Package ‘documair’

February 19, 2015

Version 0.6-0

Date 2014-09-06

Type Package

Title Automatic Documentation for R packages

Author Jean-Baptiste Denis <Jean-Baptiste.Denis@jouy.inra.fr>, Regis Pouillot <RPouillot@yahoo.fr>, Kien Kieu <Kien.Kieu@jouy.inra.fr>

Maintainer Jean-Baptiste Denis <Jean-Baptiste.Denis@jouy.inra.fr>

Description Production of R packages from tagged comments introduced within the code and a minimum of additional documentation files.

License GPL (>= 2.15)

Repository CRAN

Repository/R-Forge/Project riskassessment

Repository/R-Forge/Revision 283

Repository/R-Forge/DateTimeStamp 2014-09-17 09:16:39

Date/Publication 2014-09-22 17:22:20

NeedsCompilation yes

R topics documented:

documair-package *Automatic Documentation for (and confection of) R packages*

Description

The development of this package was led by the idea that it is a bad practice to separate code and documentation. Code and documentation should be at the same place so that both can easily be modified simultaneously. We proceed according to this principle since the middle of the 90ties using a Perl script (a very convenient language for this kind of tasks). Now, we share it with the community by the more convenient way of an R package. The recent development of R make things quite easy.
To use **documair**, each code defining an object (either function or variable) must be encapsulated within a series of tagged comments; we propose tags but their values can be adapted as you wish (by modifying the '`documair0$tag$v`' list). From these tagged comments, **documair** automatically writes '`Rd`' files and gathers them with a few more files to produce the complete package until the '`tar.gz`'. For some specific objects, the user can write manually the '`Rd`' file. All the necessary files must be gathered in a unique directory. In it, the user must place the following set of mandatory files (in the following '`pkg`' will designate the name of the package):

1. A '`pkg.DESCRIPTION`' file: the standard text DESCRIPTION file to be associated to the package. The NAMESPACE file is automatically created from the exported objects and the presence of C and Fortran files.
2. A '`pkg.package.Rd`' file: a text file describing in Rd syntax the general description of the package to appear in the documentation. This file can be slightly supplemented by **documair** to add some additional information.
3. As many as wanted '`foo.code.r`' files where are placed the documented code of each object. Each files can include more than one object. The extension '`code.r`' can be modified within the '`pkg.which.txt`' file.

Additional optional files can also be included:

- '`foo.test.r`' files including some scripts to test the functions. The extension '`test.r`' can be modified within the '`pkg.which.txt`' file.
- '`pkg.foo.rda`' files where are placed possible data sets. These binary files must be loadable with the function '`load`'. The associated '`pkg.foo.Rd`' documentation files must be provided.
- '`object(s).Rd`' files the user wants to produce by hand. They will be used by **documair** instead of the one based on the tagged code for (and only for) exported objects. By default, all the objects are exported, but some can be declared hidden with the '`pkg.which.txt`' file (see below).
- '`C`' functions must be stored into individual files with the same name and extension '`.c`'; the same for '`Fortran`' functions with extension '`.f`'.
- An optional '`pkg.which.txt`' file allows the user to override the standard use of files and functions to prepare different versions of the package with the same set of code files. It gives the possibility to hide or not functions in the package as well as to get sets of aliased objects. For details see the example below.

- Every additional file the author wants to share with the other users of the package.

The complete content of this directory will be copied in the free ‘inst’ directory of the package arborescence. Once this directory is prepared, the user successively calls the ‘documair’ functions ‘prepare8pkg’ and ‘compile8pkg’.

The denomination of **documair** stands for *documentation for R*, ’air’ having the same pronunciation as ’R’ in French.

Other R packages for generating Rd documentations files and building packages from comments inserted in R code are available. Two outstanding examples are **roxygen** and **inlinedocs**. The former is based on header comments, uses powerful parsers and propose interesting analyses like the call tree of the set of functions. The latter is very light, using simple tagging in logical places of the code. **documair** is in between these two cases, tags are also within the code but are many and varied giving rise to more possibilities than **inlinedocs**.

