PharmaSUG 2018 - Paper QT-01

ODS/RTF Pagination Revisit

Ya Huang, Halozyme Therapeutics, Inc.

Bryan Callahan, Halozyme Therapeutics, Inc.

ABSTRACT

ODS/RTF combined with PROC REPORT has been used to generate publication quality TFL by many people, and yet one problem has been persistently bothering them for years: Pagination of listings. When a listing has columns that hold long text (concatenated AE terms, Lab comments etc.), depending on the width of the columns and the length of the text, one line could wrap into multiple lines. Where the line wraps is decided by Microsoft® Word rather than SAS®. This leads to a nightmare of page break control. This paper presents an easy way to detect the number of lines needed when line wrapping occurs.

INTRODUCTION

Before ODS, many people relied on complicated macros to break up a string at proper positions [1], and they were able to calculate exactly how many lines were needed to hold the text. This is critical for those who use DATA _NULL_ for reporting. For those who use PROC REPORT, they could simply add a FLOW option in the define statement and PROC REPORT would take care of the line wrap and accurately control the page break for them. Both DATA _NULL_ and PROC REPORT work well because they are all plain text based; output is in fixed or monospace font where characters are equal in width, relatively easy for calculating the space needed.

With ODS/RTF, especially with proportional font such as Arial and Times New Roman, the number of lines needed to hold a long string is no longer predictable for 3 main reasons: 1. The total number of letters a line can hold varies due to unequal width of the letters. A string with many capital ‘W’s definitely needs wider space than a string with many lower case ‘i’s. 2. Order of “words” in the string. 3. Font attributes such as bold, italic, height, and even horizontal justification of the text. All of these may change the way MS Word wraps the line. Because of the unpredictability of line wrapping, SAS decided that instead of trying to predict the line wraps, it will simply rely on MS Word to do that. SAS came up with another ODS destination called TAGSETS.RTF, claiming that it offers better pagination control. In reality though, it has proven unreliable.

REAL EXAMPLE OF BAD PAGINATION

Figure 1 and 2 show a protocol deviation listing with a simple design: The first column is subject id/basic demographics, followed by visit, deviation category, and detailed description of deviation. In the data, the combination of the 4 columns (variables) uniquely identify each observation. Because the length of the detailed description are different, the number of lines needed by each observation varies from as few as 1 to as many as 7. Due to the unpredictability of lines needed for each deviation, the programmer decided to do some test runs using different numbers of observations per page followed by a visual examination of the output. He found out that 4 observations per page is the maximum number he could put without seeing split pages (Figure 1); otherwise, some pages will be split up by MS Word (Figure 2). While 4 observations works for this data, clearly the first 3 pages have wasted a lot of space. He could have put all the deviations from page 1 to 3 on one page and the space would still be enough.

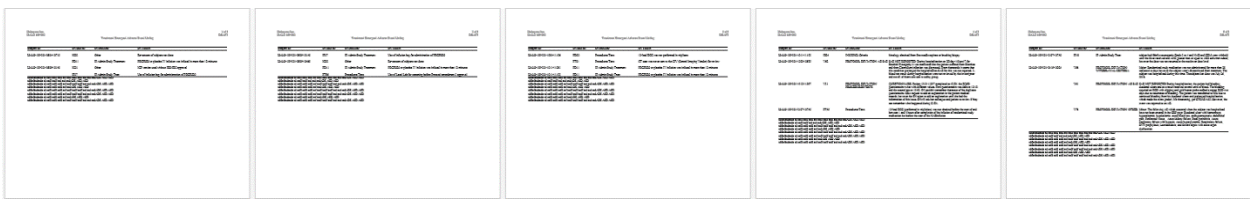

Figure 1. Bad pagination, page break every 4 observations, no split page, wasted space

