Parsing Rd files

Duncan Murdoch

October 4, 2009

Abstract

R 2.9.x introduced a parser for Rd format help files. When integrated into the build/install process in R 2.10.x, it will allow easier processing, easier syntax checking, and eventually, easier conversion to other formats.

To write this parser, it was necessary to make some small changes to the specification of the format as described in Writing R Extensions, and to make some choices when that description was ambiguous. Some existing Rd files do not meet the stricter format requirements (and some were incorrectly formatted under the old requirements, but the errors were missed by the older checks). The new stricter format is being informally called Rdoc version 2.

R 2.10.x will include some changes to the format, including some Sweave-like ways to include executable code in R documentation files.

This document describes the new format, necessary changes to existing Rd files, and the structure returned from the `parse_Rd()` function. It also includes some documentation of the new `\Sexpr` markup macro.

1 Introduction

Prior to this document, the R documentation (Rd) file format did not have a formal description. It grew over time, with pieces added, and R and Perl code written to process it.

This document describes a formal parser (written in Bison) for the format, and the R structure that it outputs. The intention is to make it easier for the format to evolve, or to be replaced with a completely new one.

The next section describes the syntax; section 3 describes changes from what is described in Writing R Extensions for R 2.9.0. The following section describes the `parse_Rd()` function and its output. Some additions to the specification have been made in R-devel (to become R 2.10.x); these are noted earlier and discussed in more detail in the final section.

Except for the new `\Sexpr` macro, this document does not describe the interpretation of the markup in Rd files; see Writing R Extensions for that.

2 Rd Syntax Specification

Rd files are text files with an associated encoding, readable as text connections in R. The syntax only depends on a few ASCII characters, so at the level of this specification the encoding is not important, however higher level interpretation will depend on the text connection having read the proper encoding of the file.

There are three distinct types of text within an Rd file: LaTeX-like, R-like, and verbatim. The first two contain markup, text and comments; verbatim text contains only text and comments.

2.1 Encodings

The parser works on connections, which have an associated encoding. It is the caller's responsibility to declare the connection encoding properly when it is opened.

Not all encodings may be supported, because the libraries R uses cannot always perform conversions. In particular, Asian double byte encodings are likely to cause trouble when that is not the native encoding. Plain ASCII, UTF-8 and Latin1 should be supported on all systems when the system is complete.

2.2 Lexical structure

The characters `\`, `%`, `{`, and `}` have special meaning in almost all parts of an Rd file; details are given below. The exceptions are listed in section 2.7.

End of line (newline) characters mark the end of pieces of text. The end of a file does the same. The newline markers are returned as part of the text.

The brackets [and] have special meaning in certain contexts; see section 2.3.

Whitespace (blanks, tabs, and formfeeds) is included in text.

2.2.1 The backslash

The backslash \ is used as an escape character: \\, \%, \{ and \} remove the special meaning of the second character. The parser will drop the initial backslash, and return the other character as part of the text.

The backslash is also used as an initial character for markup macros. In an R-like or LaTeX-like context, a backslash followed by an alphabetic character starts a macro; the macro name continues until the first non-alphanumeric character. If the name is not recognized the parser drops all digits from the end, and tries again. If it is still not recognized it is returned as an UNKNOWN token.

All other uses of backslashes are allowed, and are passed through by the parser as text.

2.2.2 The percent symbol

An unescaped percent symbol % marks the beginning of an Rd comment, which runs to the end of the current line. The parser returns these marked as COMMENT tokens. The newline or EOF at the end of the comment is not included as part of it.

2.2.3 Braces

The braces { and } are used to mark the arguments to markup macros, and may also appear within text, provided they balance. In R-like and verbatim text, the parser keeps a count of brace depth, and the return to zero depth signals the end of the token. In Latex-like mode, braces group tokens; groups may be empty. Opening and closing braces on arguments and when used for grouping in Latex-like text are not returned by the parser, but balanced braces within R-like or verbatim text are.

2.3 Markup

Markup includes macros starting with a backslash \ with the second character alphabetic. Some macros (e.g. \R) take no arguments, some

(e.g. `\code{}`) take one argument surrounded by braces, and some (e.g. `\section{}{}`) take two arguments surrounded by braces.

