ABSTRACT
As a composite table, a stack table is made up of a variable number of child tables that are placed one on top of another with the same width but different heights. Unlike a regular table that has a fixed number of columns throughout its rows, a typical stack table allows its child tables to have their own number of columns and headers and each can vary its format and style independently of the other(s). This gives a great freedom to data presentation that a regular table just cannot meet or beat even with complicated spanning and sizing techniques.

Stack tables are frequently used in clinical study analysis and reporting because of their effective appealing presentations of various data derived from different sources. However, Proc Report, as a powerful and versatile reporting tool in the SAS system, only produces regular tables; therefore, stack tables are usually implemented with lengthy and cryptic DATA NULL step(s). In this paper, I first describe a simple macro called %glue that can combine a group of regular RTF child tables generated by Proc Report step(s) into a seamless RTF stack table, and then present a simple programming framework for how to construct RTF stack tables step by step with Proc Template, ODS RTF, and Proc Report. Numerous code snippets are provided to demonstrate the powerfulness and flexibility of this new technique.

INTRODUCTION
A stack table is a composite table that is made up of a variable number of child tables that are seamlessly placed one on top of another. The major difference between a stack table and a regular table is that the stack table consists of arbitrary number of child tables with varying number of columns while the regular table consists of fixed number of columns. In fact, a regular table is just a special case of a stack table. The real power behind the stack table is that each of its child tables can vary its format and style independently of the other(s). This gives a great freedom to data presentation that a regular table just cannot meet or beat even with complex spanning and sizing techniques.

Analysis of Average Change From Baseline in YYY1 (L)
Treatment Period

<table>
<thead>
<tr>
<th>Treatment Group</th>
<th>N†</th>
<th>Mean</th>
<th>SD</th>
<th>LS Mean</th>
<th>95 % CI for LS Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drug A</td>
<td>114</td>
<td>0.38</td>
<td>0.32</td>
<td>-0.04</td>
<td>(-0.08, 0.00)</td>
</tr>
<tr>
<td>Drug B</td>
<td>115</td>
<td>0.37</td>
<td>0.28</td>
<td>-0.04</td>
<td>(-0.08, 0.01)</td>
</tr>
<tr>
<td>Placebo</td>
<td>113</td>
<td>0.38</td>
<td>0.30</td>
<td>0.04</td>
<td>(-0.00, 0.08)</td>
</tr>
</tbody>
</table>

Between-treatment Comparisons from ANOVA Model

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Difference in LS Means</th>
<th>%95 CI for Difference</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drug A vs. Placebo</td>
<td>-0.08</td>
<td>(-0.14, -0.02)</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>Drug B vs. Placebo</td>
<td>-0.08</td>
<td>(-0.14, -0.03)</td>
<td>0.412</td>
</tr>
<tr>
<td>Drug A vs. Drug B</td>
<td>0.00</td>
<td>(-0.05, 0.06)</td>
<td>0.999</td>
</tr>
</tbody>
</table>

†N is the number of patients used in the ANOVA analysis.

Figure 1. A sample stack table used in the clinical study report
Stack tables are frequently used in clinical study reports because of their effective appealing presentations of various integrated statistical results that are derived from different sources and separate aspects of concerns. For example, Figure 1 above shows a sample RTF stack table from the clinical study report:

The stack table in Figure 1 is made up of three child tables, one on top of another. Although the styles of their headers and the number of their columns are different, the three child tables are of same width and integrated without any space between them.

A great deal of effort has been made to create stack tables in RTF (Rich Text Format) quickly and easily [1][2]. In this paper, I introduce a new method that turns a set of regular RTF tables produced by Proc Report procedure(s) into a RTF stack table. I first describe a magic macro called %glue that can integrate all RTF child tables outputted by Proc Report steps in a RTF stack table, and then provide a simple yet flexible framework for constructing RTF stack tables step by step with Proc Template, ODS RTF and Proc Report. Numerous code snippets are given to illustrate how to use this powerful technique with great ease.

The technique developed in this paper is based on the Version 9.1 of the SAS system.

**MAGIC %GLUE**

Proc Report, as a very powerful report-writing tool in the SAS System, has been used by many pharmaceutical companies for regulatory submission. When given a report dataset, Proc Report has the ability to create a regular RTF table with different styles of headers, columns, and individual cells under the help of ODS RTF. This flexibility works very well with a regular table, but not with a typical stack table. This is because a typical stack table consists of a varying number of child tables, and each child table has its own dataset, layout structure and appearance.

