Parsing Rd files

Duncan Murdoch

October 21, 2008

Abstract

R-devel (to become R 2.9.x) will soon have a parser for Rd format help files. This will allow easier processing, easier syntax checking, and eventually, easier conversion to other formats.

To write this parser, it has been 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).

This document describes the new format, necessary changes to existing Rd files, and the structure returned from the new `parse_Rd()` function.

1 Introduction

The R documentation (Rd) file format has not had a formal description. It has grown over time, with pieces added, and R and Perl code written to recognize 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 2.8.0. Finally, the last section describes the `parse_Rd()` function and its output. This document does not describe the interpretation of any of the markup in Rd files.

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 Lexical structure

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

End of line (newline) characters mark the end of pieces of text. The end of a file does the same. These are not returned by the parser.

The brackets `[` and `]` have special meaning in certain contexts; see section 2.2.

Whitespace (blanks, tabs and empty lines) is ignored at the start of text in LaTeX-like sections, and at the top level.

2.1.1 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 a syntax error is signalled.

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

2.1.2 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 comments.

2.1.3 Braces

The braces `{` and `}` are used to mark the arguments to markup macros, and may also appear within R-like or verbatim text, provided they balance. The parser keeps a count of brace depth, and the return to zero depth signals the end of an argument. Opening and closing braces on arguments are not returned by the parser, but balanced braces within text are.

The one exception to special handling of braces is within quoted strings in R-like code, where braces need not be escaped. That is, `\code{ "{" }` is legal.

2.2 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 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`. For example,

```
#ifdef unix
Some text
#endif
```

is processed with first argument `unix` 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.

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 `\preformatted{}` macros is verbatim.

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

2.3 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 not considered relevant; currently the parser removes leading whitespace but may keep following whitespace. This is subject to change.

Braces within LaTeX-like text must be escaped as `\{` and `\}` except when used to delimit arguments to markup macros.

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

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

2.4 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 LaTeX-like comments, the Rd parser will return R comments as part of the text of the code; the LaTeX-like comment will be returned marked as a comment.

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
`\synopsis`	1	yes	no	Verbatim
`\arguments`	1	yes	`\item{}{}`	Latex-like
`\value`	1	yes	`\item{}{}`	Latex-like

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
\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
\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
`\dontrun`	1	no	no	R-like
`\dontshow`	1	no	no	R-like
`\donttest`	1	no	no	R-like
`\testonly`	1	no	no	R-like
`\command`	1	no	no	Verbatim
`\env`	1	no	no	Verbatim
`\kbd`	1	no	no	Verbatim
`\option`	1	no	no	Verbatim
`\preformatted`	1	no	no	Verbatim
`\samp`	1	no	no	Verbatim
`\special`	1	no	no	Verbatim
`\url`	1	no	no	Verbatim
`\deqn`	1 or 2	no	no	Verbatim
`\eqn`	1 or 2	no	no	Verbatim

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 % characters must be escaped even within strings, or they will be taken as Rd comments.

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.

2.5 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.

No markup macros are recognized within verbatim text.

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. Within LaTeX-like text, balanced braces that are not attached to a macro were ignored; now they are not allowed.
2. The `\R` macro was sometimes used with a single empty argument; now it must be used with no argument.
3. The `\code{}` macro was used for general verbatim text, similar to `\preformatted{}`; now the latter must be used when the text is not R-like. This mainly affects descriptions of escape sequences (e.g. `\code{\a}` must now be written as `\preformatted{\a}`, as otherwise `\a` would be taken to be a markup macro), and descriptions of languages other than R (e.g. examples of regular expressions are often not syntactically valid R).
4. 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
}
```

5. 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.
6. Macros need to be followed by a non-alphanumeric character, not just a numeric one. For example, `1\dots10` now should be coded as `1\dots 10`.

4 The parsing function

The `parse_Rd()` function takes a text connection and produces a parsed version of it. 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 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 RdFile.

rdTag 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`). Non-macro tags include `TEXT`, `RCODE`, `VERB`, or `COMMENT`.

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

srcref and srcfile Objects may in the future have source references added; currently this is incomplete.