There are also three special classes of macros. The `\link{}` macro (and `\Sexpr` in R 2.10.x) may take one argument or two, with the first marked as an option in square brackets, e.g. `\link[option]{arg}`. The `\eqn{}` and `deqn{}` macros may take one or two arguments in braces, e.g. `\eqn{}` or `\eqn{}{}`.

The last special class of macros consists of `#ifdef ... #endif` and `#ifndef ... #endif`. The `#` symbols must appear as the first character on a line starting with `#ifdef`, `#ifndef`, or `#endif`, or it is not considered special.

These look like C preprocessor directives, but are processed by the parser as two argument macros, with the first argument being the token on the line of the first directive, and the second one being the text between the first directive and the `#endif`. Any text on the line following the `#endif` will be discarded. For example,

```
#ifdef unix
Some text
#endif (text to be discarded)
```

is processed with first argument `unix` (surrounded by whitespace) and second argument `Some text`.

Markup macros are subdivided into those that start sections of the Rd file, and those that may be used within a section. The sectioning macros may only be used at the top level, while the others must be nested within the arguments of other macros. (The `\Sexpr` macro, to be introduced in R 2.10.0, is normally used within a section, but can be used as a sectioning macro. See section 5.)

Markup macros are also subdivided into those that contain list-like content, and those that don't. The `\item` macro may only be used within the argument of a list-like macro. Within `\enumerate{}` or `\itemize{}` it takes no arguments, within `\arguments{}`, `\value{}` and `\describe{}` it takes two arguments.

Finally, the text within the argument of each macro is classified into one of the three types of text mentioned above. For example, the text within `\code{}` macros is R-like, and that within `\samp{}` or `\verb{}` macros is verbatim.

The complete list of Rd file macros is shown in Tables 1 to 3.

Table 1: Table of sectioning macros.

Macro	Arguments	Section?	List?	Text type
`\arguments`	1	yes	`\item{}{}`	Latex-like
`\author`	1	yes	no	Latex-like
`\concept`	1	yes	no	Latex-like
`\description`	1	yes	no	Latex-like
`\details`	1	yes	no	Latex-like
`\docType`	1	yes	no	Latex-like
`\encoding`	1	yes	no	Latex-like
`\format`	1	yes	no	Latex-like
`\keyword`	1	yes	no	Latex-like
`\name`	1	yes	no	Latex-like
`\note`	1	yes	no	Latex-like
`\references`	1	yes	no	Latex-like
`\section`	2	yes	no	Latex-like
`\seealso`	1	yes	no	Latex-like
`\source`	1	yes	no	Latex-like
`\title`	1	yes	no	Latex-like
`\value`	1	yes	`\item{}{}`	Latex-like
`\examples`	1	yes	no	R-like
`\usage`	1	yes	no	R-like
`\alias`	1	yes	no	Verbatim
`\Rdversion`	1	yes	no	Verbatim
`\synopsis`	1	yes	no	Verbatim
`\Sexpr`	option plus 1	sometimes	no	R-like
`\RdOpts`	1	yes	no	Verbatim

Table 2: Table of markup macros within sections taking LaTeX-like text.

Macro	Arguments	Section?	List?	Text type
\acronym	1	no	no	Latex-like
\bold	1	no	no	Latex-like
\cite	1	no	no	Latex-like
\command	1	no	no	Latex-like
\dfn	1	no	no	Latex-like
\dQuote	1	no	no	Latex-like
\email	1	no	no	Latex-like
\emph	1	no	no	Latex-like
\file	1	no	no	Latex-like
\item	special	no	no	Latex-like
\linkS4class	1	no	no	Latex-like
\pkg	1	no	no	Latex-like
\sQuote	1	no	no	Latex-like
\strong	1	no	no	Latex-like
\var	1	no	no	Latex-like
\describe	1	no	\item{}{}	Latex-like
\enumerate	1	no	\item	Latex-like
\itemize	1	no	\item	Latex-like
\enc	2	no	no	Latex-like
\if	2	no	no	Latex-like
\method	2	no	no	Latex-like
\S3method	2	no	no	Latex-like
\S4method	2	no	no	Latex-like
\tabular	2	no	no	Latex-like
\link	option plus 1	no	no	Latex-like

Table 3: Table of markup macros within sections taking no text, R-like text, or verbatim text.