With ODS RTF startpage=no BODYTITLE setting, a bunch of RTF tables generated by Proc Report steps can be put into one page, but those tables are obviously separate, and with space between them. Figure 2 is a sample program that creates two tables in a single page, and Figure 3 is the outputted RTF table.

```sas
option nomprint nosymbolgen nomlogic nodate nonumber;
ods rtf file = "c:\ex1.rtf" startpage = no bodytitle;
ods escapechar = '^';
title1 "Analysis of Average Change from Baseline";
proc report nowindows data=summary style(report)={outputwidth=6in}
   style(header)={background=white};
   column trt n ('Baseline' mean sd) ('Average Change from Baseline' lsmean ci);
   define trt/display 'Treatment Group' style(column)=[cellwidth=1.8in just=center];
   define n/display 'N^{super *}' style(column)=[just=center];
   define mean/display 'Mean' style(column)=[just=center];
   define sd/display 'SD' style(column) =[just=center];
   define lsmean/display 'LS Mean' style(column)=[just=center];
   define ci/display '95 % CI for LS Mean' style(column)=[just=center];
footnote1;
footnote2;
run;
proc report nowindows data=comp style(report)={outputwidth=6in}
   style(header)={background=white};
   column ('{\~} ' ('Between-treatment Comparisons from ANOVA Model' comp diff ci pvalue));
   define comp/display 'Comparison' style(column)=[cellwidth=1.27in just=center];
   define diff/display 'Difference in LS Means' style(column)=[just-center] format=5.2;
   define ci/display '%$5 CI for Difference' style(column)=[just-center];
   define pvalue/display 'p-value' style(column)=[just-center];
title1;
footnote1 "^{(super *)}N is the number of patients used in the ANOVA model analysis.";
```
Figure 2. Sample SAS code to create two tables on the same page

The output is shown below.

### Analysis of Average Change From Baseline in YYY₁ (L)
#### Treatment Period

<table>
<thead>
<tr>
<th>Treatment Group</th>
<th>N†</th>
<th>Mean</th>
<th>SD</th>
<th>LS Mean</th>
<th>95 % CI for LS Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drug A</td>
<td>114</td>
<td>0.38</td>
<td>0.32</td>
<td>-0.04</td>
<td>(-0.08, 0.00)</td>
</tr>
<tr>
<td>Drug B</td>
<td>115</td>
<td>0.37</td>
<td>0.28</td>
<td>-0.04</td>
<td>(-0.08, 0.01)</td>
</tr>
<tr>
<td>Placebo</td>
<td>113</td>
<td>0.38</td>
<td>0.3</td>
<td>0.04</td>
<td>(-0.00, 0.08)</td>
</tr>
</tbody>
</table>

† N is the number of patients used in the ANOVA analysis.

### Between-treatment Comparisons from ANOVA Model

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Difference in LS Means</th>
<th>%95 CI for Difference</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drug A vs. Placebo</td>
<td>-0.1</td>
<td>(-0.14, -0.02)</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>Drug B vs. Placebo</td>
<td>-0.1</td>
<td>(-0.14, -0.03)</td>
<td>0.412</td>
</tr>
<tr>
<td>Drug A vs. Drug B</td>
<td>0.0</td>
<td>(-0.05, 0.06)</td>
<td>&gt;0.999</td>
</tr>
</tbody>
</table>

Figure 3. The two separate RTF tables generated by two Proc Report code under ODS RTF settings

Although those two separated regular RTF tables cannot be regarded as a stack table, they seem perfect to be used as the child tables of a stack table. Is there any way to combine the separated regular tables into a stack table by squeezing out the unwanted space between tables? Yes, there is. As we all know, any RTF file is a text file that consists of structured RTF codes. Therefore, if Proc Report steps and ODS RTF statements cannot directly create RTF tables that you want, you can modify the output RTF files by:

1. Inserting some RTF control words and symbols into RTF table files during the generation, and/or
2. Removing or replacing some RTF control words and symbols in the generated RTF table files.