Documenting an object

Here is an example of a simple masked function of **documair**.

```
‘#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<’
‘r.bd <- function(n1,n2)’
‘#TITLE  sequence of increasing numbers’
‘#DESCRIPTION’
‘# This function returns {n1:n2} when {n1<=n2} and’
‘#        {numeric(0)} otherwise.’
‘# Quite useful when some insertion must be done within’
‘# a sequence’
‘#DETAILS’
‘#KEYWORDS iteration’
‘#INPUTS’
‘#{n1}    <<first integer>>’
‘#{n2}    <<second integer>>’
‘#[INPUTS]’
‘#VALUE’
‘# {n1:n2} if {n1<n2}’
‘# else {numeric(0)}.’
‘#EXAMPLE’
‘# xx <- 1:5;’
‘# for (ii in 1:6) { print(c(xx[bd(1,ii-1)],10,xx[bd(ii,5)]));}’
‘#REFERENCE’
‘#SEE ALSO bf’
‘#CALLING’
‘#COMMENT’
‘#FUTURE’
‘#AUTHOR J.-B. Denis’
‘#CREATED 11_01_12’
‘#REVISED 11_05_21’
‘#------------------------------------------’
```

```
‘{’
‘if (n1 <= n2) {return(n1:n2);}’
‘numeric(0);’
‘}’
‘#>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>’
```

One can notice the different tags used to structure the information provided in the comments lines:

- ‘#<<<<<<<’. . . ‘<<<<<<<<’ to open the object,
- ‘#TITLE’ to specify the title of the object,
- ‘#DESCRIPTION’ to describe the object,
- . . . ,
- ‘#-------’. . . ‘--------’ to close the documentation part
- . . . ,
- ‘#>>>>>>>’. . . ‘>>>>>>>>’ to close the object.

Those five tags are the only compulsory ones (Note: the ‘#DESCRIPTION’ tag can be missing for the children objects of a set of aliased objects).

Monitoring documair

Introduction:

documair can build different packages from the same set of files by simply selecting which files/objects have to be compiled and which ones must be proposed or hidden to the end user. This is performed through the ‘pkg.which.txt’ text file. An example of such a file (with comments) is ‘documair.which.txt’ of the **documair** package. If you are generating different packages with the same functions, it is recommended to change the name of the package, this is the reason why you can indicate in the ‘pkg.which.txt’ file other ‘DESCRIPTION’ and PRESENTATION files than the standard ones deduced from the package name.

syntax of ‘pkg.which.txt’:

The documentation of **documair** was performed by itself. Below is provided, as an example, a possible ‘documair.which.txt’. The comments introduced in the file are self-sufficient to understand what are the different possibilities:

```
‘#’
‘# created       on 14_01_28’
‘# last modified on 14_05_27 ’
‘#’
‘# + the order of the items is irrelevant’
‘#   but they are exploited in that order’
‘# + according to an option, the existence’
‘#   of the specified file, objects is’
‘#   checked or not.’
‘# + '_ALL_' means all occurences’
‘#’
‘# specifying the description file’
‘<<DESCRIPTION>> documair.DESCRIPTION’
‘#’
```

```
‘# the Rd file to describe the package’
‘<<PRESENTATION>> documair.package.Rd’
‘#’
‘# specifying the extension for the code files’
‘<<C.EXTE>> code.r’
‘#’
‘# specifying the extension for the test files’
‘<<T.EXTE>> test.r’
‘#’
‘# specifying hidden code files’
‘# (in the example, all objects are hidden)’
‘<<HIDDEN.F>> _ALL_’
‘#’
‘# specifying exported code files and’
‘# modifying previous prescriptions’
‘<<EXPORTED.F>> user.code.r’
‘#’
‘# specifying exported code files containing alias sets’
‘<<ALIASED.F>> exterieur.code.r’
‘#’
‘# specifying hidden objects’
‘# (in the example no specific object should be hidden’
‘#  so the item here is suppressed with an '#'.)’
‘#<<HIDDEN.O>> ’
‘#’
‘# specifying exported objects’
‘# (in the example an object belonging to a hidden file’
‘#  (documair0) is exported)’
‘<<EXPORTED.O>> documair0’
‘#’
‘# specifying objects for which’
‘#  the content will be displayed on the’
‘#  screen during the process.’
‘# (the same can be done at the level of the’
‘#  files with '<<DISPLAY.F>>'.)’
‘<<DISPLAY.O>> documair0 display8tags’
‘#’
‘# specifying keywords from the components’
‘# (here 'compile' must be interpreted as’
‘#  'compilation', 'pkg' as 'package',...’
‘#  be aware that the two words must be stuck’
‘#  with '=' and that a word must not comprise’
‘#  any blank. Also that the special word '_NO_'’
‘#  means that this component must not appear’
‘#  as a keyword.)’
‘<<KEYWORDS>>/=/U’
‘compile=compilation’
‘pkg=package’
```

‘prepare=preparation’
‘documair=_NO_’
‘#’
‘# end of the which.documair.txt file’

Aliasing:
documair accepts the aliasing of set of objects but some rules must be followed for that.

