Package 'roxygen'

August 28, 2008

Version 0.1

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>

Maintainer Peter Danenberg <pcd@roxygen.org>

URL http://roxygen.org

Suggests Rgraphviz (>= 1.19.2)

Collate 'functional.R' 'list.R' 'roxygen.R' 'string.R' 'parse.R' 'roclet.R' 'callgraph.R' 'description.R' 'collate.R' 'namespace.R' 'Rd.R' 'roxygenize.R'

R topics documented:

assign.parent . . . 3
caar . . . 4
cadar . . . 4
caddr . . . 5
cadr . . . 5
car . . . 6
cat.description . . . 6
cdddr . . . 7
cddr . . . 7
cdr . . . 8
Compose . . . 8
copy.dir . . . 9
Curry . . . 9
debug . . . 10
description.dependencies . . . 10
DESCRIPTION.FILE . . . 11
DOC.DIR . . . 11
first.non.null . . . 11
guess.name . . . 11

Identity . . . 12
include . . . 12
INST.DIR . . . 12
is.assignment . . . 13
is.even . . . 13
is.function.definition . . . 14
is.nil . . . 14
is.null.string . . . 15
is.odd . . . 15
LINE.DELIMITER . . . 15
make.callgraph.roclet . . . 16
make.collate.roclet . . . 16
make.description.parser . . . 17
make.namespace.roclet . . . 18
make.Rd.roclet . . . 19
make.roclet . . . 21
MAN.DIR . . . 22
MATTER . . . 22
NAMESPACE.FILE . . . 22
Negate . . . 22
nil . . . 22
NIL.STRING . . . 23
noop.description . . . 23
nwords . . . 23
pairwise . . . 24
parse.assignee . . . 24
parse.call . . . 25
parse.default . . . 25
parse.description.file . . . 26
parse.description . . . 26
parse.description.text . . . 27
parse.element . . . 27
parse.error . . . 28
parse.file . . . 28
parse.files . . . 29
parse.formals . . . 29
parse.message . . . 30
parse.name.description . . . 30
parse.name . . . 31
parse.preref . . . 31
parser.default . . . 32
parse.ref.list . . . 32
parse.ref.preref . . . 33
parse.ref . . . 33
parse.ref.srcref . . . 34
parse.refs . . . 34
parser.preref . . . 35
parser.srcref . . . 35
parse.srcref . . . 35
parse.text . . . 36
parse.toggle . . . 36
parse.value . . . 37

parse.warning . . . 37
preorder.flatten.expression . . . 38
preorder.walk.expression . . . 38
preref.parsers . . . 38
prerefs . . . 39
R.DIR . . . 39
Reduce.paste . . . 39
register.parser . . . 40
register.parsers . . . 40
register.preref.parser . . . 41
register.preref.parsers . . . 41
register.srcref.parser . . . 42
register.srcref.parsers . . . 42
ROXYGEN.DIR . . . 42
roxygenize . . . 43
roxygen-package . . . 43
roxygen . . . 44
SPACE . . . 44
src.lines . . . 45
srcref.parsers . . . 45
strcar . . . 45
strcdr . . . 46
strcons . . . 46
strmap . . . 47
substr.regexpr . . . 47
TAG.DELIMITER . . . 48
trim.left . . . 48
trim . . . 48
trim.right . . . 49
word.ref . . . 49
zip.c . . . 50
zip.list . . . 50
zip . . . 51

Index **52**

`assign.parent` *Assign a variable in the parent environment when «-...*

Description

Assign a variable in the parent environment when <<− doesn't see 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` *Composite car/cdr...*

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...*

Description

Composite `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

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

Value

NULL

cdddr *Composite car/cdr...*

Description

Composite `car`/`cdr`

Usage

```
cdddr(list)
```

Arguments

`list`	the list from which to extract

Value

The extracted elements

cddr *Composite car/cdr...*

Description

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

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 *Convenience function to print variable-value pairs.*

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

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 — *Collate value parser...*