The benefit of using those two approaches is that you don’t have to reinvent the wheel, or start the whole RTF table work from scratch. After carefully examining the codes in the RTF files generated by Proc Report steps under ODS RTF startpage=no BODYTITLE settings, I find the gaps between tables actually fulfill the following regular expression:

First line: /pard\{par\}/
Second line: /{par}\{pard\plain\qc\}/
Third line: /{par}\{par\}/
Fourth line: blank
Fifth line: /sect\{sect\linetext\{d\}\endnhere\sbknone\headery\{d\}\footery\{d\}\marglx\margrx\margtx
If the RTF code sequences that cause gaps between RTF child tables in the generated text file can be removed, you can have a perfect RTF stack table. The small macro `%glue` below does the trick.

```sas
%Macro glue(in=, out=);%local QueueSize; %let QueueSize=5; filename infile "&in"; filename outfile "&out"; data _TMPDSN; length rtfcode $32000; infile infile; * original rtf tables created by ODS; input; rtfcode=_infile_; length=length(trim(rtfcode)); run;

data _null_; file outfile; * rtf stack table with no gaps between child tables; set _TMPDSN end=last; array queue{&QueueSize} $32000 _temporary_; retain queue; retain count 0; count = count + 1; queue[count] = rtfcode; if count = &QueueSize then do; found = 0; if queue[1] = '\pard\par' and queue[2] = '{ \par}{\pard\plain\qc' and queue[3] = '} \par}{\par}' and queue[4] = ' ') then do; if (compress(translate(queue[&QueueSize],",", "0123456789")) = '\sect\sectd\linex\endnhere\sbknone\headery\footery\marglsxn\margrsxn\margtsx xn\margbsx
n') then found=1; end; if found then do; count=0; end; else do; put queue[1]; do i = 2 to &QueueSize; queue[i-1]=queue[i]; end; count = count -1; end; end; if last then do; do i = 1 to count; put queue[i]; end; end; run; %Mend glue;
```

The `%glue` actually does a very simple thing: it takes an existing RTF file name as an input parameter, reads the RTF file into a dataset, searches and removes the RTF code sequences, or gaps from the dataset, and finally write the cleaned-up dataset to an output file. For example, with the following macro call.
You will produce a RTF stack table in file ex2.rtf that is the combination of the two tables in Figure 3. The RTF stack table is showed below.

### Analysis of Average Change From Baseline in YYY \textsubscript{1} (L)

#### Treatment Period

<table>
<thead>
<tr>
<th>Treatment Group</th>
<th>N$^\dagger$</th>
<th>Mean</th>
<th>SD</th>
<th>LS Mean</th>
<th>95 % CI for LS Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drug A</td>
<td>114</td>
<td>0.38</td>
<td>0.32</td>
<td>-0.04</td>
<td>(-0.08, 0.00)</td>
</tr>
<tr>
<td>Drug B</td>
<td>115</td>
<td>0.37</td>
<td>0.28</td>
<td>-0.04</td>
<td>(-0.08, 0.01)</td>
</tr>
<tr>
<td>Placebo</td>
<td>113</td>
<td>0.38</td>
<td>0.3</td>
<td>0.04</td>
<td>(-0.00, 0.08)</td>
</tr>
</tbody>
</table>

#### Between-treatment Comparisons from ANOVA Model

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Difference in LS Means</th>
<th>%95 CI for Difference</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drug A vs. Placebo</td>
<td>-0.1</td>
<td>(-0.14, -0.02)</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>Drug B vs. Placebo</td>
<td>-0.1</td>
<td>(-0.14, -0.03)</td>
<td>0.412</td>
</tr>
<tr>
<td>Drug A vs. Drug B</td>
<td>0.0</td>
<td>(-0.05, 0.06)</td>
<td>&gt;0.999</td>
</tr>
</tbody>
</table>

$^\dagger$N is the number of patients used in the ANOVA analysis.

Figure 4. A stack table produced by %glue

### A SIMPLE FRAMEWORK FOR CREATING STACK TABLES

As a clinical programmer, you often have to write SAS programs for a variety of tables to support clinical study reporting. During the development, many different concerns (or issues) will arise. The direct concerns for a table you may think of include: the layout structure, visual appearance, and content of the table. You may also have crosscutting concerns, such as adhering to the company SOP, meeting FDA submission requirements, having a consistent appearance for tables produced by different programmers, and so on. Some of those concerns often end up with scattered code amongst the various SAS programs, and/or tangled code within a particular SAS program. For example, scattering the code for a special visual appearance of footnotes in stack tables across a set of SAS programs will make the feature modification a substantial amount of effort. In this section, I give a simple framework for writing stack tables based on well-established software engineering principle of separation of concerns [3][4]. The programming guidelines for the stack table I propose is as follows:

- Separate content, or data from representation,
- Separate visual appearance from layout structure,
- Separate local visual appearance from global visual appearance, and
- Separate data-dependent visual appearance from data-independent visual appearance

Those guidelines also apply to regular table development. Why are those guidelines so important in the table programming? This is because the mass table generation is a tedious and time-consuming task involving lots of concerns in the clinical study analysis and reporting process. If those concerns can be managed locally, one at a time, at different levels, and in separated code segments, the table programs will be easier to write, understand, reuse, and modify. Based on the above programming guidelines, the steps for developing a stack table with \( n \) child tables can be described as follows:
1. Create \( n \) reporting datasets (such as \( ReportDSN_1, ReportDSN_2, \ldots, \text{and} \ ReportDSN_n \)) with Data and PROC steps. Make sure that every reporting dataset be sorted properly and ready for use without any further data manipulation to be done by Proc Report. The variables in the datasets should be associated with proper formats and labels if necessary. This step separates the data from its presentation.