1. Objects sharing the same alias must be proposed into a single file, and only those objects should be present in this file.
2. The parent object must be in the first position into this file.
3. The first alias name of the parent object must be the common alias
4. The name of the file must be declared as containing aliased objects in the (in that case mandatory) ‘pkg.which.txt’ file after the tag ‘<<ALIASED.F>>’.

The alias ‘Rd’ file can be either provided by the user or composed by **documair** from the documenting tags. When the ‘Rd’ file is hand-written, it must be named under the first object (the parent alias). When the documentation is built by **documair**, the descriptions of identical arguments are taken in the first object using them according to the ordering within the file.

Examples:
To get more insights about the flexibility of **documair**, the reader can have a glance to **documair** itself since all necessary files are gathered in the ‘inst’ directory. In the same directory is prepared within the script ‘make.r’ four examples of building package variations based on **documair** objects; the first one being **documair** itself. The second one (named **documair1** proposes the building of the package without ‘pkg.which.txt’ file meaning that all objects are exported. The fourth example (**documair3** gives an example of using ‘C’ and ‘Fortran’ functions which can be not effective for some configurations. Due to minor inconsistencies, some examples generate warnings.

Errors with documair

Currently, **documair** is quite sensitive to errors in the input files! Some are detected but indications are not always very clear, others are not detected. For instance the double ‘#’ in ‘##{argument} << explanation...>>’ when describing the arguments of a function causes a non explicit error. Also, it can be easily affected by a mismatched parenthesis... To help the user in seeing where the mistake is located, it is suggested to put the ‘check’ argument of ‘prepare8pkg’ to ‘TRUE’ and introduce the line ’‘<<DISPLAY.O>> _ALL_” in the used ‘pkg.which.txt’ file. This way the interpreted content of each object of the package will be displayed on the screen during the process with a pause to give an opportunity to see if everything seems consistent. It is strongly advised:

- to introduce only standard ascii characters, even in the comments.
- to test regularly the preparation of a ‘tar.gz’ with **documair** during the development of a package, rather than once at the end, in order detect more easily the origins of potantial issues.
- to avoid functions with name having more than one dot (‘.’). It is considered as a method for an S3 object by **documair** but only the last dot is taken into account.

- to be aware that the tagging of '`documair0$tags$v$deb$v`' which is '`"#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<"`'.
  must be exactly respected as well as those of '`documair0$tags$v$fin$v`' and '`"#>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>"`'
  Of course, you can modify them as you wish.

Naming conventions

Remembering object names in a given package is not always an easy task, the first reason being their number. To relieve it, we propose to follow some conventions, if so **documair** will propose deduced keywords.

- Names will be composed of *name components* separated with digits. For instance '`print8object`' has '`print`' and '`object`' as components. The number of components can be one, two,. . . or more.
- When nouns, the components can be singular or plural, making differences.
- The separating digits have a meaning:
  0 pkg0 (object proposed by the package **pkg**) /0 ~ Object/
  1 res4objA1objB (and; here 'res' obtained from 'objA' and 'objB') /1 = one = an ~ and/
  2 a2b (conversion from object a to object b, even if the conversion is not one-to-one, a reverse function b2a is supposed to exist) / 2 ~ to/
  3 series3fun (function fun belongs to the family series) /3 ~ \in/
  4 trait4object (extract some characteristic from an object) /4 = Four ~ From/
  5 <free for the moment>
  6 split8text6tag (split a text object [with | by means of] tags) /6 = sIx ~~ wIth/
  7 image7path (image path) /upper bar of 7 is similar to an hyphen used to join two nouns or an adjective and a noun./
  8 action8object (make an action on an (or several) object(s)) /8 ~ a/
  9 empty7object9 (is it an empty object?: question mark) /9 ~~ ?/

Projected evolution of documair

- Make possible to have operator functions like '`%T%`'.
- Allow the user not to have too many '`*.code.r`' files. For that, the code files could be first splitted into elementary sub-files with a splitting tag like '#<<<--->>>'. This would be convenient when there are numerous aliased object sets. Indeed, currently, each identically aliased set of objects must be in a distinct file.
- Get the main steps of the algorithm used in the function by collecting some tag contents within the code (introduced as special titles) and added either as a special section or as a new paragraph in the details section.
- Allows the possibility of including files for repeated pieces of code, at least for one level.
- Allows the introduction of enumerations in the comments.
- As an option, check and impose typographical rules like upper cases at the beginning and final dot to the argument description... with some indication in a separate file of the changes to give the user the opportunity to check them.
- Improve the way aliased objects are documented, allowing collective fields like titles...

