

Come Out of Your Shell: A Dynamic Approach to Shell Implementation in Table and Listing Programs

Paul Burmenko, PRA International, Lenexa, KS
Tony Cardozo, PRA International, Lenexa, KS

ABSTRACT

In clinical trials, the SAS® programmer uses the Statistical Analysis Plan (SAP) provided by the statistician to create Table, Figure and Listing (TFL) programs to support the clinical trial submission. Table and listing shells or mock-ups from the SAP are the SAS programmer's roadmap in generating clinical study reporting outputs. Statisticians or SAS programmers often spend many hours to create shells manually in a word processing application, and then a SAS programmer must spend additional time mimicking the shell's text content and formatting in a table or listing SAS program. What if the shell designer's efforts could be harnessed to reduce the programmer's efforts? This paper presents a case study of how shell content and rich text formatting can be automatically read in from a shell document to create a SAS PROC REPORT shell as a starting point for a table or listing SAS program.

INTRODUCTION

To the SAS programmer charged with the creation of a table or listing for clinical trial reporting, a shell or mock-up is sacrosanct. Its every letter, cell alignment, and border to be painstakingly mimicked in the final table or listing output. To the shell's creator, the same table or listing shell represents hours spent manually typing and aligning text, sizing table cells, and tweaking formatting. What if the SAS programmer could infuse his or her SAS program with formatting directly from the shell document? What if the shell creator could rest assured that hours of creating immaculate shells reduced the effort of a programmer down the line?

Some biostatistics and programming departments have developed processes to develop shells from within SAS in order that the shell SAS program can then serve as the basis for the table or listing SAS program. But in some project- or sponsor-specific instances, this harmonized approach is not possible. For example, a clinical trial sponsor may develop all SAPs or shells in-house and outsource only the following reporting work.

Progress has already been made in utilizing shells to reduce programming effort, for example by reading the titles and footnotes from a shell document and populating them in the respective table or listing program (Gupta 2011 and Huang 2008). Others have proposed methods of reading final Rich Text Format (RTF) outputs into SAS in order to automate quality control (Chen 2012).

The purpose of this paper is to propose a way to marry the efforts of the table or listing shell creator with the efforts of the table or listing programmer in cases when the shells are designed manually. The method proposed has three main components: 1) to read in a table or listing shell document generated in RTF into SAS, 2) to parse the shell for text content and formatting, and 3) to translate that text and formatting into a starter table or listing SAS program.

NAVIGATING THE SHELL

RTF allows for the increased readability and presentation quality of clinical trial reporting outputs. Shells are designed prospectively with these concepts in mind.

Table and listing shells share five common components. From here on, only table shells will be discussed since they require the full functionality of ShellMap, while listings only require a subset of ShellMap features.

- 1) Title
- 2) Column headers
- 3) Results placeholders denoted by Xs.
- 4) Footnote(s)
- 5) Programming notes

Table shells contain another component:

- 6) Row headers.

Figure 1 shows an example table shell and its six components. The dashed gridlines are shown to demonstrate the structure of RTF tables, but they are not visible when printed.

Table 14.1.2.1 Demographic Characteristics (ITT Population)				1
Variable	Placebo (N=xxx)	RTF 500 mg (N=xxx)	Total (N=xxx)	2
Sex [n (%)]				6
Male	xx (xx.x)	xx (xx.x)	xx (xx.x)	
Female	xx (xx.x)	xx (xx.x)	xx (xx.x)	
Race [n (%)]				3
American Indian or Alaska Native	xx (xx.x)	xx (xx.x)	xx (xx.x)	
Asian	xx (xx.x)	xx (xx.x)	xx (xx.x)	
Black or African American	xx (xx.x)	xx (xx.x)	xx (xx.x)	
White	xx (xx.x)	xx (xx.x)	xx (xx.x)	
Other	xx (xx.x)	xx (xx.x)	xx (xx.x)	
Ethnicity [n (%)]				
Hispanic or Latino	xx (xx.x)	xx (xx.x)	xx (xx.x)	
Not Hispanic or Latino	xx (xx.x)	xx (xx.x)	xx (xx.x)	
Age (years) ¹				
n	xxx	xxx	xxx	
Mean (SD)	xx (xx.x)	xx (xx.x)	xx (xx.x)	
Median	xx.x	xx.x	xx.x	
Min, Max	xx, xx	xx, xx	xx, xx	
Note 1: Age is derived as informed consent date – date of birth / 365.25. Source: Listing 16.1.3.1, Dataset: ADSL, Program: T_DEMO1.sas				4
Programming Note: Shell creators often include additional instructions for table or listing creation at the bottom of the shell.				5

Figure 1. Generic Demographics Table Illustrates the Six Basic Components of a Table Shell (1-6) and Four Suggested Best Practices for Shell Design (i-iv).