2. Create style definitions with a Proc Template step. Proc Template provides a central place to control visual appearances of tables. The most important advantage of using Proc Template is that you have a mechanism to separate the code for table visual appearances from the code for the table layout structures so that you can encapsulate almost all global concerns for table visual appearances in a single Proc Template. This makes it very easy to modify the color, font, header, footer, and border in a stack table, which is the trickiest part in the table programming. Besides, the inheritance that Proc Template uses to organize style definitions lets you to classify style elements into different layers so that you can separate the higher-level visual appearance concern from lower-level one. Below is a typical pattern for the organization of the style definitions in the table programming:

```latex
PROC TEMPLATE;
DEFINE STYLE Styles.stack_table;
   PARENT Styles.RTF or any project/company-wise style definition;
...;
DEFINE STYLE styles.child_table;
   PARENT Styles.stack_table;
...;
DEFINE STYLE Styles.child_table;
   PARENT Styles.stack_table;
...;
...;
DEFINE STYLE Styles.child_table;
   PARENT Styles.stack_table;
...;
End;
Run;
```

With the code pattern above, the style definition for a stack table can inherit attributes from Styles.RTF, or a permanent project/company style definition you choose, and all child table style definitions can inherit attributes from the stack table style definition, thus minimizing the redundancy of the code for table visual appearances and maximizing the capability of your understanding, modifying and reusing those style definitions.

3. Create child tables with a series of Proc Report steps. Use Column statement to define the layout structure of a child table, and interweave it with the corresponding dataset, and style definition. Within the Proc Report for a child table, you can also adjust the local appearance of the child table on the fly, such as
   - Defining the special visual appearance for a column in a child table with Define variable/style= statement,
   - Creating data-dependent visual appearance with Call Define statement by using Compute Blocks,
   - Creating special visual effects on titles, header, and footnotes etc by embedding RTF control symbols and words with inline formatting commands such as \(^R/RTF"raw rtf code", ^{...}, ^S={...} \) [5][6][7].

The coding pattern can be summarized as follows:

```latex
ODS RTF file = "StackTableFile.rtf" startpage = no bodytitle
Style=styles.stack_table;

ODS RTF Style=Styles.Child_Table;
Proc Report Data=ReportDSN1 nowd ...;
   Column ...;
   Define ...;
```
4. Call %glue to create the stack table from the ODS RTF output file, that is

%glue(StackTableFile.rtf)

The framework provided above can be recapitulated with the following formula:

Stack Table = \[ \sum \text{Dataset} + \text{Proc Template} + \sum \text{Proc Report} + \%glue \]

The obvious advantages of this programming framework are

- Creating a stack table is simply equivalent to writing \( n \) regular child tables with the Proc Report steps.
- Almost all advanced Proc Report features can be used for the construction of stack tables, such as traffic lightings, images, hyperlinks, and so on,
- The code is much easier to understand, modify and reuse. Standard macros can be developed based on this framework.
- Little or no learning curve for those SAS programmers who know how to use the popular Proc Report procedure.
- Providing a uniform programming approach for table generations that can be shared by different sites within a company, or across different companies.

A COMPLETE EXAMPLE

In order for you to fully understand this table programming technique, I will show you the SAS code all together, and then we’ll go through it in pieces. Here is the entire program to generate the stack table in Figure 1:

```sas
/** Step 1: Set up Data */
data footer;
    length footer $3000;
    footer="^R/RTF"{\super \^86}"N is the number of patients used in the ANOVA analysis.";
    Output;
run;
proc format;
    value pvalfmt 0-<0.001='<0.001'
        0.001-0.999=[5.3]
        0.999<=1.000='>0.999'
        .='.';
run;
```
/* Step 2: Style Definitions */
proc template;
   define style styles.Stack_Table_Style;
      parent=styles.RTF;
      /* Control table appearance */
      style table from table /
         cellspacing=0 rules=all
         bordertopstyle=double
         borderbottomstyle=solid
         borderleftstyle=solid
         borderrightstyle=solid
         frame=box
         outputwidth=6in
      ;
      /* Control all column header appearance */
      style header from header /
         background=white
      ;
      /* Control stack table title appearance */
      style systemtitle from titlesandfooters/
         font_style=roman
         font_weight=bold
      ;
      /* Control data in the cell */
      style data from data/
         just=C
      ;
   end;