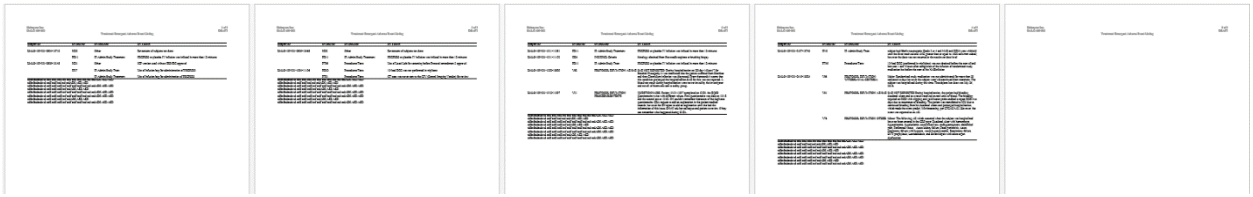

Figure 2. Bad pagination, page break every 5 observations, split page and wasted space

THE ROOT CAUSE OF BAD PAGINIATION: UNPREDICTABLE LINE WRAP

When ODS/RTF and PROC REPORT is used for tables and listings, the text is always confined in cells. Cell width is controlled by the style option in the define statement. Cell height, on the other hand, is usually not set by SAS, but by MS Word. MS Word calculates the height of the cell or physical lines needed based on many factors. As shown in Table 1 below, all five cells are equal in size. All strings inside the cells have exactly the same letters and fonts are in the same height too. The differences are the style and ordering of the words. These differences result in different wrapping positions; hence, a different number of physical lines needed to hold the string. This kind of unpredictability is the root cause of the bad pagination. Obviously, if we want to avoid the bad pagination, we need to know the accurate count of line wraps.

Line wraps due to different reasons	Line wraps due different reasons to	Line different reasons wraps due to	Line different reasons wraps due to	**Line different reasons wraps due to**	**Line wraps due to different reasons**

TABLE 1 SAME TEXT DIFFERENT WRAPPING POSITION

SOLVE THE LINE WRAP PROBLEM, THE BRUTE FORCE WAY

Over the years, several papers [2,3,4] have been published with the intention of solving the problem of unpredictable line wrapping and bad pagination. All proposed techniques help to solve the problem to some extent, but none of them will do a good job when a complex line wrap is involved. It is particularly difficult to get an optimal result when in-line formatting is used, for example the AE listing below.

Typically in an AE line listing, we need to show SOC, PT and Verbatim term of an AE within a column of limited width so that we have enough space left to show, in other columns, AE start and stop dates, study day, AE relationship to drugs, outcome, action taken, CTCAE grading, and serious flag etc. There are several ways to display the AE terms. The first and simplest way is to display a concatenated string with a '/' in between each term without a line break, as shown in the first cell below. Although easy to implement, it is very hard to read. The second way is to add a line break (as shown by the ¶ symbol) in between the terms. It is also pretty easy to implement ('^n' to connect two terms), but still hard to tell the difference between the terms when line wrap happens. The third way is to put an indentation between each term, a method where in-line formatting and raw RTF control word is involved. Clearly, this is the most visually appealing design.

System·Organ·Class·Line/Preferred·Terms·line/Verbatim·Term·line¤	System·Organ·Class·Line/¶ Preferred·Terms·line/¶ Verbatim·Term·line¤	System·Organ·Class·Line/¶ Preferred·Terms·line/¶ Verbatim·Term·line¤

Figure 3. Different ways to display AE terms.

As shown in figure 3, each of the three designs will use a different number of physical lines to hold the same string due to the line wrap. Brute force methods may be able to get a good estimate for the first design. It may also be able to get a close estimate for second design. But it is almost impossible to do that for the third design, because in-line formatting and raw RTF control word add extra text to the original AE term string, and they are invisible in the Word table cells. Any brute force estimation based on the original string will no longer work.

