The Stata Journal (2018)
18, Number 1, pp. 197–205

Some commands to help produce Rich Text Files from Stata

Matthew S. Gillman
King's College London
Faculty of Life Sciences & Medicine
School of Cancer and Pharmaceutical Sciences
London, UK
matthew.gillman@kcl.ac.uk

Abstract. Producing Rich Text Format (RTF) files from Stata can be difficult and somewhat cryptic. In this article, I introduce commands to simplify this process; one builds a table row by row, another inserts a PNG image file into an RTF document, and the others start and finish the RTF document.

Keywords: pr0068, initiatertf, insertimagertf, addrowrtf, endrtf, Rich Text Format, RTF, automation, PNG, tables, fonts

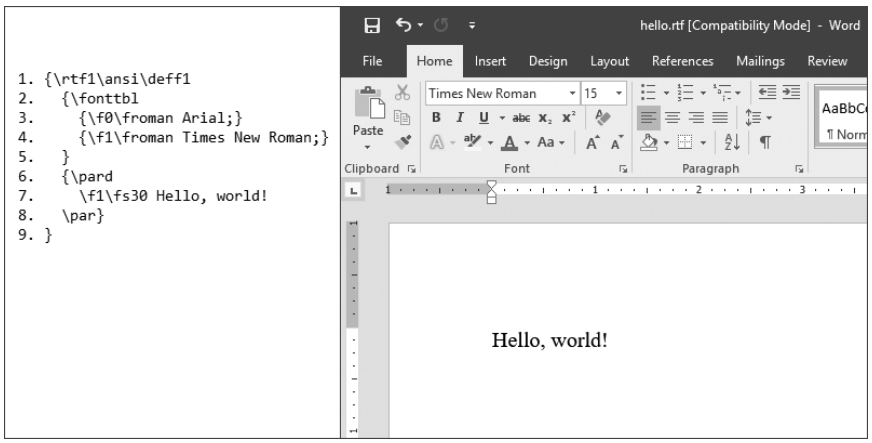
1 Introduction

Stata users sometimes wish to generate documents in Rich Text Format (RTF). Such documents are not tied to one particular operating system but instead are platform independent. They can be opened (and edited) with word-processing software such as Microsoft Word and Libre Office.

One paradigm in which RTF documents may be required is if a statistician has an analysis that they repeat many times with varying parameters, data, or both. It would be tedious (not to say error-prone) if, each time the analysis was rerun, the user had to copy the results from Stata and paste them into the document he or she was working on. It would be far more convenient (and reproducible) if the statistician used a do-file that consistently performed the same analysis and wrote the results to the document automatically.

The RTF specification was developed by Microsoft, and the final version (1.9.1) is available for download (Microsoft Corporation 2008). A key resource is the useful book written by Sean M. Burke (2003), which summarizes the main features of the specification.

Writing RTF documents is somewhat cryptic. A `Hello world` example might be the following (excluding the line numbers on the left-hand side):



If the content (excluding line numbers) in the left side of the preceding table is saved in a text file with the name `hello.rtf`, it should be viewable in standard word-processing packages (for example, Microsoft Word, Libre Office, etc.) as shown in the right side of the table. Note that such a package will display the contents in RTF format; if one wishes to view and debug code like that in the left side of the table, one must use an ASCII editor such as Notepad or vi. Here the first line opens the document and sets the default font to `f1` in the (forthcoming) font table. The second line declares the font table. Lines 3 and 4 define the fonts to be used for fonts `f0` and `f1` (there can be more than this). In this specific example, line 3 is superfluous because font `f0` is not used in the document. The closing brace on line 5 ends the font table. The `{\pard` command on line 6 opens a new paragraph. Line 7 contains the text for this paragraph, using font `f1` from the table (`f0` could be used instead, if desired) and size 15 pt. (Note that RTF requires double this number to be specified as the `fs` [font size] value, in this case 15*2=30). The `\par}` on line 8 closes the paragraph. Finally, the closing brace on line 9 completes the RTF document.

To insert a table, things get more complicated, especially if one wishes to have visible borders around the table. I wrote an ado-file to simplify this process (see section 2).

One can also add PNG images into an RTF document; I wrote a command to facilitate this (see section 3). Hopefully, these commands will add to those already produced by Roger Newson (2009), author of `rtfutil`, and others.