Acknowledgment

The authors want to thank Annie Bouvier and Caroline Bidot for their useful supports in solving some technical difficulties we had when elaborating the package.

Additional Information

- This package was built with /documair/ package (version 0.6-0) on 14_09_15
- There are 65 object(s) in total. 11 are exported and there exist 54 masked object(s): analyse8description, code7objects4text6tags, components9, extract8object, make8rd, parse8code, rrrbc, rrrbd, rrrbelong9, rrrbf, rrrdipa, rrrdisplay8k, rrrerreur, rrrexplore8list, rrrfidi9, rrrfile2list, rrrfile2text, rrrfilter8text, rrrform3crop, rrrform3display, rrrform3justify, rrrform3parag, rrrform3repeat, rrrform3title, rrrform3titre, rrrget8comp7list, rrrinterv7belonging, rrrlist2text, rrrlist4text, rrrnow, rrrobject9, rrrparse8text, rrrpause, rrrplaces4text6tags, rrrrbsa0, rrrrbsa7list9, rrrtext2file, rrrtext2list, rrrtext2vma, rrrtext3acceptance, rrrtext3brackets, rrrtext3ij2n, rrrtext3interval, rrrtext3n2ij, rrrtext3places8brackets, rrrtext3places8word, rrrtext3preparation, rrrtext3replace, rrrtext3stext, rrrtext3translate, rrrtexts4text, rrrvma2text, rrrvoid9, write8lyse.
- There were 1 C file(s).
- There were 1 Fortran file(s).
- They were provided through 8 code file(s).

Author(s)

Jean-Baptiste Denis (MIAj - Inra - Jouy-en-Josas),
R\'egis Pouillot and
Ki\^en Ki\^eu (MIAj - Inra - Jouy-en-Josas)

---

build8pkg *builds a package*

Description

From the documair directory containing at least the description file (‘pkg.DESCRIPTION’), the presentation file (‘pkk.pacakge.Rd’) and the code files (‘*.code.r’) launches the preparation, the checking and the building by **R** itself.
Resulting ‘pdf’ manual and ‘pkg.tar.gz’ files are placed into the indicated destination directory. Some intermediate directory (especially the ‘pkg.Rcheck’ directory) are placed then deleted (in case of no error) into the within directory: so the possibility to create and remove them there is necessary. To look at the intermediate results, ‘prepare8pkg’ and ‘compile8pkg’ must be directly used, or ‘l’ for ‘left’ must be included in the argument ‘what’.

Usage

```
build8pkg(pkg,documair7dir,destination7dir,
          what="pz",display=TRUE,
          signature=2,check=FALSE)
```

Arguments

`pkg`	Name of the package.
`documair7dir`	Directory where are gathered all necessary files for **documair**.
`destination7dir`	Directory where the resulting files have to be placed. If not existing will be created.
`what`	'`character(1)`'. Indicates the actions to perform. It is lowercased before decoding. Wwhen it comprises '`c`' the check is performed; when it comprises '`z`' the tar.gz is generated; when it comprises '`p`' the pdf manual is generated; when it comprises '`l`' intermediate directories are left.
`display`	Must indications about the process progress be displayed onto the screen?
`signature`	'`numeric(1)`'. Must additional information be included in the '`pkg.package.Rd`' file? When '`0`' no; when '`1`' just the name of **documair**; when '`2`' plus the list of the masked functions.
`check`	Must checking be done by the user after each debugging display? Checking means that the program rrrpause after each display awaiting for an answer from the user to continue or stop.

Details

Whatever is the argument '`what`', the preparation of the arborescence is made.
Checking with fatal errors can be issued by the functions called by '`build8pkg`'. If this occurs, it means that the check made by '`build8pkg`' are not sufficient: please indicate us it.

Value

'`character(0)`' when all went right if not a '`character`' reporting the motives of difficulties.

Future

Make '`display`' more efficient when '`FALSE`'.

