An introduction to Rich Text Format (RTF) and the `rtfSweave` package

Stephen Weigand `Weigand.Stephen@mayo.edu`

June 21, 2022

1 Overview

In this document I try to provide some very basic information about Rich Text Format (RTF) as a mark-up language and show how to use the `rtfSweave` package with `Sweave` to make RTF statistical reports. For me, the key to understanding RTF was reading the *RTF Pocket Guide* by Sean Burke[1]. It covers important aspects of RTF and only after reading it did I find that the RTF specifications published by Microsoft made some sense. The *Pocket Guide* can be purchased and downloaded as a PDF from the publisher's website (currently `http://shop.oreilly.com/product/9780596004750.do`).

Rich Text Format (RTF) is a mark-up language in that RTF files are plain text files containing document content such as text, tables, and figures along with commands indicating how these should be displayed. In this sense, RTF is somewhat similar to HTML or LaTeX. RTF was developed by Microsoft and is tightly integrated into Microsoft Word (MS Word), so much so that RTF is essentially a text-based interface to Word documents. If you are creating a statistical report for an MS Word user, there may be an advantage to writing the RTF directly rather than using a document conversion program. Still, this should not be taken as arguing that RTF is a replacement for LaTeX.

One important way that RTF differs in spirit from HTML or LaTeX is that in the RTF specification there is much less separation between content and presentation. For example, headings in an RTF document are really just short paragraphs with special formatting to make them display like a heading. That is, headings are not not structurally different from any other paragraph.

At this point, it may make sense to show an example of an RTF document. Here is the "Hello, world!" example from the *Pocket Guide*. Note that RTF has no built-in comments so I am using "//" to indicate "pseudo-comments" which annotate the file.

Blank lines are ignored in RTF but spaces in the RTF document will show up in the rendered document.

2 Document prolog

The *Pocket Guide* has detailed information on the prolog. In it you define some document defaults, a font table, a color table (optional), and an info group containing metadata

```
{\rtf1                // RTF version 1, your only choice
\ansi                 // Document is in the ANSI character set

\deff0                // Default font is font 0, defined below
\deflang1033          // Default language code 1033 = en-us
\plain \fs24          // Plain format has fontsize 24/2 = 12 points

{\fonttbl             // Font table
{\f0 Times New Roman;} // Font 0 is Times New Roman (roman)
{\f1 Arial;}          // Font 1 is Arial (sans)
{\f2 Courier New;}    // Font 2 is Courier New (monospace)
}

{\pard \fs28 \b       // Paragraph set at 28/2 = 14 pts in bold
Greeting              //   to resemble a level-1 heading
\par}                 // End paragraph

{\pard                // New paragraph set with "plain" formatting
Hello, World!
\par}                 // End paragraph
}                     // End of the RTF document
```

Figure 1: An annotated RTF version of “Hello, World!”

(optional). The font table is important for `rtfSweave` because by default code chunks use `\f2` and so this would typically be a monospaced font.

3 Paragraphs and paragraphs formatted to look like headings

As I mentioned above, RTF has a paragraph entity, but no heading entities. This means that headings are just paragraphs with special formatting such as boldface type. Following the *P*ocket Guide[1], paragraphs are written using the following form.

```
{\pard
A short paragraph.
\par}
```

Formatting instructions for the paragraph can follow the `\pard` command on the same line. For many size-related commands, the units are in "twips" where one twip equals one twentieth of a point. (If 20 twips equals 1 point, and 72 points/inch, then there are $72 \times 20 = 1440$ twips per inch.[1]) One exception is that text sizes are specified in half points. To illustrate, the following paragraph has 120 twips (or $120/20 = 6$ points) of space before and after the paragraph and is set in font zero at 24 half-points (12 points). A single space after the RTF commands is optional (and will not show up in the document) but helps readability.

```
{\pard \sb120 \sa120 \f0 \fs24
A short paragraph set in font zero at 12 points with 6 points spacing before and after.
\par}
```