A BETTER WAY – LET MS WORD TELL US

Since the number of physical lines needed is decided by MS Word and hard to predict, the information is better off retrieved from MS Word itself. We all know that MS Word can convert an ODS/RTF Word table into a plain text file, but most of us probably don't know that the "save as" command has an "inset line breaks" option. When it is checked, MS Word not only can save the RTF file into a plain text file, it can also retain the line breaks, including the soft break. As shown in Figure 4, a simple one cell Word table with the indented AE terms is converted into a plain text file using the "save as" command. The two preview windows show the impact of the "insert a line breaks" option on the plain text file. When the option is unchecked (bottom left), the plain text shows 3 lines caused by the in-line formatting and raw RTF control words built in the string. When the option is checked (bottom right), the plain text shows 6 lines, the extra three are from the soft break that MS Word added. It matches the 6 lines inside the table cell. This can also be seen in a text editor, such as Notepad after the text file is created.

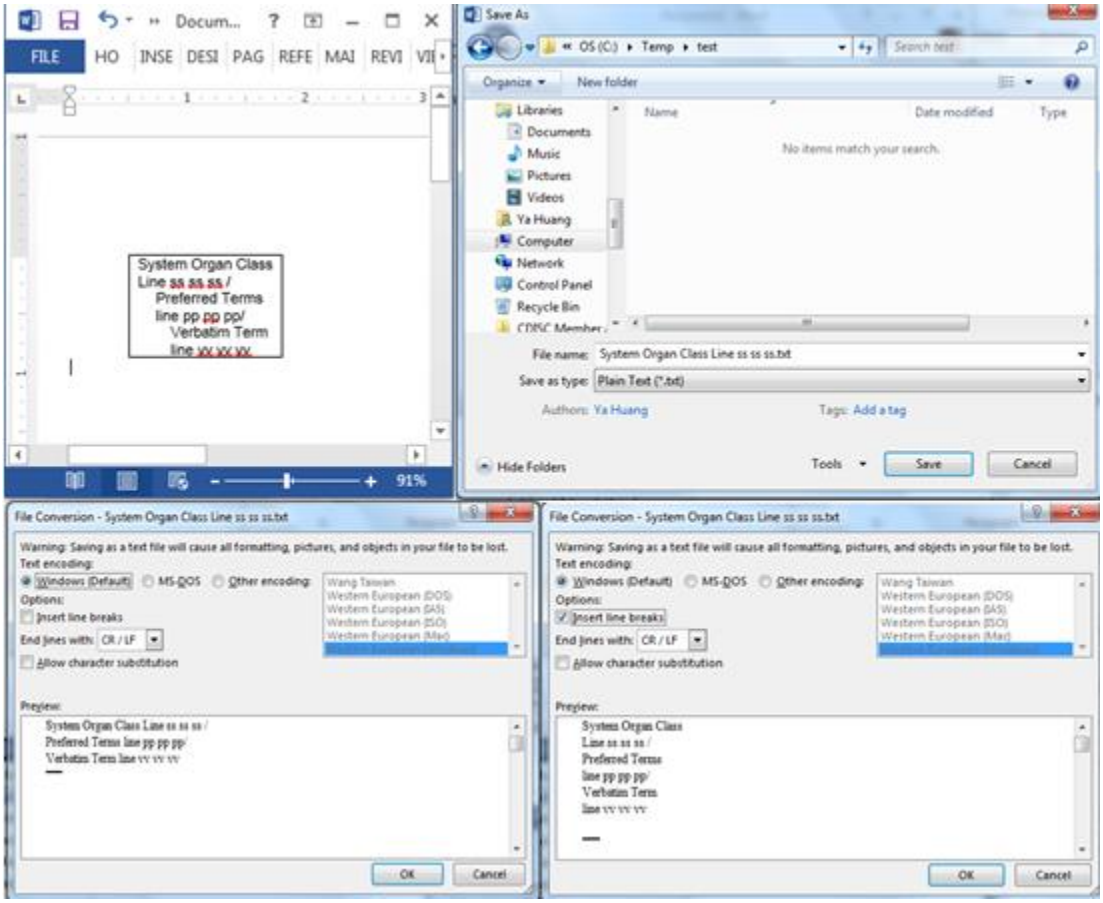

Figure 4 MS Word "Save as" to convert RTF to plain text and keep the line break.

