# Package: inlinedocs (via r-universe)

October 11, 2024

**Title** Convert Inline Comments to Documentation

**Type** Package

**Version** 2023.9.4

**Description** Generates Rd files from R source code with comments. The main features of the default syntax are that (1) docs are defined in comments near the relevant code, (2) function argument names are not repeated in comments, and (3) examples are defined in R code, not comments. It is also easy to define a new syntax.

**URL** https://github.com/tdhock/inlinedocs

**BugReports** https://github.com/tdhock/inlinedocs/issues

**Depends** methods, utils, R (>= 2.9)

**License** GPL-2 | GPL-3

**LazyLoad** yes

**Encoding** UTF-8

**Suggests** future.apply, future, R.methodsS3

**Repository** https://tdhock.r-universe.dev

**RemoteUrl** https://github.com/tdhock/inlinedocs

**RemoteRef** HEAD

**RemoteSha** ef5fa9a4b0c99419170d6f06b4ffa80670ba2373

## Contents

## apply.parsers *apply parsers*

### Description

Parse code to r objs, then run all the parsers and return the documentation list.

### Usage

```
apply.parsers(code, parsers = default.parsers,
    verbose = FALSE,
    ...)
```

### Arguments

| | |
|---|---|
| code | Character vector of code lines. |
| parsers | List of Parser Functions. |
| verbose | Echo names of Parser Functions? |
| ... | Additional arguments to pass to Parser Functions. |

### Value

A list of extracted documentation from code.

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## classic.parsers *classic parsers*

### Description

List of classic parsers which were default before 2018.

### Usage

```
"classic.parsers"
```

## combine *combine*

### Description

combine lists or character strings

### Usage

```
combine(x, y)
```

### Arguments

| | |
|---|---|
| x | x |
| y | y |

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## combine.character *combine character*

### Description

combine character strings by pasting them together

### Usage

```
## S3 method for class 'character'
combine(x,
    y)
```

### Arguments

| | |
|---|---|
| x | x |
| y | y |

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## combine.list *combine list*

### Description

combine lists by adding elements or adding to existing elements

### Usage

```
## S3 method for class 'list'
combine(x, y)
```

### Arguments

| | |
|---|---|
| x | x |
| y | y |

### Value

A list, same type as x, but with added elements from y.

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## combine.NULL *combine NULL*

### Description

combine NULL objects.

### Usage

```
## S3 method for class 'NULL'
combine(x, y)
```

### Arguments

| | |
|---|---|
| x | x |
| y | y |

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## decomment *decomment*

### Description

Remove comment prefix and join lines of code to form a documentation string.

### Usage

```
decomment(comments)
```

### Arguments

| | |
|---|---|
| comments | Character vector of prefixed comment lines. |

### Value

String without prefixes or newlines.

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## default.parsers *default parsers*

### Description

List of parsers to use by default with package.skeleton.dx.

### Usage

```
"default.parsers"
```

## descfile.names *descfile names*

### Description

Names of Parser Functions that operate on the desc arg.

### Usage

```
"descfile.names"
```

`do.not.generate` *do not generate*

## Description

Make a Parser Function used to indicate that certain Rd files should not be generated.

## Usage

```
do.not.generate(...)
```

## Arguments

| | |
|---|---|
| `...` | Character strings indicating Rd files without the .Rd suffix. |

## Value

A Parser Function that will delete items from the outer Documentation List.

## Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## Examples

```
silly.pkg <- system.file("silly",package="inlinedocs")
owd <- setwd(tempdir())
file.copy(silly.pkg,".",recursive=TRUE)

## define a custom Parser Function that will not generate some Rd
## files
custom <- do.not.generate("SillyTest-class")
parsers <- c(default.parsers,list(exclude=custom))

## At first, no Rd files in the man subdirectory.
man.dir <- file.path("silly","man")
dir(man.dir)

## Running package.skeleton.dx will generate bare-bones files for
## those specified in do.not.generate, if they do not exist.
package.skeleton.dx("silly",parsers)
Rd.files <- c("SillyTest-class.Rd","silly.example.Rd")
Rd.paths <- file.path(man.dir,Rd.files)
stopifnot(all(file.exists(Rd.paths)))

## Save the modification times of the Rd files
old <- file.info(Rd.paths)$mtime
```

```
## make sure there is at least 2 seconds elapsed, which is the
## resolution for recording times on windows file systems.
Sys.sleep(4)

## However, it will NOT generate Rd for files specified in
## do.not.generate, if they DO exist already.
package.skeleton.dx("silly",parsers)
mtimes <- data.frame(old,new=file.info(Rd.paths)$mtime)
rownames(mtimes) <- Rd.files
mtimes$changed <- mtimes$old != mtimes$new
print(mtimes)
stopifnot(mtimes["SillyTest-class.Rd","changed"]==FALSE)
stopifnot(mtimes["silly.example.Rd","changed"]==TRUE)

unlink("silly",recursive=TRUE)
setwd(owd)
```

---

DocLink-class *Link documentation among related functions*

---

## Description

The .DocLink class provides the basis for hooking together documentation of related classes/functions/objects. The aim is that documentation sections missing from the child are inherited from the parent class.

## Objects from the Class

Objects can be created by calls of the form new(DocLink ...)

## Slots

**name:** (character) name of object

**created:** (character) how created

**parent:** (character) parent class or NA

**code:** (character) actual source lines

**description:** (character) preceding description block

## Methods

No methods defined with class "DocLink" in the signature.

## escape_dots *escape dots*

### Description

Convert ... to \dots

### Usage

```
escape_dots(arg)
```

### Arguments

| | |
|---|---|
| arg | arg |

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## extra.code.docs *Extract documentation from code chunks*

### Description

Parse R code to extract inline documentation from comments around each function. These are not able to be retreived simply by looking at the "source" attribute. This is a Parser Function that can be used in the parser list of package.skeleton.dx(). TODO: Modularize this into separate Parsers Functions for S4 classes, prefixes, ##«blocks, etc. Right now it is not very clean!

### Usage

```
extra.code.docs(code,
    objs, ...)
```

### Arguments

| | |
|---|---|
| code | Code lines in a character vector containing multiple R objects to parse for documentation. |
| objs | The objects defined in the code. |
| ... | ignored |

### Value

named list of lists, one for each object to document.

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

---

## extract.docs.file *extract docs file*

---

### Description

Apply all `parsers` relevant to extract info from just 1 code file.

### Usage

```
extract.docs.file(f,
    parsers = NULL, ...)
```

### Arguments

| | |
|---|---|
| `f` | File name of R code to read and parse. |
| `parsers` | Parser Functions to use to parse the code and extract documentation. |
| `...` | Other arguments to pass to Parser Functions. |

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

### Examples

```
f <- system.file("silly","R","silly.R",package="inlinedocs")
extract.docs.file(f)
```

---

## extract.docs.setClass *S4 class inline documentation*

---

### Description

Using the same conventions as for functions, definitions of S4 classes in the form `setClass("classname",...)` are also located and scanned for inline comments.

### Usage

```
extract.docs.setClass(doc.link)
```

### Arguments

| | |
|---|---|
| doc.link | DocLink object as created by extract.file.parse. Note that source statements are *ignored* when scanning for class definitions. |

### Details

Extraction of S4 class documentation is currently limited to expressions within the source code which have first line starting with setClass("classname". These are located from the source file (allowing also for white space around the setClass and (). Note that "classname" must be a quoted character string; expressions returning such a string are not matched.

For class definitions, the slots (elements of the representation list) fill the role of function arguments, so may be documented by ##<< comments on the same line or ### comments at the beginning of the following line.

If there is no explicit title on the first line of setClass, then one is made up from the class name.