FORMATTING BEST PRACTICES DURING SHELL DESIGN

By following basic shell design best practices, a biostatistics team can improve the overall quality and consistency of the final product, the Clinical Study Report post-text TFLs. These best practices cover multiple theoretical and practical topics, but outlined below and labeled in Figure 1 are a subset of formatting specifications consistent with shell best practices that are required for compatibility with ShellMap.

- i) Place the titles and footnotes in the same RTF table as the table body, not outside of it
- ii) Design with Gridlines turned on
- iii) Only include rows, columns, and line breaks where they are intended. An extra row should be used in place of the line break in this example in Figure 1
- iv) Label programming notes consistently to distinguish them from the footnotes

READING AND PARSING THE RTF SHELL

READING THE RTF SHELL FILE USING INFILE

The advantage of using the INFILE statement in a DATA step is that all RTF syntax related to table and text formatting is preserved for use by the ShellMap program.

```
data xxx;
  length rtf $32767.;
  infile "&shldir.\&shlname..rtf" length=linelen;
  input rtf $varying. linelen;
run;
```

PARSING RTF SYNTAX WITH PERL REGULAR EXPRESSIONS

RTF syntax contains predictable commands with many possible iterations. SAS functions like INDEX or TRANWRD are not enough to identify all possible RTF control words. Regular expressions make it possible to identify RTF code and to convert it for processing using the ODS RTF statement and REPORT procedure. There are many in-depth resources for this tool online.

Separating RTF Code from Shell Content

Three uses are made of the raw RTF file:

- 1) RTF code that is useful for identifying and formatting the output table cells and text using ODS RTF is translated to easily-identifiable ShellMap control terms using the DATA step function TRANWRD. This step is necessary so that useful RTF code is not lost during part 3. See Appendix 1 for a more comprehensive list of control terms (Burke 2003).

Table 1 lists the RTF control words described in this paper:

Purpose	RTF Control Word	ShellMap Control Word
New section (for page identification)	\sect	_sect
Table row	\trowd	_trowd
New row	\row	_row
New cell	\cell	_cell
New paragraph	\par	_par
New line (carriage return)	\line	_line
Non-breaking space	\~	[space character]
Unicode	\uN (where N represents a Unicode character)	_uN
Cell width	\cellxN (where N is width in twips)	_cellxN
Cell justification	\q (followed by 'l' for left, 'c' for center, 'r' for right, 'j' for full-justified)	_q
Superscript	\super (followed by the superscripted text)	_super
Hanging left indent	\liN (followed by width in twips)	_liN
Hanging right indent	\riN (followed by width in twips)	_riN
Hanging first-line indent	\fiN (followed by width in twips)	_fiN

Table 1. Sample of RTF Control Words Translated to ShellMap Control Words Before the Remaining Useless RTF Syntax is Discarded.

- 2) Characters and digits that are part of the shell and intended to be part of the final table output are written to a new variable.
- 3) All remaining RTF syntax is discarded using the DATA step function PRXCHANGE.

Figure 2 shows the code used to identify and discard unneeded RTF code:

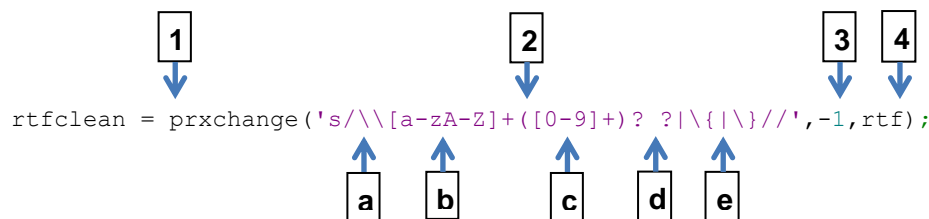


Figure 2. Perl Regular Expression Used in a DATA Step to Identify and Remove RTF Code That Was Not Translated in Step 1 Above.