See Also

`prepare8pkg` `compile8pkg`

`change8names` *changes object names in R script files*

Description

Explores a series of R script files and modify them changing some objects names for new ones.
Changes must be unambiguous; they are performed sequentially according to the order of '`nnames`' rows. Also, if some functions have got common names used in comments or messages, they will be changed as well since the syntaxic analysis made in '`rrrtext3places8word`' is very limited.

Usage

```
change8names(nfiles,nnames)
```

Arguments

`nfiles`	A two column '`character`' matrix, giving by row old and new names of files to consider. Old and new names can be identical.
`nnames`	A two column '`character`' matrix, giving by row old and new names of objects.

Details

It is difficult to ensure that the modifications are perfect, but at least they can give a hand.
Each file is dealt in turn. When old and new names of a file coincide, the old file is first saved into a temporary file suffixed with the moment of the saving and only destroyed at the end. So if something went wrong, you can recover the initial file.

Value

'`character(0)`' if everything went right, if not, some warning messages.

`compile8pkg` *compiles a package*

Description

From the standard directory containing the package sources, launches the checking and building by **R** itself. Pdf manual and tar ball can be placed in another directory.

Usage

```
compile8pkg(pkg,pkgdir,
            chkdir=getwd(),resdir=getwd(),
            what="pz",
            display=FALSE)
```

Arguments

`pkg`	Name of the package.
`pkgdir`	Directory where the building of the package has been prepared. Probably by the function '`prepare8pkg`'.
`chkdir`	Directory where to put the directory where "R" makes the checking. Must exist.
`resdir`	Directory where to copy the resulting files. Must exist.
`what`	'`character(1)`'. Which resulting file to copy: '`p`' for the pdf manual and '`z`' for the tar.gz ball.
`display`	Must indications about the process progress be displayed on the screen?

Value

Nothing but the check, the compilation, [the copy of the manual, the tar.gz] are performed.

Future

Make '`display`' more efficient when '`FALSE`'.

See Also

`prepare8pkg`

`copy8objects` *generates a file comprising indicated functions*

Description

Explores a series of R code files supposed to comprise the needed functions with **tagging** and copy them into a new not already existing file.

Usage

```
copy8objects(nfiles,nobjects,file)
```

Arguments

`nfiles`	Files where to look for the functions.
`nobjects`	A '`character`' indicating the objects to pick up from the indicated files.
`file`	Name of the file to create.

Details

Standard tags defined by '`documair0$tags`' are used.

Value

'`character(0)`' if everything went right, if not, some warning messages about found difficulties.

display8tags *returns the contents of a 'tags' object*

Description

Typically, 'tags' is either 'documair0$tags' or 'documair0$tgs' which defines the tagging of the R code to introduce the documentation to be tackled by /documair/.
Such a function is provided to help the user in seeing the present constants, and possibly to change their value.

Usage

```
display8tags(tags=documair0$tags,
             what="values",imp=FALSE)
```

Arguments

tags	List to be displayed.
what	a 'character(1)' indicating what must be returned. The acceptable values are 'names', 'definitions', 'usage', 'content', 'preparation', 'title' and 'values'. For user convenience, the argument is lowercased before checked, and only the first character is taken into consideration.
imp	indicates if an adapted printing of the results must be done.

Details

More details on the possibilities offered with argument 'what' are obtained into 'documair0$tags$d'.

Value

A character or a list according to 'what'

Future

Make an equivalent function to change the tags values

documair0 *list of the /documair/ constants*

Description

Just a list of constants defined with a name, a definition and a value. They can be modified by the user. Of course the modification must be consistent with the role of the constant, and only values should be modified.

Details

It is a named list, one component for each constant. A sublist is associated to each constant with two components: '`$d`' for the definition and '`$v`' for the value. Be aware that the value can be any object (vector, list, matrix, function,...)

Value

A list (it is a self-documented object).

Examples

```
names(documair0);
print(documair0$text1);
print(documair0$text2$v);
```

dpert *The (Modified) PERT Distribution*

Description

((just an example extracted from the **mc2d** package to illustrate the use of reduced documentation with a series of related and aliased functions.))
Density, distribution function, quantile function and random generation for the PERT (*aka* Beta PERT) distribution with minimum equals to '`min`', mode equals to '`mode`' and maximum equals to '`max`'.

