key
message	the apposite message

Value

NULL

parse.file	<i>Parse a source file containing roxygen directives.</i>
------------	---

Description

Parse a source file containing roxygen directives.

Usage

```
parse.file(file)
```

Arguments

file	string naming file to be parsed
------	---------------------------------

Value

List containing parsed directives

`parse.files` *Parse many files at one.*

Description

Parse many files at one.

Usage

```
parse.files(...)
```

Arguments

... files to be parsed

Value

List containing parsed directives

See Also

[parse.file](#)

`parse.formals` *Find the formal arguments associated with a given...*

Description

Find the formal arguments associated with a given expression (may be NULL).

Usage

```
parse.formals(expressions)
```

Arguments

`expressions` the expressions from which to extract formal arguments

Value

The formal arguments of said expression or NULL

parse.message *Centrally formatted message...*

Description

Centrally formatted message

Usage

```
parse.message(key, message)
```

Arguments

key	the offending key
message	the apposite message

Value

The formatted message

parse.name *Parse an element containing a single name and only a name;...*

Description

Parse an element containing a single name and only a name; extra material will be ignored and a warning issued.

Usage

```
parse.name(key, name)
```

Arguments

key	parsing key
name	the name to be parsed

Value

A list containing key and name

`parse.name.description`
Parse an element containing a mandatory name...

Description

Parse an element containing a mandatory name and description (such as @param).

Usage

```
parse.name.description(key, rest)
```

Arguments

<code>key</code>	the parsing key
<code>rest</code>	the expression to be parsed

Value

A list containing the key, name and description

`parse.preref` *Resorts to the default parser but with a warning about the...*

Description

Resorts to the default parser but with a warning about the unknown key.

Usage

```
parse.preref(key, rest)
```

Arguments

<code>key</code>	the parsing key
<code>rest</code>	the expression to be parsed

Value

A list containing the key and expression (possibly null)

See Also

[parse.default](#)

parse.ref	<i>Parse either srcrefs, prerefs or pairs of the same.</i>
-----------	--

Description

Parse either srcrefs, prerefs or pairs of the same.

Usage

```
parse.ref(ref, ...)
```

Arguments

ref	the srcref, preref or pair of the same
...	ignored

Value

List containing the parsed srcref/preref

parse.ref.list	<i>Parse a preref/srcrefs pair..</i>
----------------	--------------------------------------

Description

Parse a preref/srcrefs pair

Usage

```
## S3 method for class 'list':
parse.ref(ref, ...)
```

Arguments

ref	the preref/srcref pair
...	ignored

Value

List combining the parsed preref/srcref

`parse.ref.preref` *Parse a preref...*

Description

Parse a preref

Usage

```
## S3 method for class 'preref':  
parse.ref (ref, ...)
```

Arguments

<code>ref</code>	the preref to be parsed
<code>...</code>	ignored

Value

List containing the parsed preref

`parse.ref.srcref` *Parse a srcref...*

Description

Parse a srcref

Usage

```
### parse.ref.srcref(ref, ...)  
parse.ref.srcref(...)
```

Arguments

<code>...</code>	the srcref to be parsed
------------------	-------------------------

Value

List containing the parsed srcref

<code>parse.refs</code>	<i>Parse each of a list of preref/srcref pairs.</i>
-------------------------	---

Description

Parse each of a list of preref/srcref pairs.

Usage

```
parse.refs (preref.screfs)
```

Arguments

<code>preref.screfs</code>	list of preref/srcref pairs
----------------------------	-----------------------------

Value

List combining parsed preref/screfs

<code>parse.srcref</code>	<i>By default, screfs are ignored; this parser returns nil.</i>
---------------------------	---

Description

By default, screfs are ignored; this parser returns nil.

Usage

```
parse.srcref (pivot, expression)
```

Arguments

<code>pivot</code>	the parsing pivot
<code>expression</code>	the expression to be parsed

Value

nil

<code>parse.text</code>	<i>Text-parsing hack using tempfiles for more facility.</i>
-------------------------	---

Description

Text-parsing hack using tempfiles for more facility.

Usage

```
parse.text(...)
```

Arguments

... lines of text to be parsed

Value

The parse tree

<code>parse.toggle</code>	<i>Turn a binary element on; parameters are ignored.</i>
---------------------------	--

Description

Turn a binary element on; parameters are ignored.

Usage

```
parse.toggle(key, rest)
```

Arguments

key parsing key
rest the expression to be parsed

Value

A list with the key and TRUE

parse.value *Parse an element with a mandatory value.*

Description

Parse an element with a mandatory value.