The class definition skeleton includes an Objects from the Class section, to which any ##details<< documentation chunks are written. It is given a vanilla content if there are no specific ##details<< documentation chunks.

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

---

## extract.file.parse *File content analysis*

---

### Description

Using the base parse function, analyse the file to link preceding "prefix" comments to each active chunk. Those comments form the default description for that chunk. The analysis also looks for S4 class "setClass" calls and R.oo setConstructorS3 and setMethodS3 calls in order to link the documentation of those properly.

### Usage

```
extract.file.parse(code)
```

### Arguments

| | |
|---|---|
| code | Lines of R source code in a character vector - note that any nested source statements are *ignored* when scanning for class definitions. |

### Details

If the definition chunk does not contain a description, any immediately preceding sequence consecutive "prefix" lines will be used instead.

Class and method definitions can take several forms, determined by expression type:

**assignment (<-)** Ordinary assignment of value/function;

**setClass** Definition of S4 class;

**setConstructorS3** Definition of S3 class using R.oo package;

**R.methodsS3::setMethodS3** Definition of method for S3 class using R.oo package.

Additionally, the value may be a name of a function defined elsewhere, in which case the documentation should be copied from that other definition. This is handled using the concept of documentation links.

The `R.methodsS3::setMethodS3` calls introduce additional complexity: they will define an additional S3 generic (which needs documentation to avoid warnings at package build time) unless one already exists. This also is handled by "linking" documentation. A previously unseen generic is linked to the first defining instances, subsequent definitions of that generic also link back to the first defining instance.

### Value

Returns an invisible list of .DocLink objects.

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

---

## `extract.xxx.chunks` *Extract documentation from a function*

---

### Description

Given source code of a function, return a list describing inline documentation in that source code.

### Usage

```
extract.xxx.chunks(src,
    name.fun = "(unnamed function)",
    ...)
```

### Arguments

| | |
|---|---|
| `src` | The source lines of the function to examine, as a character vector. |
| `name.fun` | The name of the function/chunk to use in warning messages. |
| `...` | ignored. |

## Details

For simple functions/arguments, the argument may also be documented by appending ##<< comments on the same line as the argument name. Mixing this mechanism with ### comment lines for the same argument is likely to lead to confusion, as the ### lines are processed first.

Additionally, consecutive sections of ## comment lines beginning with ##*xxx*<< (where *xxx* is one of the fields: `alias`, `details`, `keyword`, `references`, `author`, `note`, `seealso`, `value`, `title` or `description`) are accumulated and inserted in the relevant part of the .Rd file.

For `value`, `title`, `description` and function arguments, these *append* to any text from "prefix" (`^### `) comment lines, irrespective of the order in the source.

When documenting S4 classes, documentation from `details` sections will appear under a section `Objects from the Class`. That section typically includes information about construction methods as well as other description of class objects (but note that the class Slots are documented in a separate section).

Each separate extra section appears as a new paragraph except that:

- empty sections (no matter how many lines) are ignored;
- `alias` and `keyword` sections have special rules;
- `description` should be brief, so all such sections are concatenated as one paragraph;
- `title` should be one line, so any extra `title` sections are concatenated as a single line with spaces separating the sections.

As a special case, the construct ##`describe`<< causes similar processing to the main function arguments to be applied in order to construct a describe block within the documentation, for example to describe the members of a list. All subsequent "same line" ##<< comments go into that block until terminated by a subsequent ##*xxx*<< line.

Such regions may be nested, but not in such a way that the first element in a `describe` is another `describe`. Thus there must be at least one ##<< comment between each pair of ##`describe`<< comments.

When nested `describe` blocks are used, a comment-only line with ##`end`<< terminates the current level only; any other valid ##*xxx*<< line terminates all open describe blocks.

## Value

Named list of character strings extracted from comments. For each name N we will look for N{...} in the Rd file and replace it with the string in this list (implemented in `modify.Rd.file`).

## Note

`alias` extras are automatically split at new lines.