Description

Collate value parser

See Also

make.collate.roclet

INST.DIR — *Whither to copy installables...*

Description

Whither to copy installables

is.assignment *Whether the expression implies assignment by <-...*

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`	anounce what we're doing

Details

The callgraph roclet supports the following tags:

`@callGraph`	Create a call graph of the default depth, excluding primitive functions.
`@callGraphPrimitives`	Create a call graph of the default depth, including primitive functions.
`@callGraphDepth`	Change 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

merge.file	'DESCRIPTION' file with which to merge directive; or NULL for none
target.file	whither to cat directive (whether merged or not); blank line is standard out
verbose	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
roxygen()

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

parse.default	the default parser receiving a field and value
pre.parse	a function receiving the parsed fields before individual parsing
post.parse	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 anounce what we're doing with the *outfile*

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`

`@export` May 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, ...)`.

`@exportClass` Overrides `setClass`.

`@exportMethod` Overrides `setMethod` or `setGeneric`.

`@exportPattern` See "1.6.2 Registering S3 methods" from *Writing R Extensions*.

`@S3method` Overrides the export of an S3 method.

`@import`	See "1.6.1 Specifying imports and exports" from *Writing R Extensions*.
`@importFrom`	See "1.6.1 Specifying imports and exports" from *Writing R Extensions*.
`@importClassesFrom`	See "1.6.6 Name spaces with formal classes and methods" from *Writing R Extensions*.
`@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` *Make an Rd roclet which parses the given files and, if specified, populates...*

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 *subdir*

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`	*n/a*

`@examples`	`\examples`
`@keywords`	`\keyword, ...`
`@method`	`\method`
`@name`	`\name`
`@note`	`\note`
`@param`	`\arguments{\item, ...}`
`@references`	`\references`
`@return`	`\value`
`@seealso`	`\seealso`
`@title`	`\title`
`@TODO`	*n/a*
`@usage`	`\usage`

`@author` See "2.1.1 Documenting functions" from *Writing R Extensions*.

`@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.

`@concept` See "2.8 Indices" from *Writing R Extensions*.

`@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{...}`.

`@examples` Verbatim examples; see "2.1.1 Documenting functions" from *Writing R Extensions*.

`@keywords` `@keywords a b ...` translates to `\keyword{a}`, `\keyword{b}`, &c.

`@method` Use `@method <generic> <class>` to document S3 functions.

`@name` In the absense of an explicit `@name` tag, the name of an assignment is plucked from the assignee.

`@note` See "2.1.1 Documenting functions" from *Writing R Extensions*.

`@param` Each function variable should have a `@param <variable> <description>` specified.

`@references` See "2.1.1 Documenting functions" from *Writing R Extensions*.

`@return` The return value of the function, or `NULL`.

`@seealso` See "2.1.1 Documenting functions" from *Writing R Extensions*.

`@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`.

`@TODO` Note to developers to get off their asses.

`@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

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.roclet` *Abstract roclet that serves as a rudimentary API.*

Description

Abstract roclet that serves as a rudimentary API.

Usage

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

Arguments

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

Details

Contains the following member functions:

register.parser	takes `key` and `parser`
register.parsers	takes `parser` and `keys`
register.default.parser	takes a `key`
register.default.parsers	take `parsers`
parse	parses material contained in files

MAN.DIR *Whither to copy Rds...*

Description

Whither to copy Rds

MATTER *Anti-anti-words...*

Description

Anti-anti-words

NAMESPACE.FILE *Whither to copy namespace...*

Description

Whither to copy namespace

Negate *Negate a function; borrowed from src/library/base/R/funprog...*

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 *The empty list...*

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

field	the field to be parsed
value	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

string	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

parse.call *Parse a function call, paying special attention to...*

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).

parse.default *Default parser which simply emits the key and expression;...*

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.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 *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.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 tex

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 — *Centrally formatted error; stopping execution...*

Description

Centrally formatted error; stopping execution

Usage

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

Arguments

key	the offending key
message	the apposite message

Value

NULL

parse.file — *Parse a source file containing roxygen directives.*

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.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

key	the parsing key
rest	the expression to be parsed

Value

A list containing the key, name and description

`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.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