Macro	Arguments	Section?	List?	Text type
`\cr`	0	no	no	
`\dots`	0	no	no	
`\ldots`	0	no	no	
`\R`	0	no	no	
`\tab`	0	no	no	
`\code`	1	no	no	R-like
`\dontshow`	1	no	no	R-like
`\donttest`	1	no	no	R-like
`\testonly`	1	no	no	R-like
`\dontrun`	1	no	no	Verbatim
`\env`	1	no	no	Verbatim
`\kbd`	1	no	no	Verbatim
`\option`	1	no	no	Verbatim
`\out`	1	no	no	Verbatim
`\preformatted`	1	no	no	Verbatim
`\samp`	1	no	no	Verbatim
`\special`	1	no	no	Verbatim
`\url`	1	no	no	Verbatim
`\verb`	1	no	no	Verbatim
`\deqn`	1 or 2	no	no	Verbatim
`\eqn`	1 or 2	no	no	Verbatim

2.4 LaTeX-like text

LaTeX-like text in an Rd file is a stream of markup, text, and comments.

Text in LaTeX-like mode consists of all characters other than markup and comments. The white space within text is kept as part of it. This is subject to change.

Braces within LaTeX-like text must be escaped as `\{` and `\}` to be considered as part of the text. If not escaped, they group tokens in a LIST group.

Brackets `[` and `]` within LaTeX-like text only have syntactic relevance after the `\link` or `\Sexpr` macros, where they are used to delimit the optional argument.

Quotes `"`, `'` and `` ` `` have no syntactic relevance within LaTeX-like text.

2.5 R-like text

R-like text in an Rd file is a stream of markup, R code, and comments. The underlying mental model is that the markup could be replaced by suitable text and the R code would be parseable by `parse()`, but `parse_Rd()` does not enforce this.

There are two types of comments in R-like mode. As elsewhere in Rd files, Rd comments start with `%`, and run to the end of the line.

R-like comments start with `#` and run to either the end of the line, or a brace that closes the block containing the R-like text. Unlike Rd comments, the Rd parser will return R comments as part of the text of the code; the Rd comment will be returned marked as a COMMENT token.

Quoted strings (using `"`, `'` or `` ` ``) within R-like text follow R rules: the string delimiters must match and markup and comments within them are taken to be part of the strings and are not interpreted by the Rd parser. This includes braces and R-like comments, but there are two exceptions:

1. `%` characters must be escaped even within strings, or they will be taken as Rd comments.
2. The sequences `\l` or `\v` in a string will be taken to start a markup macro. This is intended to allow `\link` or `\var` to be used in a string (e.g. the common idiom `\code{"\link{someclass}"}`). While `\l` is not legal in an R string, `\v` is a way to encode the

rarely used "vertical tab". To insert a vertical tab into a string within an Rd file it is necessary to use `\\v`.

Braces within R-like text outside of quoted strings must balance, or be escaped.

Outside of a quoted string, in R-like text the escape character `\` indicates the start of a markup macro. No markup macros are recognized within quoted strings except as noted in 2 above.

2.6 Verbatim text

Verbatim text within an Rd file is a pure stream of text, uninterpreted by the parser, with the exceptions that braces must balance or be escaped, and `%` comments are recognized, and backslashes that could be interpreted as escapes must themselves be escaped.

No markup macros are recognized within verbatim text.

2.7 Exceptions to special character handling

The escape handling for `\`, `{`, `}` and `%` described in the preceding sections has the following exceptions:

1. In quoted strings in R-like code, braces are not counted and should not be escaped. For example, `\code{ "{" }` is legal.
2. In the first argument to `\eqn` or `\deqn` (whether in one- or two-argument form), no escapes or Rd comments are processed. Braces are counted unless escaped, so must be paired or escaped. This argument is passed verbatim to LaTeX for processing, including all escapes. Thus

```
\deqn{ f(x) = \left\{
                \begin{array}{ll}
                  0 & x < 0 \\
                  1 & x \ge 0
                \end{array}
              \right. }{ (non-Latex version) }
```

can be used to display the equation

$$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

when rendered in LaTeX, and a non-LaTeX version in other formats.