`keyword` extras are automatically split at white space, as all the valid keywords are single words.

The "value" section of a .Rd file is implicitly a describe block and ##`value`<< acts accordingly. Therefore it automatically enables the describe block itemization (##« after list entries).

## Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

`fake_package_env` *fake package env*

## Description

Create a fake package environment in a way that keeps S4 working (so there is a .packageName) and also conforms to byte-code interpreter requirements on environment structure, particularly ensuring that the created environment is a namespace. A similar procedure (with the exception of not deleting objects) is now in testthat (test_pkg_env).

## Usage

```
fake_package_env()
```

## Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

`findGeneric` *findGeneric*

## Description

Copied from R-3.0.1, to support findGeneric.

## Usage

```
findGeneric(fname, envir)
```

## Arguments

| | |
|---|---|
| `fname` | fname |
| `envir` | envir |

## Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## fixPackageFileNames *fixPackageFileNames*

### Description

Copied from R-3.0.1, to support fixPackageFileNames.

### Usage

```
fixPackageFileNames(list)
```

### Arguments

| | |
|---|---|
| list | list |

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## forall *forall*

### Description

For each object in the package that satisfies the criterion checked by subfun, parse source using FUN and return the resulting documentation list.

### Usage

```
forall(FUN, subfun = function(x) TRUE)
```

### Arguments

| | |
|---|---|
| FUN | Function to apply to each element in the package. |
| subfun | Function to select subsets of elements of the package, such as is.function. subfun(x)==TRUE means FUN will be applied to x and the result will be returned. |

### Value

A Parser Function.

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## forall.parsers *forall parsers*

### Description

List of Parser Functions that can be applied to any object.

### Usage

```
"forall.parsers"
```

## forfun *forfun*

### Description

For each function in the package, do something.

### Usage

```
forfun(FUN)
```

### Arguments

| | |
|---|---|
| FUN | FUN |

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## forfun.parsers *forfun parsers*

### Description

Parsers for each function that are constructed automatically. This is a named list, and each element is a parser function for an individual object.

### Usage

```
"forfun.parsers"
```

getKnownS3generics *getKnownS3generics*

## Description

Copied from R-3.0.1, to support getKnownS3generics.

## Usage

```
getKnownS3generics()
```

## Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

getSource *getSource*

## Description

Extract a function's source code.

## Usage

```
getSource(fun.obj)
```

## Arguments

| | |
|---|---|
| fun.obj | A function. |

## Value

Source code lines as a character vector.

## Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

---

`get_internal_S3_generics` *get internal S3 generics*

---

## Description

Copied from R-3.0.1, to support `getKnownS3generics`.

## Usage

```
get_internal_S3_generics(primitive = TRUE)
```

## Arguments

| | |
|---|---|
| `primitive` | primitive |

## Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

---

`get_S3_primitive_generics` *get S3 primitive generics*

---

## Description

Copied from R-3.0.1, to support `getKnownS3generics`.

## Usage

```
get_S3_primitive_generics(include_group_generics = TRUE)
```

## Arguments

| | |
|---|---|
| `include_group_generics` | include_group_generics |

## Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## is_primitive_in_base *is primitive in base*

### Description

Copied from R-3.0.1, to support `getKnownS3generics`.

### Usage

```
is_primitive_in_base(fname)
```

### Arguments

| | |
|---|---|
| `fname` | fname |

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## kill.prefix.whitespace *kill prefix whitespace*

### Description

Figure out what the whitespace preceding the example code is, and then delete that from every line.

### Usage

```
kill.prefix.whitespace(ex)
```

### Arguments

| | |
|---|---|
| `ex` | character vector of example code lines. |

### Value

Character vector of code lines with preceding whitespace removed.