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

Value

A list containing the key and expression (possibly null)

See Also

`parse.default`

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

table	the parser table from which to look
key	the key upon which to look
default	the parser to return upon unsuccessful lookup

Value

The parser

parse.ref.list *Parse a preref/srcrefs pair...*

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

ref	the preref to be parsed
...	ignored

Value

List containing the parsed preref

parse.ref *Parse either srcrefs, prerefs or pairs of the same.*

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.srcref *Parse a srcref...*

Description

Parse a srcref

Usage

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

Arguments

ref	the srcref to be parsed
...	ignored

Value

List containing the parsed srcref

parse.refs *Parse each of a list of preref/srcref pairs.*

Description

Parse each of a list of preref/srcref pairs.

Usage

```
parse.refs(preref.srcrefs)
```

Arguments

preref.srcrefs	list of preref/srcref pairs

Value

List combining parsed preref/srcrefs

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

Description

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

Arguments

`key`	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

`parse.srcref` *By default, srcrefs are ignored; this parser returns nil.*

Description

By default, srcrefs are ignored; this parser returns `nil`.

Usage

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

Arguments

`pivot`	the parsing pivot
`expression`	the expression to be parsed

Value

`nil`

parse.text *Text-parsing hack using tempfiles for more facility.*

Description

Text-parsing hack using tempfiles for more facility.

Usage

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

Arguments

...	lines of text to be parsed

Value

The parse tree

parse.toggle *Turn a binary element on; parameters are ignored.*

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

`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

`proc`	the procedure to apply to each subexpression
`expression`	the root of the expression

Value

NULL

`preref.parsers` *Preref parser table...*

Description

Preref parser table

TODO

number parser?

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

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 — *Whence to copy source code...*

Description

Whence to copy source code

Reduce.paste — *Ad-hoc abstraction to paste processed list-elements together.*

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

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

Value

NULL

register.parsers *Register many parsers at once.*

Description

Register many parsers at once.

Usage

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

Arguments

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

Value

NULL

register.preref.parser

Specifically register a preref parser...

Description

Specifically register a preref 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.preref.parsers

Register many preref parsers at once.

Description

Register many preref parsers at once.

Arguments

parser	the parser to register
...	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.DIR` *Whither to copy package...*

Description

Whither to copy package

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

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)
```

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

Value

NULL

TODO

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

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
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')
```

---

roxygen *No-op for sourceless files...*

Description

No-op for sourceless files

Value

`NULL`

SPACE *Absence of words...*

Description

Absence of words

`src.lines` *Extract the source code from parsed elements...*

Description

Extract the source code from parsed elements

Usage

```
src.lines(partitum)
```

Arguments

`partitum`	the parsed elements

Value

The lines of source code

`srcref.parsers` *Srcref parser table...*

Description

Srcref parser table

`strcar` *First word in a string.*

Description

First word in a string.

Usage

```
strcar(string)
```

Arguments

`string`	the string whose word to finde

Value

The first word

strcdr *Words after first in a string.*

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

consor	the joining string
consee	the joined string
sep	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

proc	procedure to apply to each word
sep	the separator joining the mapped words
string	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 *Actually do the substring representation that...*

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 *Symbol that delimits tags.*

Description

Symbol that delimits tags.

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

Description

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

Usage

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

Arguments

string	the string to be trimmed

Value

A left-trimmed string

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

Description

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

Usage

```
trim(string)
```

Arguments

string	the string to be trimmed

Value

A trimmed string

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

Description

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

Usage

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

Arguments

`string`	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

`string`	the string to search in
`n`	the nth word to find

Value

A list containing:

`start`	the first letter of the word.
`end`	the last letter of the word.

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

zip.c *Zip using c.*

Description

Zip using `c`.