3 Changes from R 2.8.x Rd format

The following list describes syntax that was accepted in R 2.8.x but which is not accepted by the `parse_Rd()` parser.

1. The `\code{}` macro was used for general verbatim text, similar to `\samp{}`; now `\verb{}` (or `\kbd`, or `\preformatted`) must be used when the text is not R-like. This mainly affects text containing the quote characters `"`, `'` or `` ` ``, as these will be taken to start quoted strings in R code. Escape sequences (e.g. `\code{\a}`) should now be written as `\verb{\a}`, as otherwise `\a` would be taken to be a markup macro. Descriptions of languages other than R (e.g. examples of regular expressions) are often not syntactically valid R and may not be parsed properly in `\code{}`. Note that currently `\verb{}` is only supported in Rdoc version 2.

2. Treating `#ifdef ... #endif` and `#ifndef ... #endif` as markup macros means that they must be wholly nested within other macros. For example, the construction

   ```
   \title{
   #ifdef unix
   Unix title}
   #endif
   #ifdef windows
   Windows title}
   #endif
   ```

   needs to be rewritten as

   ```
   \title{
   #ifdef unix
   Unix title
   #endif
   #ifdef windows
   Windows title
   #endif
   }
   ```

3. R strings must be completely nested within markup macros. For example, `\code{"my string}"` will now be taken to be an unterminated `\code` macro, since the closing brace is within the string.

4. Macros should be followed by a non-alphanumeric character, not just a numeric one. For example, `1\dots10` now should be coded as `1\dots{}10`. (In this case, it will be parsed properly even though `\dots10` is not a legal macro, because the parser will attempt to drop the digits and reinterpret it as `\dots` followed by the text `10`. However, `1a\dots10b` will be parsed as text `1a` followed by the unknown macro `\dots10b`.) There is an internal limit (currently about 25) on the number of digits that can be removed before the pushback buffer in the parser overflows.

5. In processing in earlier versions, braces in R strings could be escaped, or left unescaped if they balanced within the section. Now if they are escaped within a string the escape character will be treated as part of the string, and since `"\{"` is not legal R code, this can lead to problems. In order to create files compatible with both the new parser and the older system, braces should not be quoted within strings, and they should be balanced, perhaps by adding a matching brace in a comment. For example,

   ```
   \code{"\{"}
   ```

   could now be entered as

   ```
   \code{"{" # not "}"}
   ```

   The matching brace is not needed if the new parser is the only target.

4 The parsing function

The `parse_Rd()` function takes a text connection and produces a parsed version of it. The arguments to the `parse_Rd()` function are described in its man page. The general structure of the result is a list of section markup, with one entry per Rd section. Each section consists of a list of text and markup macros, with the text stored as one-element character vectors and the markup macros as lists.

Single argument macros store the elements of the argument directly, with each element tagged as described below. Double argument macros are stored as two element lists; the first element is the first argument, the second element is the second argument. Neither one is tagged. The macros with either one or two arguments (currently `\eqn` and `\deqn`) store the two element form like other double argument

macros, and store the one element form in an analogous one element list.

The attributes of each element give information about the type of element. The following attributes are used:

class The object returned by `parse_Rd()` is of class "Rd".

`Rd_tag` Each object in the list generated from markup macros has a tag corresponding to its type. If the item is a markup macro, the tag is the macro (e.g. `\code` or `#ifdef`). Zero argument markup which is followed by `{}` will include the braces as part of the tag.

Non-macro tags include `TEXT`, `RCODE`, `VERB`, `COMMENT`, `UNKNOWN` (an unrecognized macro), and `LIST` (in Latex-like mode, a group of tokens in braces).

`Rd_option` Markup lists which had an optional parameter will have it stored in the `Rd_option` attribute.

srcref and srcfile Objects include source references.