- 1) Search and replace source variable using PRXCHANGE function and return modified string to destination variable.
- 2) Match the following patterns:
 - a) Backslash character “\”
 - b) followed by 1 or more non-case sensitive characters (RTF control words)
 - c) followed by 1 or 0 sets of 1 or more digits (measurement in twips)
 - d) Extra spaces
 - e) “{“ and “}” characters
- 3) Specify iterations to perform matching (-1 performs the function until no more matches are made)
- 4) Define source variable

CREATING THE SHELL MAP

The end of each RTF table row and table cell is identified with the tags \row and \cell, which are replaced with _row and _cell to prevent PRXCHANGE from removing them from the data. These tags are not reliably located on separate rows – multiple cell definitions can exist on the same row of RTF code. Since the goal is to transpose the RTF code into rows and columns using \row and \cell tags, a DATA step is necessary to split rows and cells into separate records. The same step is performed with the _line and _par control words to split text that contains carriage returns or multiple paragraphs into separate records. This step is especially useful for titles and footnotes since multiple titles or footnotes are contained in one RTF table cell spanning the entire page (see Figure 1 #4).

Figure 3 shows the before and after of the table cell splitting step. The same step is repeated for RTF rows.



Figure 3. Multiple RTF Cells Contained Within One SAS Data Set Record Are Split Across Multiple Records.

Next, a line counter variable is created to track data set row order, a page counter variable is created to track separate shell pages, and flag variables are retained from beginning to end of each table row or cell over the entire set of records associated with that row or cell.

```
data xxx;
    set xxx;
    *Count data set rows;
    line+1;
run;

*Sort in reverse order to identify RTF table rows and cells;
proc sort data=xxx;
    by descending line;
run;

data xxx;
    set xxx;
    if index(rtfclean, '_trowd') then rowstart+1;
    if index(rtfclean, '_row') then rowend+1;
run;

data xxx;
    retain page 0;
    set xxx;
    by rowend;

    if index(rtfclean, '_sect') then page+1;

    if first.rowend then cellend=.;
    if index(rtfclean, '_cell') then cellend+1;
run;
```

Transposing the Data

Once all the two-dimensional boundaries of the RTF table have been identified in proper order, the data set can be transposed 1) to consolidate all data set records associated with the same RTF table row and 2) to create one data set column for each RTF table column.

Transpose #1

```

*Descending order is used since PAGE, ROWEND, and CELLEND variables are ordered
starting from the end to the beginning of each element;
proc sort data=xxx;
    by descending page descending rowend descending cellend;
run;
proc transpose data=xxx out=xxx;
    by descending page descending rowend descending cellend;
    var rtfclean;
run;

*Count and combine the columns created during transposition associated with each row
and cell;
data _null_;
    set sashelp.vtable(where=(memname="xxx"));
    call symput('nvar',strip(put(nvar-4,best.)));
run;
data xxx;
    length text $1000;
    set xxx;
    text=catx(' ',of coll-col&nvar);
run;

```

Transpose #2

```

proc transpose data=xxx out=xxx prefix=col;
    by descending page descending rowend ;
    var text;
    id cellend;
run;

```

Deriving Sorting Variables for Tables

Sorting variables must be derived in order to identify titles (SORT1=-99), column header justification (SORT1=-98), column justification in the table body (SORT1=-97), column width (SORT1=-96), column headers (SORT1=0), each block of related rows in the table body (SORT1=1 for first block to N for Nth block), and footnotes (SORT1=99).

```

*Determine maximum number of columns in RTF table;
proc sql noprint;
    select max(cellend) into: maxcell
    from xxx;
quit;
%let maxcell=&maxcell;
data xxx;
    set xxx;
    by descending page;
    if first.page then sort1=-99;
    if last.page then sort1=99;
    array cells[*] coll-_

```

Another variable, SORT2 is incremented by 1 for each new row within the same SORT1 block (not shown).

Creating rows to store column width and justification information

The width and justification of every table cell is stored in the transposed data set. This information is output into a separate record distinguished by the sorting variable SORT1 for later use. Once this information is extracted and stored in separate records, it is removed from the shell data set cells. RTF control words that were kept in the data set for later use are translated back to RTF using the DATA step TRANWRD function and concatenated with the ODS RTF control string ^R/RTF. Brackets and quotation marks are also added as needed (e.g. the superscript "1"

character in the example shell is read in from the original RTF file as “\super 1”, is translated to “_super 1” during the processing and transposition step, and is translated one final time to “^R/RTF\super 1 ’”) for final presentation. See Appendix 2 for an example of one RTF cell’s full journey through the process.