This is a really exciting finding, it means that we've found an indirect way to retrieve the number of line wraps controlled by MS Word. We can take advantage of this finding by converting the RTF to a plain text file with "insert line breaks" option, and then count the number of lines in the text file. By doing so, we

determine how many lines the data really needs in the Word table. We can then set the page break accordingly and ultimately gain better page break control.

The idea is a very simple. To make it work, we just need to run ODS/RTF based PROC REPORT twice. The first PROC REPORT will create a temporary RTF file with two columns: the first column is the record ID (_n_ from the input dataset) and the second column is the text variable for which we want to retrieve the number of the line wraps. For the second column, we need to make sure the style options in the define statement are exactly the same as when it is used in the second PROC REPORT that will generate the final listing. After the first PROC REPORT is run, we need to invoke MS Word from within SAS and ask it to convert the temporary RTF file to a temporary plain text file with the inserted line break. This can be done by a simple VB Script. In the text file, lines from the first column are interlaced with lines from the second column. By parsing the text file, we can retrieve the original record ID, and the number of lines associated with each record ID. We can then merge it back to the original data by record ID. In this way, we add a new variable which carries the critical information, i.e. the number of line wraps.

A macro called %getlines is developed to do all of this for us. It has the following parameters:

```
%getlines(dsin=,     /* dataset for listing */
          colv=,     /* column variable we want to get the line count */
          colstyle=, /* column style used in second proc report */
          rptsyle=   /* report style in second proc report */
          )
```

After calling the macro, we will apply an enhanced page break setting algorithm to the dataset, then it will be ready for the second PROC REPORT to generate the final listing.

RESET PAGE BREAK FOR BETTER PAGINIATION

The algorithm of setting a page break variable is quite simple. Assuming that we want to put a page break on every N observations of the dataset, a simple data step as below will do the work.

```
data xx;
 set xx;
pg_=ceil(_n_/N);
run;
```

If we apply this algorithm to the protocol deviation listing discussed earlier, and try a different N, we will find that 4 is the maximum number we can do as we discussed earlier. Otherwise, we will see split pages.

Now let's see what happens if we call the macro and apply the enhanced page break algorithm to it.

```
%getlines(dsin=dv,
          colv=dvdesc,
          colstyle=%str(width=2.5in just=l),
          style=halotfl);
```

Calling the macro will add a new variable named dvdesc_n to dataset DV, which is named after the column variable dvdesc. An enhanced page break algorithm is shown below:

```
data dv;
 set dv;
by subjid visit dvcat dvdesc;
if _n_ then pg0_=1;
pg0_+dvdesc_n;
if last.dvdesc then pg0_+1; /* blank space in between each deviation */
pg_=ceil(pg0_/20); /* assume 20 is the physical lines each page can use
                      for listing content, excluding title and footnote */
```

```
run;

/* second proc report for the final listing */
proc report data=dv style=halotfl;
column pg_ subjid visit dvcat dvdesc;
define pg_/noprint order order=internal;
 …
define dvdesc /’Description’ style=[width=2.5in just=l]; /* this is used
                 for colstyle parameter */
…

break after pg_ /page;
run;
```

A MUCH BETTER RESULT

With the same dataset and an enhanced page break algorithm, we rerun the protocol deviation listing. The result looks much better. As shown in figure 5, the orignal pages 1 to 3 are now all fitting into page 1, and page 4 and 5 still look good, without splitting.