Usage

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

Arguments

`...`	the lists to be zipped

Value

A list of tuplets

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 tuplets

See Also

`zip`

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

Description

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

Usage

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

Arguments

`zipper`	the zipping function
`...`	the lists to be zipped

Value

A list of tuplets

Index

∗Topic **package**
roxygen-package, 41

aliases *(make.Rd.roclet)*, 17
assign.parent, 1
author *(make.Rd.roclet)*, 17

caar, 2
cadar, 2
caddr, 3
cadr, 3
callGraph
(make.callgraph.roclet), 14
callGraphDepth
(make.callgraph.roclet), 14
callGraphPrimitives
(make.callgraph.roclet), 14
car, 4
cat.description, 4
cdddr, 5
cddr, 5
cdr, 6
Compose, 6
concept *(make.Rd.roclet)*, 17
copy.dir, 7
Curry, 7

debug, 8
description.dependencies, 8
DESCRIPTION.FILE, 9
DOC.DIR, 9

example *(make.Rd.roclet)*, 17
examples *(make.Rd.roclet)*, 17
export *(make.namespace.roclet)*, 16
exportClass
(make.namespace.roclet), 16
exportMethod
(make.namespace.roclet), 16
exportPattern
(make.namespace.roclet), 16

first.non.null, 9

guess.name, 9

Identity, 10
import *(make.namespace.roclet)*, 16
importClassesFrom
(make.namespace.roclet), 16
importFrom
(make.namespace.roclet), 16
importMethodsFrom
(make.namespace.roclet), 16
include, 10
INST.DIR, 10
is.assignment, 11
is.even, 11
is.function.definition, 12
is.nil, 12
is.null.string, 13
is.odd, 13

keywords *(make.Rd.roclet)*, 17

LINE.DELIMITER, 13

make.callgraph.roclet, 14, *42*
make.collate.roclet, 14, *42*
make.description.parser, 15
make.namespace.roclet, 16, *42*
make.Rd.roclet, 17, *42*
make.roclet, *15*, 19
MAN.DIR, 20
MATTER, 20
method *(make.Rd.roclet)*, 17

name *(make.Rd.roclet)*, 17
NAMESPACE.FILE, 20
Negate, 20
nil, 20
NIL.STRING, 21
noop.description, 21
note *(make.Rd.roclet)*, 17
nwords, 21

pairwise, 22
param *(make.Rd.roclet)*, 17
parse.assignee, 22
parse.call, 23
parse.default, 23, *29*

parse.description, 24
parse.description.file, *16*, 24
parse.description.text, *24*, 25
parse.element, 25
parse.error, 26
parse.file, 26, *27*
parse.files, 27
parse.formals, 27
parse.message, 28
parse.name, 29
parse.name.description, 28
parse.preref, 29
parse.ref, 31
parse.ref.list, 30
parse.ref.preref, 31
parse.ref.srcref, 32
parse.refs, 32
parse.srcref, 33
parse.text, 34
parse.toggle, 34
parse.value, 35
parse.warning, 35
parser.default, 30
parser.preref, 33
parser.srcref, 33
preorder.flatten.expression, 36
preorder.walk.expression, 36
preref.parsers, 36
prerefs, 37

R.DIR, 37
Reduce.paste, 37
references *(make.Rd.roclet)*, 17
register.parser, 38, *39, 40*
register.parsers, 38
register.preref.parser, 39
register.preref.parsers, 39
register.srcref.parser, 40
register.srcref.parsers, 40
return *(make.Rd.roclet)*, 17
roxygen, 42
roxygen-package, 41
ROXYGEN.DIR, 40
roxygenize, 41, *42*

S3method *(make.namespace.roclet)*, 16
seealso *(make.Rd.roclet)*, 17
setClass *(make.Rd.roclet)*, 17
setGeneric *(make.Rd.roclet)*, 17
setMethod *(make.Rd.roclet)*, 17
SPACE, 42
src.lines, 43
srcref.parsers, 43
strcar, 43
strcdr, 44
strcons, 44
strmap, 45
substr.regexpr, 45

TAG.DELIMITER, 46
title *(make.Rd.roclet)*, 17
TODO *(make.Rd.roclet)*, 17
trim, 46
trim.left, 46
trim.right, 47

usage *(make.Rd.roclet)*, 17

word.ref, 47

zip, *48*, 49
zip.c, 48
zip.list, 48