Display 1 shows a sample shell data set:

	col1	col2	col3	col4	sort1	page	sort2
1	Table 14.1, 2.1 Demographic Characteristics (ITT Population)				.99	1	0
2	I	c	c	c	.98	1	0
3	I	c	c	c	.97	1	0
4	38.7	19.7	19.7	19.9	.96	1	0
5	Variable	Placebo (N= xxx)	RTF 500 mg (N=xxx)	Total (N= xxx)	0	1	0
6					0	1	0
7	Sex [n (%)]				1	1	1
8	^R/RTF\%200 ' Male	xx [xx.x]	xx [xx.x]	xx [xx.x]	1	1	2
9	^R/RTF\%200 ' Female	xx [xx.x]	xx [xx.x]	xx [xx.x]	1	1	3
10					1	1	4
11	Race [n (%)]				2	1	1
12	^R/RTF\%200 ' American Indian or Alaska Native	xx [xx.x]	xx [xx.x]	xx [xx.x]	2	1	2
13	^R/RTF\%200 ' Asian	xx [xx.x]	xx [xx.x]	xx [xx.x]	2	1	3
14	^R/RTF\%200 ' Black or African American	xx [xx.x]	xx [xx.x]	xx [xx.x]	2	1	4
15	^R/RTF\%200 ' White	xx [xx.x]	xx [xx.x]	xx [xx.x]	2	1	5
16	^R/RTF\%200 ' Other	xx [xx.x]	xx [xx.x]	xx [xx.x]	2	1	6
17					2	1	7
18	Ethnicity [n (%)]				3	1	1
19	^R/RTF\%200 ' Hispanic or Latino	xx [xx.x]	xx [xx.x]	xx [xx.x]	3	1	2
20	^R/RTF\%200 ' Not Hispanic or Latino	xx [xx.x]	xx [xx.x]	xx [xx.x]	3	1	3
21					3	1	4
22	Age [years] ^R/RTF\super 1 '				4	1	1
23	^R/RTF\%200 ' n	xxx	xxx	xxx	4	1	2
24	^R/RTF\%200 ' Mean (SD)	xx [xx.x]	xx [xx.x]	xx [xx.x]	4	1	3
25	^R/RTF\%200 ' Median	xx.x	xx.x	xx.x	4	1	4
26	^R/RTF\%200 ' Min, Max	xx, xx	xx, xx	xx, xx	4	1	5
27	Note 1: Age is derived as informed consent date - date of birth / 365.2				99	1	1
28	Source: Listing 16.1.3.1, Dataset: ADSL, Program: T_DEM01.sas				99	1	2
29					99	1	3
30	Programming Note: Shell creators often include additional instructions for table or listing creation at the bottom of the shell.				99	1	4

Display 1. Example of Fully Processed RTF Shell Converted to a SAS Data Set

CALLING THE SHELL MAP FOR THE DESIRED OUTPUT

The RTF-formatted SAP document is parsed for all shells using the framework described above, and each shell is output as a SAS data set to a shell library with a unique name, the most convenient being International Conference on Harmonization nomenclature as shown in Display 2.

Display 2 shows the ShellMap data set library:

Name	Size	Type
T_14_02_01_03_...		Table
T_14_02_01_04_...		Table
T_14_02_01_04_...		Table
T_14_02_01_04_...		Table
T_14_02_01_04_...		Table
T_14_02_01_04_...		Table
T_14_02_01_05_...		Table
T_14_02_01_06_...		Table
T_14_02_01_07_...		Table
T_14_02_01_08_...		Table
T_14_02_01_09_...		Table
T_14_02_01_10_...		Table

Display 2. Example of ShellMap Library Generated from the SAP Shells Document

The name of the ShellMap data set corresponding to each table output is passed as a parameter to a ShellMap subroutine macro called in each individual table program.

Example call to %import_shell:

```
%import_shell(shell=t_14_01_02_01_demo);
```

GENERATING THE TITLES AND FOOTNOTES

The %import_shell macro called by each individual table program passes strings contained in all rows associated with SORT1=-99 to macro variables to store the title text and number of title rows. The same type of logic is used for SORT1=99 footnote rows. A macro variable is also generated to store the count of titles and footnotes. The counts variables are used to dynamically define the upper limit of loops during later parts of the process.

```
if sort1 = -99 then do;
    call symputx('title' || strip(put(_n_, best.)), col1, 'G');
    titlecnt+1;
end;
if eof then do;
    call symputx('title_cnt', titlecnt, 'G');
end;
```