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

`leadingS3generic` *check whether function name is an S3 generic*

## Description

Determines whether a function `name` looks like an S3 generic function

## Usage

```
leadingS3generic(name,
    env, ...)
```

## Arguments

| | |
|---|---|
| `name` | `name` of function |
| `env` | environment to search for additional generics |
| `...` | ignored here |

## Details

This function is one of the default parsers, but exposed as possibly of more general interest. Given a function `name` of the form x.y.z it looks for the generic function x applying to objects of class y.z and also for generic function x.y applying to objects of class z.

Assumes that the first `name` which matches any known generics is the target generic function, so if both x and x.y are generic functions, will assume generic x applying to objects of class y.z

## Value

If a matching generic found returns a list with a single component:

| | |
|---|---|
| `.s3method` | a character vector containing generic `name` and object `name`. |

If no matching generic functions are found, returns an empty list.

## Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## lonely *lonely*

### Description

List of parser functions that operate on single objects. This list is useful for testing these functions.

### Usage

```
"lonely"
```

### Examples

```
  f <- function # title
### description
  (x, ##<< arg x
   y
### arg y
   ){
    ##value<< a list with elements
    list(x=x, ##<< original x value
         y=y, ##<< original y value
         sum=x+y) ##<< their sum
    ##end<<
  }
  src <- getSource(f)
  lonely$extract.xxx.chunks(src)
  lonely$prefixed.lines(src)
```

## make.package.and.check *make package and check*

### Description

Assemble some R code into a package and process it using R CMD check, stopping with an error if the check resulted in any errors or warnings.

### Usage

```
make.package.and.check(f,
    parsers = default.parsers,
    verbose = TRUE)
```

### Arguments

| | |
|---|---|
| f | R code file name from which we will make a package |
| parsers | Parsers to use to make the package documentation. |
| verbose | print the check command line? |

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

---

## modify.Rd.file *modify Rd file*

---

### Description

Add inline documentation from comments to an Rd file automatically-generated by package.skeleton.

### Usage

```
modify.Rd.file(N, pkg,
    docs, verbose = FALSE)
```

### Arguments

| | |
|---|---|
| N | Name of function/file to which we will add documentation. |
| pkg | Package name. |
| docs | Named list of documentation in extracted comments. |
| verbose | Cat messages? |

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

---

## non.descfile.names *non descfile names*

---

### Description

Names of Parser Functions that do NOT use the desc arg.

### Usage

```
"non.descfile.names"
```

## nondesc.parsers *nondesc parsers*

### Description

Parsers that operate only on R code, independently of the description file.

### Usage

```
"nondesc.parsers"
```

## package.skeleton.dx *Package skeleton deluxe*

### Description

Generates Rd files for a package based on R code and DESCRIPTION metadata. After reading the pkgdir/R/*.R code files to find inline documentation (by default R code in *.r files will not be used for inlinedocs), writes docs to pkgdir/man/*.Rd files, possibly overwriting the previous files there.

### Usage

```
package.skeleton.dx(pkgdir = "..",
    parsers = NULL, namespace = FALSE,
    excludePattern = "[.][rsqS]$",
    verbose = FALSE)
```

### Arguments

| | |
|---|---|
| pkgdir | Package directory where the DESCRIPTION file lives. Your code should be in pkgdir/R. |
| parsers | List of Parser functions, which will be applied in sequence to extract documentation from your code. Default NULL means to first search for a definition in the variable "parsers" in pkgdir/R/.inlinedocs.R, if that file exists. If not, we use the list defined in options("inlinedocs.parsers"), if that is defined. If not, we use the package global default in the variable default.parsers. |
| namespace | A logical indicating whether a NAMESPACE file should be generated for this package. If TRUE, all objects whose name starts with a letter, plus all S4 methods and classes are exported. |
| excludePattern | A regular expression matching the files that are not to be processed e.g. because inlinedocs can not handle them yet (like generic function definitions). Default value means to only process inlinedocs in .R files. Set excludePattern=NULL to process all code files, e.g. *.r files. |
| verbose | show messages about parser functions used? |

## Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## Examples

```
owd <- setwd(tempdir())

## get the path to the silly example package that is provided with
## package inlinedocs
testPackagePath <- file.path(system.file(package="inlinedocs"),"silly")
## copy example project to the current unlocked workspace that can
## be modified
file.copy(testPackagePath,".",recursive=TRUE)

## generate documentation .Rd files for this package
package.skeleton.dx("silly")

## check the package to see if generated documentation passes
## without WARNINGs.
if(interactive()){
  cmd <- sprintf("%s CMD check --as-cran silly",file.path(R.home("bin"), "R"))
  print(cmd)
  system(cmd)
}
## cleanup: remove the test package from current workspace again
unlink("silly",recursive=TRUE)
setwd(owd)
```

---

`prefix` *prefix*

---

## Description

Prefix for code comments used with grep and gsub.

## Usage

```
"prefix"
```

`prefixed.lines` *prefixed lines*

## Description

The primary mechanism of inline documentation is via consecutive groups of lines matching the specified `prefix` regular expression "^### " (i.e. lines beginning with "### ") are collected as follows into documentation sections:

**description** group starting at line 2 in the code

**arguments** group following each function argument

**value** group ending at the penultimate line of the code

These may be added to by use of the ##<< constructs described below.

## Usage

```
prefixed.lines(src, ...)
```

## Arguments

| | |
|---|---|
| `src` | src |
| `...` | ... |

## Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## Examples

```
test <- function
### the description
(x,
### the first argument
 y ##<< another argument
 ){
  5
### the return value
##seealso<< foobar
}
src <- getSource(test)
prefixed.lines(src)
extract.xxx.chunks(src)
```

## print.allfun *print allfun*

### Description

Print method for functions constructed using `forall`.

### Usage

```
## S3 method for class 'allfun'
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | x |
| ... | ... |

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## removeAliasesfrom.Rd.file *removeAliasesfrom Rd file*

### Description

remove aliases to methodnames from the Rd file of a class automatically-generated by package.skeleton.

### Usage

```
removeAliasesfrom.Rd.file(N,
    pkg, code)
```

### Arguments

| | |
|---|---|
| N | Name of function/file to which we will add documentation. |
| pkg | Package name. |
| code | The code of the package |

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## replace.one *replace one*

### Description

Do find and replace for one element of an inner documentation list on 1 Rd file.

### Usage

```
replace.one(torep, REP,
    txt, verbose = FALSE)
```

### Arguments

| | |
|---|---|
| torep | tag to find. |
| REP | contents of tag to put inside. |
| txt | text in which to search. |
| verbose | cat messages? |

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## save.test.result *save test result*

### Description

For unit tests, this is an easy way of getting a text representation of the list result of `extract.docs.file`.

### Usage

```
save.test.result(f)
```

### Arguments

| | |
|---|---|
| f | R code file with inlinedocs to process with `extract.docs.file`. |

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

## test.file *test file*

### Description

Check an R code file with inlinedocs to see if the `extract.docs.file` parser accurately extracts all the code inside! The code file should contain a variable .result which is the documentation list that you should get when you apply `extract.docs.file` to the file. We check for identity of elements of elements of the list, so the order of elements should not matter, and thus this should be a good robust unit test.

### Usage

```
test.file(f, CRAN.checks = TRUE,
    verbose = FALSE)
```

### Arguments

| | |
|---|---|
| f | File name of R code file with inlinedocs to parse and check. |
| CRAN.checks | try to make a package and run CRAN checks? |
| verbose | Show output? |

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

### See Also

`save.test.result`

## test.parsers *test parsers*

### Description

List of classic parsers which were default before 2018.

### Usage

```
"test.parsers"
```

`whole.word` *whole word*

## Description

Regex for a whole word to code/link tags.

## Usage

```
whole.word(...)
```

## Arguments

| | |
|---|---|
| ... | ... |

## Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre], Keith Ponting [aut], Thomas Wutzler [aut], Philippe Grosjean [aut], Markus Müller [aut], R Core Team [ctb, cph]

# Index