   define style styles.child_table1;
      parent=styles.Stack_Table_Style;
   end;

   define style styles.child_table2;
      parent=styles.Stack_Table_Style;
      /* Change the appearance of child table 2 */
      style table from table/
         bordertopstyle=solid
      ;
   end;

   define style styles.child_table3;
      parent=styles.Stack_Table_Style;
      /* Change the appearance of child table 3 */
      style table from table/
         bordertopstyle=solid
         borderbottomwidth=5
      ;
   end;
run;

/* Step 3: Weaving */
option nomprint nosymbolgen nomlogic nodate nonumber;
ods listing close;
ods rtf file = "c:\ex3.$rtf" startpage = no bodytitle
   style=styles.Stack_Table_Style;
ods escapechar = '^';
title1 "Analysis of Average Change From Baseline in YYY^{sub 1}(L)";
Datasets and formats are created for the stack table to be generated. In this program, as an example, a dataset for footnotes is created so that the footnotes will be the last child table of the stack table.

One style definition for the stack table and three for its child tables are created. The style definitions for child tables inherit all style attributes from the stack table definition which, in turn, inherits all style attributes from Styles.RTF. The Styles.RTF can be replaced with your own project-wise or company-wise permanent style definition(s) in real-life projects. Child table style definitions can have their own special attributes defined that will override their parent style attributes. For example, Styles.Child_table3 changes the bottom borderline width of the table frame to 5 pixel points with attribute borderbottomwidth=5. If you want to change the border line style to the
double line, you can add a new attribute borderbottomstyle=double in the table style element.

**``ODS RTF destination with startpage = no bodytitle, and style definition “Styles.Stack/Table” is created. The symbol “^” is defined as an escape character.**

**``Child tables with a series of Proc Report steps are created. Column statements are used to define the layout structures of individual child tables, and then are integrated with the datasets and style definitions in first two steps to produce the specific child tables. Since the style definition for each child table only controls the overall appearance, if you want to change the local appearance of the individual table, such as headings, columns, and data in the cells, you have to change them on the fly with The DEFINE variable/Style(column/header)= statements, Compute blocks with Call define statements, and/or inline formatting commands.**

**``Finally all child tables are combined together by macro %glue to produce the table shown in Figure 1.**

**CONCLUSION**

I hope that this paper has given you a powerful and flexible way of constructing a variety of RTF stack tables for your clinical studies. The techniques and codes described in this paper also show that the maximum flexibility in creating RTF stack tables can be achieved with the principle of the separation of concerns under the help of macro %glue, ODS RTF, Proc Template and Proc Report.

**DISCLAIMER:** The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Merck & Co. Inc.

**REFERENCES**

4. D.L.Parnas, on the criteria to be used in decomposing systems into modules. Communications of the ACM, 15(12): 1053-1058, December 1972
7. David Shannon, “To ODS RTF and Beyond”, SUGI 27, Amadeus Software Limited, UK
8. SAS-L communications

**ACKNOWLEDGMENTS**

The author wishes to thank Cindy Song, John Troxell, Susan Neidlinger, and Donna Usavage for their kind comments and constructive advice during the preparation of this paper.

**CONTACT INFORMATION**

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

Lei Zhang
Merck & Co., Inc.
RY34-A320 P.O. Box 2000
Rahway NJ 07065-0900
Work Phone: (732)-594-9865
Fax: (732)-594-6075
Email: Lei_Zhang4@Merck.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
Datasets used in the SAS programs:

data summary;
  input  trt $1-7 N 9-11 mean 13-16 sd 18-21
         lsmean 23-27 ci $29-41;
  datalines;
  Drug A   114 0.38 0.32 -0.04 (-0.08, 0.00)
  Drug B   115 0.37 0.28 -0.04 (-0.08, 0.01)
  Placebo 113 0.38 0.30  0.04 (-0.00, 0.08)
;
Run;

data analysis;
  input  comp $1-18 diff 20-24 ci $26-39 pvalue 41-46;
  datalines;
  Drug A vs. Placebo -0.08 (-0.14, -0.02) 0.0001
  Drug B vs. Placebo -0.08 (-0.14, -0.03) 0.4123
  Drug A vs. Drug B   0.00 (-0.05,  0.06) 0.9993
;
Run;