PASSING COLUMN HEADERS TO PROC REPORT

The number of columns in the shell data set is counted and passed to a macro variable. The column number variable is used in all succeeding subroutines to dynamically set loop iteration.

```
proc contents data=&shell out=col_&shell (where=(name =: 'col')) noprint;
run;

data _null_;
    set col_&shell end=eof;
    if eof then call symputx('col_num', _n_, 'G');
run;
```

Once the number of columns is established, the macro loops through each column of the shell data set to pass all the column values for the column header justification (SORT1=-98), column justification for the table body (SORT1=-97), column width (SORT1=-96), and column header text (SORT1=0) records. The following type of logic is used for each SORT1 value using unique macro variable names (represented as ellipses due to the redundancy of the logic).

```
*Define header justification using &header_just;
if sort1 = -98 then do;
    %do ____j = 1 %to %eval(&col_num);
        call symputx('header_just' || strip(put(&____j, best.)), col&____j, 'G');
    %end;
end;
else if sort1 = -97 then do;
*Define column justification using &column_just;
    ...
end;
else if ...
```

MERGING THE SHELL MAP WITH THE STATISTICAL RESULTS DATA SET

Since the ShellMap data set for the desired output has been called by the %import_shell macro, the programmer can now merge the header rows in the COL1 variable with the statistical results data set output by the Means, Freq, Mixed, or any other SAS Procedures. All that is required is for the programmer to assign the same SORT1 and SORT2 values to the statistical results as are assigned to the corresponding ShellMap header rows in the imported ShellMap data set and to perform the merge. This step is useful only for tables that contain pre-determined header row values, such as the demographics shell example used throughout this paper. Some tables contain header rows determined by the data (e.g. system organ class and preferred term values from the adverse events data are used as header rows for adverse events tables). Some tables contain a mixture – header rows determined by the data (e.g. laboratory category and parameter values from the laboratory data) and pre-determined header rows (e.g. “n,” “Mean (SD),” “Median,” and “Min, Max” headers for descriptive statistics for continuous variables). Listings contain no header rows at all. Therefore, it is the programmer’s prerogative to select the SORT1 and SORT2 values associated with ShellMap data set records that are useful for creating the table or listing output.

Display 3 shows partial ShellMap and statistical results data sets side by side:

col1	sort1	sort2	COL4	SORT1	SORT2
Sex [n (%)]	1	1			
^R/RTF^i200 ' Male	1	2			
^R/RTF^i200 ' Female	1	3	^R^q\tddec\tx600 ' 159 (93.0)	1	2
	1	4	^R^q\tddec\tx600 ' 12 (7.	1	3
Race [n (%)]	2	1	^R^q\tddec\tx600 ' 2 (1.2	2	2
^R/RTF^i200 ' American Indian or Alaska Native	2	2	^R^q\tddec\tx600 ' 8 (4.7)	2	3
^R/RTF^i200 ' Asian	2	3			

Display 3. The ShellMap Data Set Merged With Statistical Results by Variables SORT1 and SORT2

GENERATING THE RTF OUTPUT FILE

The final step of the table program is to output the RTF file using the data set containing the merged ShellMap and statistical results. This final step is performed by another ShellMap macro subroutine called %proc_report. This subroutine allows the user to choose between using the justifications, widths, and column headers from the ShellMap or overwriting the ShellMap values with user-defined values. For example, a user may choose to leave the justifications and column headers as defined in the ShellMap data set, but to manually pass through values to be used for column width definitions in PROC REPORT. User-defined formatting values are passed to the macro as parameters for each column of the ShellMap. For the parameters that the user chooses not to define, the values from the ShellMap are used.