Usage

parse.value(key, rest)

Arguments

key	the parsing key
rest	the expression to be parsed

Value

A list containing the key and value

parse.warning *Centrally formatted warning...*

Description

Centrally formatted warning

Usage

parse.warning(key, message)

Arguments

key	the offending key
message	the apposite message

Value

NULL

`parser.default` *Default parser-lookup; if key not found, return...*

Description

Default parser-lookup; if key not found, return the default parser specified.

Usage

```
parser.default(table, key, default)
```

Arguments

<code>table</code>	the parser table from which to look
<code>key</code>	the key upon which to look
<code>default</code>	the parser to return upon unsuccessful lookup

Value

The parser

`parser.preref` *Preref parser-lookup; defaults to parse...*

Description

Preref parser-lookup; defaults to `parse.preref`.

Arguments

<code>key</code>	the key upon which to look
------------------	----------------------------

Value

The parser

parser.srcref *Srcref parser-lookup; defaults to parse...*

Description

Srcref parser-lookup; defaults to parse.srcref.

Arguments

key the key upon which to look

Value

The parser

preorder.flatten.expression
 Flatten a nested expression into a list, preorderly.

Description

Flatten a nested expression into a list, preorderly.

Usage

preorder.flatten.expression(expression)

Arguments

expression the root of the expression to be flattened

Value

A list containing the flattened expression

```
preorder.walk.expression
```

Recursively walk an expression (as returned by parse) in...

Description

Recursively walk an expression (as returned by `parse`) in preorder.

Usage

```
preorder.walk.expression(proc, expression)
```

Arguments

<code>proc</code>	the procedure to apply to each subexpression
<code>expression</code>	the root of the expression

Value

NULL

```
preref.parsers
```

Preref parser table...

Description

Preref parser table

TODO

number parser?

prerefs	<i>Comment blocks (possibly null) that precede a file's expressions.</i>
---------	--

Description

Comment blocks (possibly null) that precede a file's expressions.

Usage

```
prerefs(srcfile, srcrefs)
```

Arguments

srcfile	result of running srcfile on an interesting file
srcrefs	the resultant srcrefs

Value

A list of prerefs that resemble srcrefs in form, i.e. with srcfile and lloc

R.DIR	<i>Whence to copy source code...</i>
-------	--------------------------------------

Description

Whence to copy source code

Reduce.paste	<i>Ad-hoc abstraction to paste processed list-elements together.</i>
--------------	--

Description

Ad-hoc abstraction to paste processed list-elements together.

Usage

```
Reduce.paste(proc, elts, sep)
```

Arguments

proc	the procedure to apply to the elements
elts	the elements to be processed
sep	the glue to joined the processed elements

Value

The processed elements as a glued string

`register.parser` *Register a parser with a table...*

Description

Register a parser with a table

Usage

```
register.parser(table, key, parser)
```

Arguments

<code>table</code>	the table under which to register
<code>key</code>	the key upon which to register
<code>parser</code>	the parser callback to register; a function taking <code>key</code> and expression

Value

NULL

`register.parsers` *Register many parsers at once.*

Description

Register many parsers at once.

Usage

```
register.parsers(table, parser, ...)
```

Arguments

<code>table</code>	the table under which to register
<code>parser</code>	the parser to register
<code>...</code>	the keys upon which to register

Value

NULL

`register.preref.parser`*Specifically register a preref parser...*

Description

Specifically register a preref parser

Arguments

<code>key</code>	the key upon which to register
<code>parser</code>	the parser callback to register; a function taking <code>key</code> and expression

Value

NULL

See Also

[register.parser](#)

`register.preref.parsers`*Register many preref parsers at once.*

Description

Register many preref parsers at once.

Arguments

<code>parser</code>	the parser to register
<code>...</code>	the keys upon which to register

Value

NULL

```
register.srcref.parser
```

Specifically register a srcref parser..

Description

Specifically register a srcref parser

Arguments

key	the key upon which to register
parser	the parser callback to register; a function taking key and expression

Value

NULL

See Also

[register.parser](#)

```
register.srcref.parsers
```

Register many srcref parsers at once.

Description

Register many srcref parsers at once.

Arguments

parser	the parser to register
...	the keys upon which to register

Value

NULL

roxygen	<i>No-op for sourceless files...</i>
---------	--------------------------------------

Description

No-op for sourceless files

Value

NULL

ROXYGEN.DIR	<i>Whither to copy package...</i>
-------------	-----------------------------------

Description

Whither to copy package

roxygenize	<i>Process a package with the Rd, namespace and collate roclets.</i>
------------	--

Description

Process a package with the Rd, namespace and collate roclets.