4.1 Example

The following example looks at the `parse_Rd()` man page. The `tools:::RdTags` function extracts the tags from an "Rd" object. See the `Rd2HTML` function in the `tools` package for an extended example of working with the object.

```
> library(tools)
> infile <- file.path(R.home(), "src/library/tools/man/parse_Rd.Rd")
> Rd <- parse_Rd(infile)
> print(tags <- tools:::RdTags(Rd))

 [1] "COMMENT"       "COMMENT"       "COMMENT"
 [4] "COMMENT"       "TEXT"          "\\name"
 [7] "TEXT"          "\\alias"       "TEXT"
[10] "\\title"       "TEXT"          "\\description"
[13] "TEXT"          "\\usage"       "TEXT"
[16] "\\arguments"   "TEXT"          "\\details"
[19] "TEXT"          "\\value"       "TEXT"
[22] "\\section"     "TEXT"          "\\references"
[25] "TEXT"          "\\author"      "TEXT"
[28] "\\seealso"     "TEXT"          "\\examples"
[31] "TEXT"          "\\keyword"     "TEXT"
[34] "\\keyword"     "TEXT"
```

```
> Rd[[1]]

[1] "% File src/library/tools/man/parse_Rd.Rd\n"
attr(,"Rd_tag")
[1] "COMMENT"

> Rd[[which(tags == "\\title")]]

[[1]]
[1] " Parse an Rd file "
attr(,"Rd_tag")
[1] "TEXT"

attr(,"Rd_tag")
[1] "\\title"

> tools:::RdTags(Rd[[which(tags == "\\value")]])

 [1] "TEXT"   "TEXT"   "\\code" "TEXT"   "TEXT"   "TEXT"
 [7] "TEXT"   "TEXT"   "\\code" "TEXT"   "TEXT"   "TEXT"

> Rd <- parse_Rd(infile, verbose = TRUE)

1:1: COMMENT: % File src/library/tools/man/parse_Rd.Rd

2:1: COMMENT: % Part of the R package, http://www.R-project.org

3:1: COMMENT: % Copyright 2008 R Core Development Team

4:1: COMMENT: % Distributed under GPL 2 or later

5:1: TEXT:

6:1: VSECTIONHEADER: \name
6:6: '{'
6:7: VERB: parse_Rd
6:15: '}'
6:16: TEXT:

7:1: VSECTIONHEADER: \alias
7:7: '{'
7:8: VERB: parse_Rd
7:16: '}'
```

```
7:17: TEXT:

8:1: SECTIONHEADER: \title
8:7: '{'
8:8: TEXT:  Parse an Rd file
8:26: '}'
8:27: TEXT:

9:1: SECTIONHEADER: \description
9:13: '{'
9:14: TEXT:

10:1: TEXT:   This function reads an R documentation (Rd) file and parses it, for

11:1: TEXT:   processing by other functions.

12:1: TEXT:

13:1: TEXT:   It is
13:9: LATEXMACRO: \bold
13:14: '{'
13:15: TEXT: experimental
13:27: '}'
13:28: TEXT: .

14:1: '}'
14:2: TEXT:

15:1: RSECTIONHEADER: \usage
15:7: '{'
15:8: RCODE:

16:1: RCODE: parse_Rd(file, srcfile = NULL, encoding = "unknown", verbose = FALSE)

17:1: '}'
17:2: TEXT:

18:1: LISTSECTION: \arguments
18:11: '{'
18:12: TEXT:
```

```
19:1: TEXT:
19:3: LATEXMACRO2: \item
19:8: '{'
19:9: TEXT: file
19:13: '}'
19:14: '{'
19:15: TEXT: A filename or text-mode connection.  At present filenames

20:1: TEXT:     work best.
20:15: '}'
20:16: TEXT:

21:1: TEXT:
21:3: LATEXMACRO2: \item
21:8: '{'
21:9: TEXT: srcfile
21:16: '}'
21:17: '{'
21:18: RCODEMACRO: \code
21:23: '{'
21:24: RCODE: NULL
21:28: '}'
21:29: TEXT: , or a
21:36: RCODEMACRO: \code
21:41: '{'
21:42: RCODE: "srcfile"
21:51: '}'
21:52: TEXT:  object.  See the

22:1: TEXT:
22:5: LATEXMACRO: \sQuote
22:12: '{'
22:13: TEXT: Details
22:20: '}'
22:21: TEXT:  section.
22:30: '}'
22:31: TEXT:

23:1: TEXT:
```

```
23:3: LATEXMACRO2: \item
23:8: '{'
23:9: TEXT: encoding
23:17: '}'
23:18: '{'
23:19: TEXT: Encoding to be assumed for input strings.
23:60: '}'
23:61: TEXT:

24:1: TEXT:
24:3: LATEXMACRO2: \item
24:8: '{'
24:9: TEXT: verbose
24:16: '}'
24:17: '{'
24:18: TEXT: Logical indicating whether detailed parsing

25:1: TEXT:     information should be printed.
25:35: '}'
25:36: TEXT:

26:1: '}'
26:2: TEXT:

27:1: SECTIONHEADER: \details
27:9: '{'
27:10: TEXT:

28:1: TEXT:   This experimental function parses Rd files according to the

29:1: TEXT:   specification given in

30:1: TEXT:
30:3: VERBMACRO: \url
30:7: '{'
30:8: VERB: http://developer.r-project.org/parseRd.pdf
30:50: '}'
30:51: TEXT: .  At the current

31:1: TEXT:   writing, this is not identical to the rules used by other tools:  it
```

```
32:1: TEXT:   is somewhat stricter.

33:1: '}'
33:2: TEXT:

34:1: LISTSECTION: \value
34:7: '{'
34:8: TEXT:

35:1: TEXT:   An object of class
35:22: RCODEMACRO: \code
35:27: '{'
35:28: RCODE: "Rd"
35:32: '}'
35:33: TEXT: .  The internal format of this object is

36:1: TEXT:   subject to change.

37:1: TEXT:

38:1: TEXT:   At present files which are marked as Latin-1 or UTF-8 or where

39:1: TEXT:
39:3: RCODEMACRO: \code
39:8: '{'
39:9: RCODE: encoding
39:17: '}'
39:18: TEXT:  has one of those values result in text marked in the

40:1: TEXT:   encoding. In a non-UTF-8 MBCS locale the result will be marked as

41:1: TEXT:   UTF-8.  In all other cases files are assumed to be in the native enco

42:1: '}'
42:2: TEXT:

43:1: SECTIONHEADER2: \section
43:9: '{'
43:10: TEXT: Warning
```

```
43:17: '}'
43:18: '{'
43:19: TEXT:

44:1: TEXT:   These functions are still experimental.  Names, interfaces and values

45:1: TEXT:   might change in future versions.

46:1: '}'
46:2: TEXT:

47:1: SECTIONHEADER: \references
47:12: '{'
47:13: TEXT:
47:14: VERBMACRO: \url
47:18: '{'
47:19: VERB: http://developer.r-project.org/parseRd.pdf
47:61: '}'
47:62: TEXT:
47:63: '}'
47:64: TEXT:

48:1: SECTIONHEADER: \author
48:8: '{'
48:9: TEXT:  Duncan Murdoch
48:25: '}'
48:26: TEXT:

49:1: SECTIONHEADER: \seealso
49:9: '{'
49:10: TEXT:

50:1: TEXT:
50:3: RCODEMACRO: \code
50:8: '{'
50:9: OPTMACRO: \link
50:14: '{'
50:15: TEXT: Rd2HTML
50:22: '}'
50:23: '}'
```

```
50:24: TEXT: .

51:1: TEXT:

52:1: TEXT:
52:3: RCODEMACRO: \code
52:8: '{'
52:9: OPTMACRO: \link
52:14: '{'
52:15: TEXT: Rd_parse
52:23: '}'
52:24: '}'
52:25: TEXT: , the function used for parsing

53:1: TEXT:   Rd files by
53:15: VERBMACRO: \command
53:23: '{'
53:24: VERB: R CMD check
53:35: '}'
53:36: TEXT: .

54:1: '}'
54:2: TEXT:

55:1: RSECTIONHEADER: \examples
55:10: '{'
55:11: RCODE:

56:1: RCODE: baseRd <- Rd_db("base")

57:1: RCODE: con <- textConnection(baseRd[[1]], "rt")

58:1: RCODE: parse_Rd(con)

59:1: RCODE: close(con)

60:1: '}'
60:2: TEXT:

61:1: SECTIONHEADER: \keyword
```

```
61:9: '{'
61:10: TEXT: utilities
61:19: '}'
61:20: TEXT:

62:1: SECTIONHEADER: \keyword
62:9: '{'
62:10: TEXT: documentation
62:23: '}'
62:24: TEXT:

63:1: END_OF_INPUT
```