Below is an excerpt from the %proc_report macro. This code assumes that the titles and footnotes from the ShellMap data set are to be printed in the header and footer of the RTF document with user-defined justification parameters (&TJUST for titles and &FJUST for footnotes). The table-specific titles follow the global study-wide headers (the number of which is stored in the &head_cnt macro variable). An alternative approach (not shown) is to populate the titles as the first row of the RTF table rather than the RTF document header and the footnotes as the last row of the RTF table rather than the RTF document footer (Carr 2009). The SAS logic for this alternative approach would utilize the same title and footnote macro variables created in the “Generating Titles and Footnotes” section above.

```
*Write report file to RTF destination;
ods rtf file = %str("%&outpath\&rtf_out..rtf%");
ods listing close;

*Populate all title header rows;
%do ___i = 1 %to %eval(&title_cnt);
  title%eval(&head_cnt+&___i) %str("%&&title&___i%") j=&tjust;
%end;

*Generate report;
proc report data=&dsin nowindows missing headline headskip split = '|' ;
*Loop through all report columns and populate column headers;
  column sort1 sort2 %do ___i = 1 %to %eval(&col_num); " col&___i" %end; ;
  define sort1 / order order = internal noprint;
  define sort2 / order = internal noprint;
*Loop through all report columns and populate justification and width;
%do ___j = 1 %to %eval(&col_num);
  define col&___j / display
    style(column) = {just = &&column_just&___j
                    cellwidth = &&width&___j.%;}
    style(header) = {just = &&header_just&___j}
    %str("%&&col_name&___j%");
  %end;
run;

*Populate all footer footnote rows;
%do ___i = 1 %to %eval(&foot_cnt);
  footnote&___i %str("%&&foot&___i%") j=&fjust;
%end;
*Stop writing to RTF destination;
ods rtf close;
ods listing;
```


Display 4 shows a sample completed demographics table:

Carmenco, Inc.
RTF2013-101

Page 1 of 1
December 30, 2013

**Table 14.1.2.1 Demographic Characteristics
(ITT Population)**

Variable	Placebo (N=42)	RTF 500 mg (N=42)	Total (N=84)
Sex [n (%)]			
Male	18 (42.9)	21 (50.0)	39 (46.4)
Female	24 (57.1)	21 (50.0)	45 (53.6)
Race [n (%)]			
American Indian or Alaska Native	0 (0.0)	0 (0.0)	0 (0.0)
Asian	4 (9.5)	2 (4.8)	6 (7.1)
Black or African American	15 (35.7)	22 (52.4)	37 (44.0)
White	22 (52.4)	15 (35.7)	37 (44.0)
Other	1 (2.4)	3 (7.1)	4 (4.8)
Ethnicity [n (%)]			
Hispanic or Latino	4 (9.5)	4 (9.5)	8 (9.5)
Not Hispanic or Latino	38 (90.5)	38 (90.5)	76 (90.5)
Age (years) ¹			
n	42	42	42
Mean (SD)	37.2 (8.67)	36.8 (10.51)	38.0 (9.54)
Median	39.5	38.0	38.5
Min, Max	25, 68	21, 69	21, 69
Age group [n (%)]			
< 65 years	37 (88.1)	40 (95.2)	77 (91.7)
≥ 65 years	5 (11.9)	2 (4.8)	7 (8.3)
65 - 74 years	5 (11.9)	2 (4.8)	7 (8.3)
≥ 75 years	0 (0.0)	0 (0.0)	0 (0.0)

Note 1: Age is derived as informed consent date – date of birth / 365.25.
Source: Listing 16.1.3.1, Dataset: ADSL, Program: T_DEMO1.sas

Display 4. Example of RTF Table Output

SPECIAL CONSIDERATIONS

This section addresses some common shell variations and suggests possible improvements to ShellMap functionality, which are not discussed in detail.

HANDLING PANELED OUTPUTS

Table and listing outputs sometimes panel across multiple pages due to page width limitations. For example, a dose escalation protocol may call for more than the typical two to four different investigational drug treatment columns in the tables. In this case, the table may be paneled with the same header rows in column 1 spanning across two or more pages so that all the treatment columns may be presented. Though not shown here, ShellMap framework allows for the programmatic identification of paneled outputs. This can be achieved using either DATA step BY-Group processing with FIRST.variable and LAST.variable logic or the SQL procedure using the COUNT function with the GROUP BY clause. In the event that paneling is identified, the shell map data set is transposed from vertical to horizontal with the paneled columns handled the same way as in a non-paneled table. A MACRO variable flag is turned on to pass the ID and PAGE options in PROC REPORT DEFINE statements for the fixed and paneled columns, respectively.

CREATING TABLES WITH SPANNING COLUMN HEADERS

Some shells contain headers that span across multiple columns. In such cases, the table header has multiple rows instead of only one row and must be handled differently.