Usage

```
roxygenize(package.dir, roxygen.dir, copy.package=TRUE, overwrite=TRUE,
           unlink.target=FALSE, use.Rd2=FALSE)
```

Arguments

package.dir	the package's top directory
roxygen.dir	whither to copy roxygen files; defaults to 'package.roxygen'.
copy.package	copies the package over before adding/manipulating files.
overwrite	overwrite target files
unlink.target	unlink target directory before processing files
use.Rd2	use the Rd2 roclet

Value

NULL

TODO

Options to enable/disable specific roclet (`--no-callgraphs`, etc.)

<code>SPACE</code>	<i>Absence of words...</i>
--------------------	----------------------------

Description

Absence of words

<code>src.lines</code>	<i>Extract the source code from parsed elements...</i>
------------------------	--

Description

Extract the source code from parsed elements

Usage

`src.lines` (partitum)

Arguments

`partitum` the parsed elements

Value

The lines of source code

<code>srcref.parsers</code>	<i>Srcref parser table...</i>
-----------------------------	-------------------------------

Description

Srcref parser table

strcar	<i>First word in a string.</i>
--------	--------------------------------

Description

First word in a string.

Usage

```
strcar(string)
```

Arguments

string the string whose word to find

Value

The first word

strcdr	<i>Words after first in a string.</i>
--------	---------------------------------------

Description

Words after first in a string.

Usage

```
strcdr(string)
```

Arguments

string the string whose words to find

Value

The words after first in the string

`strcons` *Join two string.*

Description

Join two string.

Usage

```
strcons(consor, consee, sep)
```

Arguments

<code>consor</code>	the joining string
<code>consee</code>	the joined string
<code>sep</code>	the intervening space

Value

The joined strings

`strmap` *Map through the words in a string, joining the mapped...*

Description

Map through the words in a string, joining the mapped words with a separator.

Usage

```
strmap(proc, sep, string)
```

Arguments

<code>proc</code>	procedure to apply to each word
<code>sep</code>	the separator joining the mapped words
<code>string</code>	the string to be mapped

Details

General enough to be designated 'map': isn't it closer to a specialized reduce?

Value

Mapped words separated by `sep`

substr.regexpr	<i>Actually do the substring representation that...</i>
----------------	---

Description

Actually do the substring representation that regexpr should do; does not acknowledge groups, since regexpr doesn't.

Usage

```
substr.regexpr(pattern, text)
```

Arguments

pattern	the pattern to match
text	the text to match against

Value

The matched substring

TAG.DELIMITER	<i>Symbol that delimits tags.</i>
---------------	-----------------------------------

Description

Symbol that delimits tags.

trim	<i>Trim [:space:] on both sides of a string.</i>
------	--

Description

Trim [:space:] on both sides of a string.

Usage

```
trim(string)
```

Arguments

string	the string to be trimmed
--------	--------------------------

Value

A trimmed string

<code>trim.left</code>	<i>Trim [:space:] to the left of a string.</i>
------------------------	--

Description

Trim [:space:] to the left of a string.

Usage

```
trim.left(string)
```

Arguments

<code>string</code>	the string to be trimmed
---------------------	--------------------------

Value

A left-trimmed string

<code>trim.right</code>	<i>Trim [:space:] to the right of a string.</i>
-------------------------	---

Description

Trim [:space:] to the right of a string.

Usage

```
trim.right(string)
```

Arguments

<code>string</code>	the string to be trimmed
---------------------	--------------------------

Value

A right-trimmed string

`word.ref`*Find the nth word in a string.*

Description

Find the nth word in a string.

Usage

```
word.ref(string, n)
```

Arguments

<code>string</code>	the string to search in
<code>n</code>	the nth word to find

Value

A list containing:

<code>start</code>	the first letter of the word.
<code>end</code>	the last letter of the word.

Undefined if no such word; though `end` may be less than `start` in such a case.

`zip`*Zip n lists together into tuples of...*

Description

Zip *n* lists together into tuples of length *n*.

Usage

```
zip(zipper, ...)
```

Arguments

<code>zipper</code>	the zipping function
<code>...</code>	the lists to be zipped

Value

A list of tuples

`zip.c`*Zip using c.*

Description

Zip using `c`.