5 The Sexpr macro in R 2.10.0

R 2.10.0 will contain the new macros `\Sexpr` and `\RdOpts`. These are modelled after Sweave, and are intended to control R expressions in the Rd file. To the parser, the `\Sexpr` macro is simply a macro taking an optional verbatim argument in square brackets, and a required R-like argument in the braces. For example, `\Sexpr{ x <- 1 }` or `\Sexpr[stage=build]{ loadDatabase() }`. The `\RdOpts` macro takes a single verbatim argument, intended to set defaults for the options in `\Sexpr`.

These two macros are modelled after Sweave, but the syntax and options are not identical. (We will expand on the differences below.)

The R-like argument to `\Sexpr` must be valid R code that can be executed; it cannot contain any expandable macros other than `#ifdef/#ifndef/#endif`. Depending on the options, the code may be executed at package build time, package install time, or man page rendering time. Since package tarballs are built with the conditionals included, `#ifdef/#ifndef/#endif` blocks cannot be included in code designed to be executed at build time. Rd comments using `%` are ignored during execution.

The options follow the same format as in Sweave, but different options are supported. Currently the allowed options and their defauls are:

eval=TRUE Whether the R code should be evaluated.

echo=FALSE Whether the R code should be echoed. If TRUE, a display will be given in a preformatted block. For example,

`\Sexpr[echo=TRUE]{ x <- 1 }` will be displayed as

```
> x <- 1
```

keep.source=TRUE Whether to keep the author's formatting when displaying the code, or throw it away and use a deparsed version.

results=text How should the results be displayed? The possibilities are

text Apply `as.character()` to the result of the code, and insert it as a text element.

verbatim Print the results of the code just as if it was executed at the console, and include the printed results verbatim. (Invisible results will not print.)

rd The result is assumed to be a character vector containing markup to be passed to `parse_Rd(fragment=TRUE)`, with the result inserted in place. This could be used to insert computed aliases, for instance.

hide Insert no output.

strip.white=TRUE Remove leading and trailing white space from each line of output if `strip.white=TRUE`. With `strip.white=all`, also remove blank lines.

stage=install Control when this macro is run. Possible values are

build The macro is run when building a source tarball.

install The macro is run when installing from source.

render The macro is run when displaying the help page.

The `#ifdef` conditionals are applied after the `build` macros but before the `install` macros. In some situations (e.g. installing directly from a source directory without a tarball, or building a binary package) the above descriptions may not be accurate, but authors should be able to rely on the sequence being `build`, `#ifdef`, `install`, `render`, with all stages executed.

Code is only run once in each stage, so a `\Sexpr[results=rd]` macro can output an `\Sexpr` macro designed for a later stage, but not for the current one or any earlier stage.

width, height, fig These options are currently allowed but ignored. Eventually the intention is that they will allow insertion of graphics into the man page.

5.1 Differences from Sweave

As of the current writing, these are the important differences from Sweave:

1. Our `\Sexpr` macro takes options, and can give full displayed output. In Sweave the Noweb syntax or other macros are needed to process options.
2. The Sweave `\Sexpr` macro is roughly equivalent to our default `\Sexpr[eval=TRUE,echo=FALSE,results=text]` but we will insert whatever `as.character()` gives for the whole result, while Sweave only inserts the first element of a vector result.
3. We use `\RdOpts` rather than `\SweaveOpts` here, to emphasize that we are not running Sweave, just something modelled on it.
4. We run `\Sexpr` macros in the three stage system, whereas Sweave does calculations in one pass.