Display 5 shows an example of spanning headers:

Table 14.1.1.3 Subject Disposition (ITT Population)

Spanning column header	RTF					
Variable	Placebo (N=XXXX) n (%)	5 mg (N=XXXX) n (%)	10 mg (N=XXXX) n (%)	20 mg (N=XXXX) n (%)	Sub-total (N=XXXX) n (%)	Total (N=XXXX) n (%)

Display 5. Example of a Spanning Header Across Multiple Columns

Since ShellMap processes RTF code rows in descending order to facilitate identification of table row and column boundaries, spanning column header text ends up in the right-most of the columns over which it spans. Since SAS data sets cannot have merged columns the way RTF can, the other columns in the spanning column header row have null values. Without special handling, spanning column headers present a challenge – any regular column header on the second row and to the left of the spanning column header could potentially be part of the spanning column header in the first row (see Display 5). Though more dynamic approaches are under investigation, one potential solution is create a macro parameter for %proc_report to be utilized by the user to define the columns on the succeeding column header ShellMap data set record that are described by the spanning column header.

CONCLUSION

Post-text TFLs created for clinical trial submissions continue to become more standardized. The future could bring harmonized industry reporting standards that could be supported by a library of standardized programs. However, the intricacies of reporting results for complex clinical trials used to evaluate treatments for drastically different indications will always require some level of shell customization. The charge of every SAS programmer analyzing and reporting clinical study data should be to approach these intricacies with a mind toward innovation and efficient use of resources. The goal of this paper is to address some specific tools and methods that can help to realize efficiencies and to spark elegant and innovative solutions in the future.

REFERENCES

Burke, Sean. "The RTF Cookbook." 2003. Available at <http://search.cpan.org/~sburke/RTF-Writer-1.11/lib/RTF/Cookbook.pod>

Burke, Sean. 2003. *RTF Pocket Guide*. Sebastopol, CA: O'Reilly Media, Inc. Available at http://books.google.com/books?id=3yA1cG6_Tu4C&pg=PA82&ots=P0-7rjvhQf&dq=%5Cbrdrw15&pg=PA82#v=onepage&q=%5Cbrdrw15&f=false

Carr, David. 2009. "Getting More from the Compute Block to Enhance Table and Listing Output for Clinical Trials Reporting." PharmaSUG 2009 Annual Conference. Available at: <http://www.lexjansen.com/pharmasug/2009/tt/tt13.pdf>

Chen, Honghua. 2012. "Prove QC Quality – Create SAS ® Dataset from RTF file." *NESUG 2012 Annual Conference*. Available at: <http://www.nesug.org/Proceedings/nesug12/ph/ph03.pdf>

Gupta, Ajay. 2011. "Reading Title and Footnote from RTF Output into SAS® utilizing Microsoft® Excel." *PharmaSUG 2011 Annual Conference*. Available at <http://www.pharmasug.org/proceedings/2011/CC/PharmaSUG-2011-CC11.pdf>

Huang, Daniel; Lynn, Lois. 2008. "Read Titles and Footnotes from a Table Shell into a SAS Dataset." *PharmaSUG 2008 Annual Conference*. Available at <http://www.lexjansen.com/pharmasug/2008/po/po07.pdf>.

RECOMMENDED READING

Microsoft. Word 2007: Rich Text Format (RTF) Specification, version 1.9.1. 2008. Available at <http://www.microsoft.com/en-us/download/details.aspx?id=10725>

SAS Knowledge Base. "Perl Regular Expressions Tip Sheet." Available at http://support.sas.com/rnd/base/datastep/perl_regexp/regexp-tip-sheet.pdf

Steele, Oliver. "reWork." 2003. Available at: <http://osteele.com/tools/rework/>

CONTACT INFORMATION

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

Name:	Paul Burmenko	Tony Cardozo
Enterprise:	PRA International	
Address:	9755 Ridge Drive	
City, State ZIP:	Lenexa, KS 66219	
Work Phone:	(913) 410-2154	(913) 410-2592
E-mail:	burmenkopaul@praintl.com	CardozoTony@PRAIntl.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

Appendix 1

Purpose	RTF Control Word	ShellMap Control Word
New section (for page identification)	\sect	_sect
Table row	\trowd	_trowd
New row	\row	_row
New cell	\cell	_cell
New paragraph	\par	_par
New line (carriage return)	\line	_line
Non-breaking space	\~	[space character]
Unicode	\uN (where N represents a Unicode character)	_uN
Cell width	\cellxN (where N is width in twips)	_cellxN
Cell justification	\q (followed by 'l' for left, 'c' for center, 'r' for right, 'j' for full-justified)	_q
Superscript	\super (followed by the superscripted text)	_super
Subscript	\sub (followed by the subscripted text)	_sub
Underline	\ul	_ul
Bold	\b	_b
Italic	\i	_i
Hanging left indent	\liN (where N is width in twips)	_liN
Hanging right indent	\riN (where N is width in twips)	_riN
Hanging first-line indent	\fiN (where N is width in twips)	_fiN
Decimal tab	\tqdec	_tqdec
Bottom border	\brdrb	_brdrb
Top border	\brdrt	_brdrt
Left border	\brdrl	_brdrl
Right border	\brdrr	_brdrr
Single-thickness border	\brdrs	_brdrs
Border width	\brdrwN (followed by thickness in twips)	_brdrwN