Usage

```
zip.c(...)
```

Arguments

... the lists to be zipped

Value

A list of tuples

See Also

[zip](#)

`zip.list`*Zip using list.*

Description

Zip using `list`.

Usage

```
zip.list(...)
```

Arguments

... the lists to be zipped

Value

A list of tuples

See Also

[zip](#)

Index

*Topic package

roxygen-package, 2

aliases (*make.Rd.roclet*), 21

assign.parent, 3

author (*make.Rd.roclet*), 21

caar, 4

cadar, 4

caddr, 5

cadr, 5

callGraph
(*make.callgraph.roclet*), 17

callGraphDepth
(*make.callgraph.roclet*), 17

callGraphPrimitives
(*make.callgraph.roclet*), 17

car, 6

cat.description, 6

cdddr, 7

cddr, 7

cdr, 8

Compose, 8

concept (*make.Rd.roclet*), 21

copy.dir, 9

Curry, 9

debug, 10

description.dependencies, 10

DESCRIPTION.FILE, 11

DOC.DIR, 11

example (*make.Rd.roclet*), 21

examples (*make.Rd.roclet*), 21

export (*make.namespace.roclet*), 19

exportClass
(*make.namespace.roclet*), 19

exportMethod
(*make.namespace.roclet*), 19

exportPattern
(*make.namespace.roclet*), 19

expression.from.partitum, 11

first.non.null, 12

guess.name, 12

Identity, 13

import (*make.namespace.roclet*), 19

importClassesFrom
(*make.namespace.roclet*), 19

importFrom
(*make.namespace.roclet*), 19

importMethodsFrom
(*make.namespace.roclet*), 19

include, 13

INST.DIR, 13

is.assignment, 14

is.even, 14

is.function.definition, 15

is.nil, 15

is.null.string, 16

is.odd, 16

keywords (*make.Rd.roclet*), 21

LINE.DELIMITER, 16

make.callgraph.roclet, 3, 17

make.collate.roclet, 3, 18

make.description.parser, 19

make.namespace.roclet, 3, 19

make.Rd.roclet, 3, 21, 23

make.Rd2.roclet, 23

make.roclet, 18, 24

MAN.DIR, 24

MATTER, 25

name (*make.Rd.roclet*), 21

NAMESPACE.FILE, 25

Negate, 25

nil, 25

- NIL.STRING, 26
- noop.description, 26
- nord(*make.Rd2.roclet*), 23
- note(*make.Rd.roclet*), 21
- nwords, 26
- pairwise, 27
- parse.assignee, 27
- parse.call, 28
- parse.default, 28, 34
- parse.description, 29
- parse.description.file, 19, 29
- parse.description.text, 29, 30
- parse.element, 30
- parse.error, 31
- parse.file, 31, 32
- parse.files, 32
- parse.formals, 32
- parse.message, 33
- parse.name, 33
- parse.name.description, 34
- parse.preref, 34
- parse.ref, 35
- parse.ref.list, 35
- parse.ref.preref, 36
- parse.ref.srcref, 36
- parse.refs, 37
- parse.srcref, 37
- parse.text, 38
- parse.toggle, 38
- parse.value, 39
- parse.warning, 39
- parser.default, 40
- parser.preref, 40
- parser.srcref, 41
- preorder.flatten.expression, 41
- preorder.walk.expression, 42
- preref.parsers, 42
- prerefs, 43
- R.DIR, 43
- rdname(*make.Rd2.roclet*), 23
- Reduce.paste, 43
- references(*make.Rd.roclet*), 21
- register.parser, 44, 45, 46
- register.parsers, 44
- register.preref.parser, 45
- register.preref.parsers, 45
- register.srcref.parser, 46
- register.srcref.parsers, 46
- return(*make.Rd.roclet*), 21
- roxygen, 47
- roxygen-package, 2
- ROXYGEN.DIR, 47
- roxygenize, 3, 47
- S3method(*make.namespace.roclet*), 19
- seealso(*make.Rd.roclet*), 21
- slot(*make.Rd2.roclet*), 23
- SPACE, 48
- src.lines, 48
- srcref.parsers, 48
- strcar, 49
- strcdr, 49
- strcons, 50
- strmap, 50
- substr.regexpr, 51
- TAG.DELIMITER, 51
- title(*make.Rd.roclet*), 21
- trim, 51
- trim.left, 52
- trim.right, 52
- usage(*make.Rd.roclet*), 21
- word.ref, 53
- zip, 53, 54
- zip.c, 54
- zip.list, 54