Of course, we assume that the RTF document is produced by Stata. Hence, the lines above might be placed in a do-file as `file write` statements:

```
// This is helloworldtest.do
global RTFout = "hello.rtf"
file open handle1 using "$RTFout", write replace
file write handle1 "{\rtf1\ansi\deff1" _n
file write handle1 "{\fonttbl" _n
file write handle1 "{\f0\froman Arial;}" _n
file write handle1 "{\f1\froman Times New Roman;}" _n
file write handle1 "}" _n
file write handle1 "{\pard" _n
file write handle1 "\f1\fs30 Hello, world!" _n
file write handle1 "\par}" _n
file write handle1 "}"
file close handle1
```

Newlines (`_n`) are not strictly necessary, but they aid in the debugging of RTF file syntax.

It is tedious to set up the font table each time one wishes to create an RTF document. Accordingly, I wrote the `initiatertf` command and wrote the `endrtf` command to finalize the document. When we use these helper commands, the above becomes

```
global RTFout = "hello.rtf"
file open handle1 using "$RTFout", write replace
initiatertf handle1

file write handle1 "{\pard" _n
file write handle1 "\f1\fs30 Hello, world!" _n
file write handle1 "\par}" _n

endrtf handle1
file close handle1
```

`initiatertf` has options to specify the fonts used in the document's font table, to set the size of the margins (in mm), and to set the size of the width of the entire page (again in mm).

2 Adding rows to a table

To write a simple 2×2 table (without borders) to an RTF file, one might include

```
// Row 1
file write handle1 "{" _n "\trowd \trgaph180" _n "\cellx1440\cellx2880" _n
file write handle1 "\pard\intbl {Add.}\cell" _n "\pard\intbl {some.}\cell"
file write handle1 "\row" _n "}"

// Row 2
file write handle1 "{" _n "\trowd \trgaph180" _n "\cellx1440\cellx2880" _n
file write handle1 "\pard\intbl {data.}\cell" _n "\pard\intbl {here.}\cell" _n
file write handle1 "\row" _n "}"
```

Here is the above code fragment in context, using the helper commands already introduced:

```
// This is twobytwo.do
global RTFout = "myfirsttable.rtf"
file open handle1 using "$RTFout", write replace
initiatertf handle1

// Add a paragraph of text:
file write handle1 "{\pard" _n
file write handle1 "\f1\fs30 Here is a table:\line" _n
file write handle1 "\par}" _n

// Opening brace to enclose the table (for safety)
file write handle1 "{" _n

// Row 1
file write handle1 "{" _n "\trowd \trgaph180" _n "\cellx1440\cellx2880" _n
file write handle1 "\pard\intbl {Add.}\cell" _n "\pard\intbl {some.}\cell"
file write handle1 "\row" _n "}"

// Row 2
file write handle1 "{" _n "\trowd \trgaph180" _n "\cellx1440\cellx2880" _n
file write handle1 "\pard\intbl {data.}\cell" _n "\pard\intbl {here.}\cell" _n
file write handle1 "\row" _n "}"

// Closing brace to enclose the table (for safety)
file write handle1 "}" _n

endrtf handle1  // End of document
file close handle1
```

Note that RTF does not require a "table object" to be declared but builds up a table one row at a time. The Stata code above would result in content looking like the following:

Here is a table:

Add.	some.
data.	here.

Note that although the Stata newline character (`_n`) can be used to generate a new line in the RTF file generated by Stata, adding a blank line within the RTF document as it is displayed by a word processor requires use of `\line` (see bold above). These can be chained together: `\line\line\line`.

Readers will note the increasingly cryptic nature of the RTF text. In this example based on one in Burke (2003), the internal margins of the cells are 180 twips.[1] That is what the `\trgaph180` sets up. Starting from the left, the first cell ends at 1,440 twips (1 inch) from the margin and the second at 2,880 twips (2 inches). Each row is declared to begin with `\trowd` and end with `\row`. This example follows the safety recommendation in Burke (2003) to wrap the text of each cell, each table row, and indeed the entire table in braces `{}`.

1. 1 twip = 1/20 of a typographical point = 1/1440 inch or $\sim$ 1/57 mm.

The situation is even more complicated when one wishes to add borders to a table. However, rather than showing the full details of this, I introduce `addrowrtf`, which is very simple to use; a table similar to the one above but with borders (the default) would be declared in the do-file as

```
file write handle1 "{" _n          // Opening brace to enclose table.
addrowrtf handle1 Add. some.       // Row 1
addrowrtf handle1 data. here.      // Row 2
file write handle1 "}" _n          // Closing brace to enclose table.
```

Notice that here the only parameters passed to `addrowrtf` are the file handle and the text to be entered in the cells. This example will produce a table looking something like this:

Add.	some.
data.	here.

The `addrowrtf` command has a number of options that can be used. Here no options were specified, so the default table font size of 10 points was used. Options are described in the help file and include the ability to specify the cell width, whether borders are drawn, the font size, the font number, and relative cell widths.