Table 2. RTF Control Words Translated to ShellMap Control Words

Appendix 2

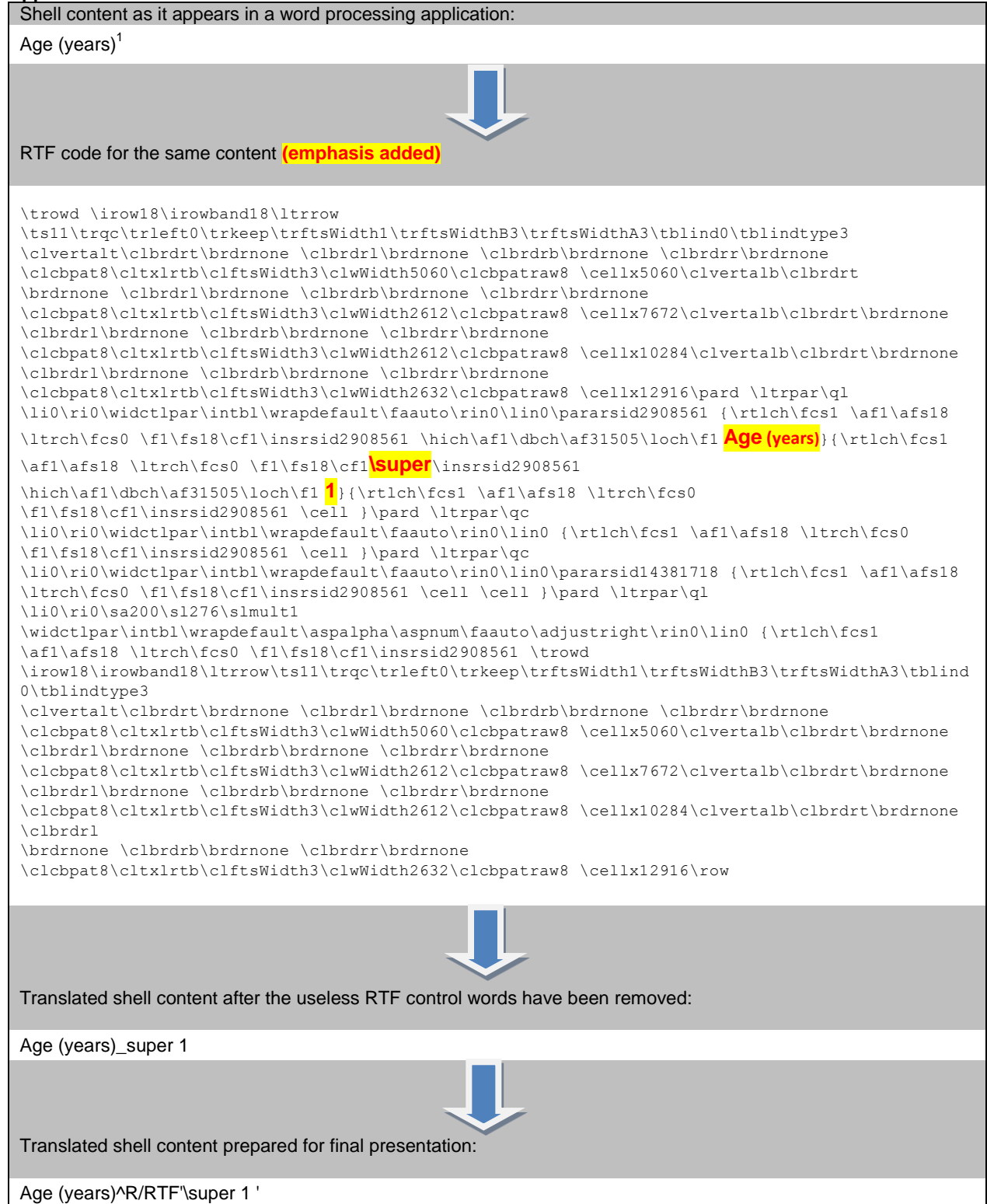


Table 3. Diagram of the Full RTF Translation Process As It Applies to One Single RTF Table Cell.