Usage

```
dpert(x,min=-1,mode=0,max=1,shape=4,log=FALSE)
ppert(q,min=-1,mode=0,max=1,shape=4,lower.tail=TRUE,log.p=FALSE)
qpert(p,min=-1,mode=0,max=1,shape=4,lower.tail=TRUE,log.p=FALSE)
rpert(n,min=-1,mode=0,max=1,shape=4)
```

Arguments

`x,q`	Vector of quantiles.
`p`	Vector of probabilities.
`n`	Number of observations. If length(n) > 1, the length is taken to be the number required.
`min`	Vector of minima.
`mode`	Vector of modes.
`max`	Vector of maxima.
`shape`	Vector of scaling parameters. Default value: 4.
`log, log.p`	Logical; if ‘`TRUE`’, probabilities ‘`p`’ are given as ‘`log(p)`’.
`lower.tail`	Logical; if ‘`TRUE`’ (default), probabilities are ‘`P[X <= x]`’, otherwise, ‘`P[X > x]`’.

Details

The PERT distribution is a beta distribution extended to the domain ‘`[min, max]`’ with mean

$$\mu = \frac{min + shape \times mode + max}{shape + 2}$$

The underlying beta distribution is specified by α_1 and α_2 defined as

$$\alpha_1 = \frac{(\mu - min)(2 \times mode - min - max)}{(mode - \mu)(max - min)}$$

$$\alpha_2 = \frac{\alpha_1 \times (max - \mu)}{mu - min}$$

If $\mu = mode$, α_1 is set to $1 + \nu/2$. David Vose proposed a modified PERT distribution with a shape parameter different from 4. The PERT distribution is frequently used to translate expert estimates of the min, max and mode of a random variable in a smooth parametric distribution.

Value

‘`dpert`’ gives the density, ‘`ppert`’ gives the distribution function, ‘`qpert`’ gives the quantile function, and ‘`rpert`’ generates random deviates.

Examples

```
curve(dpert(x,min=3,mode=5,max=10,shape=6), from = 2, to = 11, lty=3)
curve(dpert(x,min=3,mode=5,max=10), from = 2, to = 11, add=TRUE)
curve(dpert(x,min=3,mode=5,max=10,shape=2), from = 2, to = 11, add=TRUE,lty=2)
legend(x = 8, y = 2, c("Default","shape:2","shape:6"), lty=1:3)
```

`prepare8pkg` *prepares a package*

Description

Prepares all necessary directories and files for the building of an R package named ‘`pkg`’ from files made by the user and placed in the ‘`perdir`’ directory. The built arborescence will placed into the preexisting ‘`pkgdir`’ directory.
When an error occurs, it is not always very simple to know the origin of it. To get some clues, the user is suggested to switch the display options on within the ‘`pkg.which.txt`’ file and putting the ‘`check`’ argument to ‘`TRUE`’.

Usage

```
prepare8pkg(pkg,perdir,pkgdir,
            signature=2,
            display=TRUE,
            check=FALSE)
```

Arguments

`pkg`	Name of the package, associated files like ‘`pkg.DESCRIPTION`’ must exist into ‘`perdir`’ directory.
`perdir`	Directory where the prepared files have to be found.
`pkgdir`	Directory where the building of the package has to be prepared. It is supposed to have already been created, at least empty. The result of the preparation of the package will be placed within it with a directory having the package name. In case, it is not empty, it is first completely cleaned of its contents.
`signature`	‘`numeric(1)`’. Must additional information be included in the ‘`pkg.package.Rd`’ file? When ‘`0`’ no; when ‘`1`’ just the name of **documair**; when ‘`2`’ plus the list of the masked functions.
`display`	Must the programmed debbuging displays be performed?
`check`	Must checking be done by the user after each debugging display? Checking means that the program rrrpause after each display awaiting for an answer from the user to continue or stop.

Details

The behaviour of ‘`prepare8pkg`’ with respect to files and objects coded in them, is also driven by a possible ‘`pkg.which.txt`’ file where can be indicated which objects are exported or hidden, which files contains a series of aliased objects, which files and/or objects must be displayed. See the general description of the package for details.
When no such file is present, the default behavior is that all object are exported, no series of objects are aliased, the displaying is just listing the explored files and the objects they contain. Notice that the default displaying as well as the ‘`which`’ file display options can be cancelled with the argument ‘`display`’.

Value

Nothing but the preparation is made (see the description section) with possible displays to and checks from the user.

Future

Introduce more sections and improve the existing ones.

See Also

`compile8pkg` `display8tags`

Index