In fact, `addrowrtf` will dynamically calculate the number of columns based on the number of arguments supplied. There is no requirement for successive rows to have the same number of cells (columns) as the preceding rows. An example might be the following,

```
// Note the quotes to force "my data" to appear in one cell:
addrowrtf handle1 this is "my data" 26

// Need a space in quotes for a blank cell:
addrowrtf handle1 more data " "
addrowrtf handle1 a b c d e f g h i j k l m n o p q r s t u v w x y z
file write handle1 "}" _n
```

which produces this:

this	is	my data	26	
more	data			
a	b	c	d	e

Here the rows are positioned using the default offset and cell widths. Using these, the third row expands beyond the margin of the page. Options such as `tablewidth()` and `internal()` may be used to prevent this.

A suitable do-file might look like this:

```
// File diffrowlengths.do
program drop _all
global outputRTF = "Table with different row lengths.rtf"
file open handle1 using "$outputRTF", write replace
initiatertf handle1 , fonts ("Calibri, Edwardian Script ITC, Arial")

addrowrtf handle1 this is "my data" 26
addrowrtf handle1 more data " "
addrowrtf handle1 a b c d e f g h i j k l m n o p q r s t u v w x y z ,  ///
   tablewidth(100) internal(0.5)
file write handle1 "}" _n

endrtf handle1
file close _all
```

The appearance is thus changed:

this							is							my data							26				
more							data																		
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

For full details of `addrowrtf`'s options, refer to the help file.

3 Inserting a PNG image

One can dynamically create a graph in Stata, save it to a PNG file, and then insert the latter into an RTF document file. It is tricky to do this because Stata must open the PNG file, read a single byte at a time, and convert each byte into hexadecimal format before writing it to the RTF file. Readers will be pleased to know I wrote the `insertimagertf` command for just such an eventuality. Like `addrowrtf`, it uses a file handle. Nichols (2010) has written `png2rtf`, which is similar, although it requires a filename to be provided and has more options; `insertimagertf` is a simpler tool.

Note that the only two arguments supplied are the RTF file handle (as before) and the name of the PNG file written by Stata previously. A simple design philosophy was adopted because a user can produce the final formatted version in the word processor to his or her own preference.

This command has a sole option, `leftoffset()`, that specifies (in mm) the distance of the left-hand side of the image from the left-hand page margin. Negative values are allowed to position it to the left of this margin.

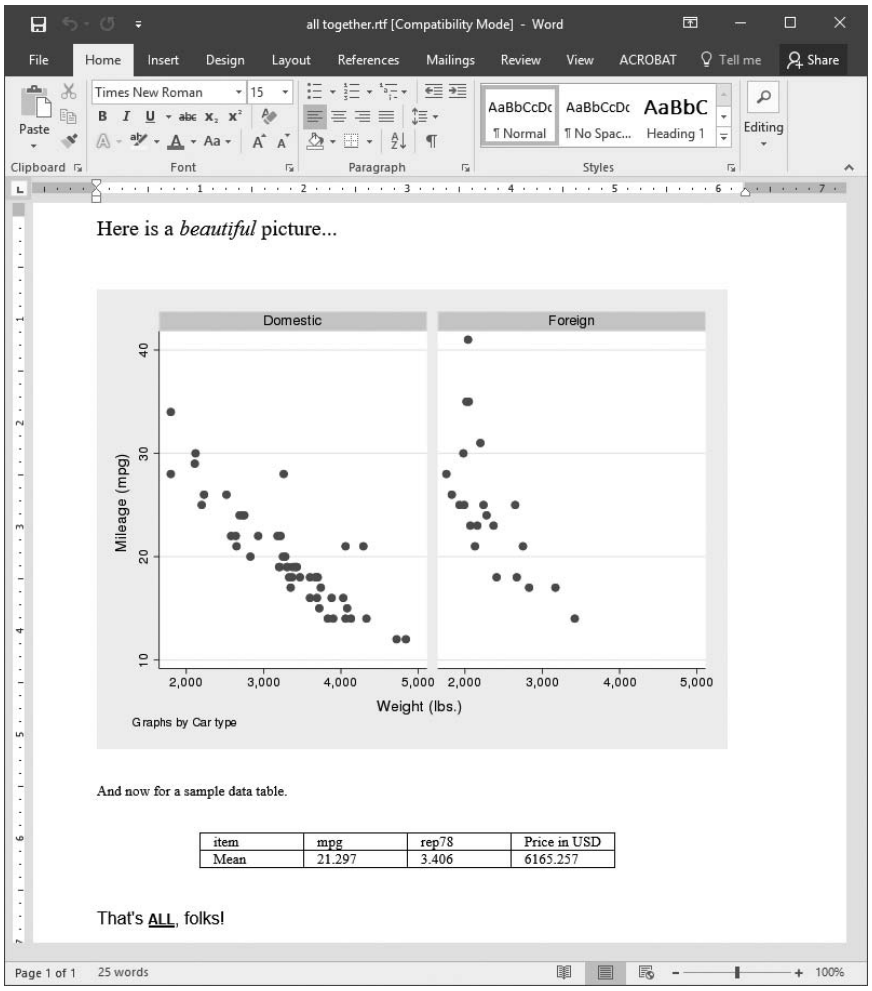
This command builds upon `rtfutil`'s `rtflink` program, except that `rtflink` inserts a link to an image, and `insertimagertf` inserts a copy of the actual image.