If the paragraph you are writing is more than one line, a helpful way to handle spaces is to indent everything after the first line with a space. That way, it will be easy to see that there are no inadvertent spaces at the end of a line. This is an example.

```
{\pard \sb120 \sa120 \f0 \fs24
A short paragraph that
 is written in multiple
 lines.
\par}
```

A heading paragraph is similar but with additional formatting instructions. The following creates the appearance of a heading by being displayed in font 1 (`\f1`), at 16 points (`\fs32`) and in bold (`\b`). The command `\keepn` keeps the headling-like paragraph on the same page as the paragraph that follows it. The command `\outlinelevel1` is one of the few structural mark-up commands in RTF and facilitates document navigation.[2]

```
{\pard \sb120 \sa120 \f1 \fs32 \b \keepn \outlinelevel1
Introduction
\par}
```

RTF mark-up *is* readable but verbose. One simple solution to the verbosity is to create macro-like substitutions. For example, one could define a macro-like command of `\paragraph` and then write paragraphs such as this:

[1]My guess is that twips were used to allow RTF commands to be specified in integer units.

[2]MS Word has a "navigation pane" and paragraphs with `\outlinelevel`*N* show up in this pane.

```
{\paragraph
A short paragraph.
\par}
```

The file could then be“preprocessed” using text substitutions.

For example, here is a simple function to read an RTF file, replace “`\paragraph`” with valid RTF mark-up, and write out the result.

```
> preprocess <- function(original.rtf, new.rtf){
+   doc <- readLines(original.rtf)
+   doc <-
+     sub("^{\\paragraph$",
+         "{\\pard \\sb120 \\sa120 \\f0 \\fs24",
+         doc)
+   writeLines(doc, new.rtf)
+ }
```

With these kind of substitutions specifying paragraphs, sections, subsections, etc, the document can be made more readable and much easier to write. One limitation of these simple text substitutions is that the ending `\par` command and closing curly bracket is still needed.

Unlike LaTeX, RTF paragraphs need some markup. For example, in LaTeXthe following text would constitute three paragraphs:

```
This is paragraph 1.

This is paragraph 2.

This is paragraph 3.
```

However, in RTF that the text would consitute a single (possibly partial) paragraph that displayed like this: “This is paragraph 1.This is paragraph 2.This is paragraph 3.”

RTF does have a “short cut” form to a paragraph which is just text ending in a backslash. Here is an example which is analogous to typing a paragraph in MS Word, hitting the return key twice, and typing another paragraph.

```
A first
 paragraph.\
\
A second paragraph.\
```

4 Figures

PNG, JPEG, or Windows enhanced metafile (EMF) images can be included in an RTF document by converting the image file to text using a hexdump and wrapping it in some RTF control commands. (See `http://en.wikipedia.org/wiki/Hex_dump` for details on a hexdump.)

Here is a partial example of RTF that would display a PNG file in a paragraph by itself. The key commands are texttt\pict and texttt\pngblip.

```
{\pard{\pict\pngblip
8950 4e47 0d0a 1a0a 0000 000d 4948 4452
0000 0064 0000 0064 0803 0000 0047 3c65
6600 0000 0650 4c54 4500 0000 ffff ffa5
...
5771 c762 1cd6 f1ed 15bf 211b b221 2b21
3f2f a623 9b76 0cff 8d00 0000 0049 454e
44ae 4260 8200}\par}
```