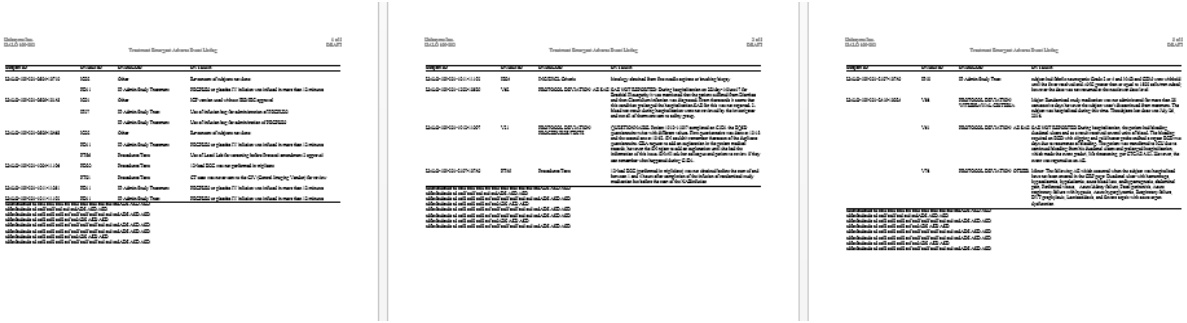

Figure 5, rerun of the example with better pagination

Conclusion

Unpredictable line wrapping causes bad pagination. The technique discussed in this paper provides a relatively robust way to overcome this problem. When a listing has multiple columns with potential wrapping issues, we can call the macro multiple times for different columns, and then use the maximum number of lines in the enhanced page break algorithm.

In this paper we assume the font height is fixed, and the method discussed here won’t be applicable if the font height is not fixed.

Since this technique needs MS Word and VB Script support, it obviously has to be run in a Windows platform. For those using Unix/Linux SAS, unfortunately, this won’t work.

REFERENCES

[1] H. Ian Whitlock, A Macro to Word Wrap Long Text Strings into a SAS® Array, SUGI 96

[2] Chao Su, William Conover, Pagination in Clinical Trial PROC REPORT ODS, MWSUG 2011

[3] Songtao Jiang, Daniel Boisvert, Effective Strategy to Set Page Breaks for ODS RTF Output. Pharmasug 2006

[4] John Kirkpatrick, Pagination in the ODS, Phuse 2005

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Ya Huang
Halozyme Therapeutics, Inc.
11388 Sorrento Valley Road
San Diego, CA 92121
United States
yhuang@halozyme.com

APPENDIX

```
%macro getlines(ds=,colv=,colstyle=,style=);


ods listing close;
ods rtf file="%sysfunc(pathname(work))\_getnlen.rtf" style=&style;
ods escapechar='^';
options nodate nonumber;
title;
footnote;

data &ds.0;
set &ds;
length n_ $50;
n_=cat('>>>>>>>> ',put(_n_,z8.));
run;

proc report data=&ds.0 noheader;
column n_ &colv;
define n_ / display style=[width=2in];
define &colv / style=[&colstyle];
run;

ods _all_ close;

filename script "%sysfunc(pathname(work))\wd2txt.vbs";
data _null_;
file script;
put 'Const wdFormatText = 2';
put 'Set objWord = CreateObject("Word.Application")';
put 'Set objDoc = objWord.Documents.Open("'
"%sysfunc(pathname(work))\_getnlen.rtf" '")';
put 'objDoc.SaveAs "' "%sysfunc(pathname(work))\_getnlen.txt" '",
wdFormatText,,,,,,,,,,True';
put 'objWord.Quit';
run;

filename rs pipe "cscript //nologo ""%sysfunc(pathname(work))\wd2txt.vbs""";
data _null_;
infile rs;
input;
put _infile_;
run;

data &ds.__1;
length n_ $50;
infile "%sysfunc(pathname(work))\_getnlen.txt";
input;
retain n_;
if _infile_=:'>>>>>>>>' then do;
n_=_infile_;
&colv._n=0;
end;
else &colv._n+1;
run;
```

```
data &ds.__1;
 set &ds.__1;
by n_;
if last.n_;
keep n_ &colv._n;
run;

data &ds;
 merge &ds.0 (in=a_) &ds.__1;
by n_;
if a_;
drop n_;
run;

%mend getlines;
```