Full details are given in the `insertimagertf` help file. If the required image file is missing, this command will gracefully fail without corrupting the output RTF document.

4 Example: Putting it all together

Here is a complete example that brings together the thoughts in the previous sections.

The document shown below is produced by the Stata code following it.



Notice the table set up by referencing the auto mean results in the following code:

```
// This is alltogether.do
clear all
sysuse auto, clear
set graphics on
scatter mpg weight, by(foreign)

global pngfile = "myplot.png"
graph export "$pngfile", replace                    // Save graphics file to disk.

// Set up output file.
global outputRTF = "all together.rtf";              // The name of the new RTF file
file open handle1 using "$outputRTF", write replace

initiatertf handle1

file write handle1 "{\pard"                         // Start new paragraph
file write handle1 "\f1\fs30 Here is a {\i beautiful} picture... \line\line"
file write handle1 "\par}"                          // Close paragraph

insertimagertf handle1 "$pngfile" , leftoffset(0)  // Adds graph into document.
rm "$pngfile"                                       // Delete file (if desired).

//////////////////////////////////////////////////////

file write handle1 "{\pard\f1\fs20 \line\line And now for a sample data " ///
   "table. \line\line\par}"
file write handle1 "{" _n                           // Start of table
tabstat mpg rep78 price, save

// This is a semiautomated way of obtaining the values desired:
matrix mydata = r(StatTotal)
local cols = colsof(mydata)                         // Number of columns
forvalues colnum = 1/`cols´ {
    global d`colnum´ : display %10.3f mydata[1, `colnum´]
}
// Or you can do it manually, like this (no formatting here):
// global d1 = mydata[1,1]
// global d2 = mydata[1,2]
// global d3 = mydata[1,3]

addrowrtf handle1 item mpg rep78 "Price in USD"
addrowrtf handle1 Mean $d1 $d2 $d3
file write handle1 "}" _n                          // End of table

//////////////////////////////////////////////////////

file write handle1 "{\pard\f0\fs24 \line\line That´s {\ul\b\scaps all}, " ///
   "folks!\line\line\par}"

endrtf handle1
file close handle1
clear all
```

5 Known issues

As stated previously, both commands have deliberately been designed to be as easy to use as possible, given the complexities of RTF usage. Although there are options to control output, the final RTF document will be viewed in a word processor and can be edited to the user's taste therein.

With `addrowrtf`, a truly long list of arguments (cell values) for a single row may cause the table row to extend past the right-hand edge of the page in some cases. It may be possible to keep such a long table row within the page margins by judicious use of options.

6 Conclusion

Using Stata to produce RTF files can be tricky. I introduced two commands that will hopefully aid users who wish to automatically generate RTF documents from Stata.

7 Acknowledgments

This work was supported by Cancer Research UK C8162/A16892. I wish to thank Stata Technical Support, especially Derek Wagner, who assisted with the conversion from byte values to hexadecimal values in the `insertimagertf` command. I also thank Dr. Francesca Pesola and Professor Peter Sasieni, both formerly of Queen Mary University of London and now of King's College London, for their helpful comments and suggestions. The suggestions of an anonymous reviewer greatly improved the final submission.

8 References

Burke, S. M. 2003. *RTF Pocket Guide*. Sebastopol, CA: O'Reilly.

Microsoft Corporation. 2008. Word 2007: Rich Text Format (RTF) Specification, version 1.9.1. https://www.microsoft.com/en-gb/download/details.aspx?id=10725.

Newson, R. 2009. rtfutil: Stata module to provide utilities for writing Rich Text Format (RTF) files. Statistical Software Components S457080, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s457080.html.

Nichols, A. 2010. png2rtf: Stata module to include PNG graphics in RTF documents. Statistical Software Components S457192, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s457192.html.

About the author

Matthew S. Gillman is a statistical programmer at King's College London. This work was undertaken at Queen Mary University of London.