Parsing Rd files

Duncan Murdoch

December 24, 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). The new stricter format is being informally called Rdoc version 2.

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 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 2.8.0. 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 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; currently no attempt is made to deal with encodings.

2.2 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. 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.

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.

In quoted strings in R-like code, braces need not be escaped. For example, `\code{ "{" }` is legal.

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 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` (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.

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{}` macros is verbatim.

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

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` macro, where they are used to delimit the optional argument.

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

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
`\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

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 sequence `\l` in a string will be taken to start a markup macro. This is intended to allow `\link` to be used in a string (e.g. the common idiom `\code{"\link{someclass}"}`).

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.

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. The `\code{}` macro was used for general verbatim text, similar to `\samp{}`; now the latter (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 `\samp{\a}`, as otherwise `\a` would be taken to be a markup macro if recognized). 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{}`.
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.

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 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.