This type of hexdump can be generated as follows. (Note the artificially low resolution of the PNG figure in this example.)

```
> tmp <- tempfile()
> png(tmp, height = 2, width = 2, units = "in", res = 10)
> plot(1:10)
> dev.off()
```

```
null device
          1
```

```
> size <- file.info(tmp)$size
> hex <- readBin(tmp, what = "raw", size)
> cat(hex, fill = 60, sep = " ")
```

```
89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00 00 00 14
00 00 00 14 08 03 00 00 00 ba 57 ed 3f 00 00 00 5a 50 4c 54
45 84 84 84 92 92 92 94 94 94 96 96 96 97 97 97 9b 9b 9b 9d
9d 9d 9e 9e 9e a9 a9 a9 ab ab ab ac ac ac ae ae ae b1 b1 b1
b2 b2 b2 b4 b4 b4 b7 b7 b7 b8 b8 b8 ba ba ba bc bc bc bf bf
bf c2 c2 c2 c8 c8 c8 ce ce ce d8 d8 d8 dc dc dc e4 e4 e4 ea
ea ea f3 f3 f3 fe fe fe ff ff ff e4 a4 87 cd 00 00 00 09 70
48 59 73 00 00 01 89 00 00 01 89 01 9e 2e 11 35 00 00 00 5c
49 44 41 54 18 95 63 90 c5 02 18 06 a1 20 8f 00 98 e6 e3 11
17 e5 e7 e6 e4 e4 02 0b 4a 0b cb ca f2 8b c8 4a b2 31 33 b2
b0 72 08 09 41 04 c5 a0 7a a4 a4 80 48 46 0a d5 4c 7e 59 59
41 59 59 69 1e 14 41 11 59 59 26 59 59 31 01 72 dd 29 2e 81
45 90 81 9d 97 8f 64 6f 62 13 04 00 6f 7a 2a ff cd 51 22 34
00 00 00 00 49 45 4e 44 ae 42 60 82
```

The good news is that `rtfSweave` handles this for the user. Like in Sweave, code chunks just need `fig = TRUE` (or `fig = true`). There is also an option invoked with `hex = FALSE` which will skip the hex dump and instead use an `INCLUDEPICTURE` RTF field instruction which is an analog to a LaTeX`\includegraphics` command. This makes for a smaller RTF file (since the figures are not part of the file) but special steps need to be taken to embed the figures in the document. I recommend staying with `hex = FALSE` since the `INCLUDEPICTURE` approach is not very robust and does not make it easy to honor figure sizes.

5 Tables

Tables in RTF are defined by a sequence of independent table rows defined by the `{\trowd ... \row}` construct. Each row consists of some number of "cells." Here is a simple two-row table where each row has three cells.

```
{\trowd
\cellx1440\cellx2880\cellx4320
\pard\intbl Aligator \cell
\pard\intbl Bear \cell
\pard\intbl Cat \cell
\row}

{\trowd
\cellx1440\cellx2880\cellx4320
\pard\intbl Dog \cell
\pard\intbl Elephant \cell
\pard\intbl Fox \cell
\row}
```

The `\cellx` control word specifies where the right margin of the cell is placed relative to the page margin. In both rows above, the first cell's right edge is at 1440 twips (1 inch) from the page margin, the second cell's right edge is at 2880 twips (2 inches), and the third cell's right margin is at 4320 twips (3 inches).

6 Using rtfSweave

At present, the `rtfSweave` simply provides a driver so that `Sweave` can process a file with R code chunks and RTF documentation chunks. Assuming that the file with text and code is `myreport.Rrtf`, the commands to process the file are

```
library("rtfSweave")
Sweave("myreport.Rrtf",
       driver = RweaveRtf(),    ## Important
       syntax = SweaveSyntaxRtf,
       png = FALSE, wmf = TRUE) ## options
```

Actually, the syntax argument is not necessary because Sweave is smart enough to find the right syntax object, SweaveSyntaxRtf.

This syntax object closely follows the SweaveSyntaxNoweb object and therefore, code chunks are defined by

```
<<optional label, various options>>=
Your code here
@
```

To include R code in text chunks, use {\Sexpr <your code>}.

As with Sweave, options can be set within the document. An example is below.

```
{\SweaveOpts echo = false, resolution = 100}
```

Three options apparently cannot be set this way: rtf.Schunk, rtf.Sinput, and rtf.Soutput. These options take a string of RTF commands and the backslashes cause problems. It seems that these options have to be passed as arguments to Sweave which means they can't be changed within the document.

References

[1] Sean M Burke, *RTF Pocket Guide.* O'Reilly, 2003.