Paper 089-31

Constructing Stack Tables with Magic %GluePlus

(Another User-Defined Macro for Creating Stack Tables)

Lei Zhang, Celgene Corporation, George Li, Amgen Inc.

ABSTRACT

As a composite table, a stack table is made up of a varying number of child tables that are placed one on top of another without any gaps. Unlike a regular table that has a fixed number of columns throughout the rows, a typical stack table allows each of its child tables to have its own number of columns and headers and each can vary its size, layout and style separately. A stack table hence gives a much greater freedom and creativity to the data presentation than a regular table does.

Stack tables are commonly used in complex reports because of their effective integration of statistical results from multiple sources and flexible presentations of various aspects of concerns. In this paper, we extend the method introduced in [1] and present another macro called %GluePlus that can assemble multiple separate tables in a RTF file into a single stack table. The new macro is implemented by taking advantage of the power of Microsoft scripting technology via the SAS unnamed pipe mechanism. Based on the %GluePlus, an improved programming framework is provided for creating all kinds of stack tables systematically with a series of Data Steps, Proc Template and Proc Report Steps. A complete example is presented at the end to demonstrate the powerfulness and flexibility of the technique described in this paper.

INTRODUCTION

A stack table is a composite table that consists of an arbitrary number of child tables that are seamlessly placed one on top of another. It is widely used in complex reports because of their effective integrated presentations of various summary data. Before we discuss about how to construct a stack table, let's take a look at Figure 1, a stack table with three child tables.

Analysis of Average Change From Baseline in T_4 (µg/dL)
Treatment Period

Child table 1

		Baseline		Average Change from Baseline	
Treatment Group	**N†**	**Mean**	**SD**	**LS Mean**	**95 % CI for LS Mean**
Drug A	114	3.8	0.32	-0.01	(-0.04, 0.00)
Drug B	115	3.7	0.28	-0.03	(-0.08, 0.01)
Placebo	113	3.8	0.3	0.02	(-0.00, 0.08)

Child table 2

Between-treatment Comparisons from ANOVA Model

Comparison	Difference in LS Means	%95 CI for Difference	p-value
Drug A vs. Drug B	-0.07	(-0.14, -0.02)	<0.001
Drug B vs. Placebo	-0.10	(-0.14, -0.03)	0.712
Drug A vs. Placebo	0.01	(-0.05, 0.06)	0.999

Child table 3

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

Figure 1 - A sample stack table from a clinical study report

This stack table is a mock table for free T_4 lab test analysis in a clinical study report. Unlike a regular table that has a fixed number of columns throughout the rows, the stack table has three unique features. First, it is made up of three regular child tables (as indicated in Figure 1). Second, all the child tables are seamlessly integrated and there is no gap between them. Third, each child table has different number of columns and headers with different size, layout and style. (In fact, a regular table is a degenerate stack table.) The real power behind a stack table is that each of its child tables can vary separately in regard to its size, layout and style. It therefore gives a great freedom to data presentation that a regular table just cannot meet or beat even with complex column spanning and sizing techniques.

A great deal of effort has been made to create stack tables quickly and easily within SAS programs [1][2]. In this paper, I extend the method introduced in [1] and present a new macro called %GluePlus that can create a stack table in RTF format from a series of separate regular tables generated either by Proc Report (under ODS RTF settings in SAS version 8 or 9), or by any third party tools. Based on the `%GluePlus`, I further provide a modified programming framework for constructing stack tables by the combination of Data steps, Proc Report, and Proc Template. A complete code example with detailed explanation is also provided at the end of this paper.

CREATING A SINGLE STACK TABLE FROM MULTIPLE REGULAR TABLES

As a powerful standard report-writing tool in the SAS System, Proc Report has been widely used by many pharmaceutical and biotechnology companies in their regulatory submission process. Under the ODS RTF settings, we can use multiple Proc Report steps to create a sequence of regular tables in a RTF file. Listing 1 is a small program that shows how to create two tables in a file (TwoTables.RTF) under ODS RTF settings.

```
/* Create first table */
option nomprint nosymbolgen nomlogic nodate nonumber;
ods rtf file = "c:\TwoTables.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;

/* Create second table */
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 '%95 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.";
run;
ods rtf close;
```

Listing 1 - A SAS program that creates two separate tables in a RTF file

The tables with a noticeable gap in the TwoTables.RTF are shown in Figure 2.

Analysis of Average Change From Baseline in T_4 (µg/dL)
Treatment Period

		Baseline		Average Change from Baseline	
Treatment Group	**N***	**Mean**	**SD**	**LS Mean**	**95 % CI for LS Mean**
Drug A	114	3.8	0.32	-0.01	(-0.04, 0.00)
Drug B	115	3.7	0.28	-0.03	(-0.08, 0.01)
Placebo	113	3.8	0.3	0.02	(-0.00, 0.08)

Gap between two tables

Between-treatment Comparisons from ANOVA Model			
Comparison	**Difference in LS Means**	**%95 CI for Difference**	**p-value**
Drug A vs. Drug B	-0.07	(-0.14, -0.02)	0.0001
Drug B vs. Placebo	-0.10	(-0.14, -0.03)	0.7123
Drug A vs. Placebo	0.01	(-0.05, 0.06)	0.9993

**** N is the number of patients used in the ANOVA model analysis.***

Figure 2. - Two separate regular tables generated by two Proc Report steps under ODS RTF settings

The macro `%Glue` introduced by [1] can be used to remove the gap between the two tables if the file is generated under SAS 9 environment. However, %Glue requires that all RTF tables generated by Proc Report use ODS RTF `startpage=no` and `bodytitle` options. In this paper, we provide a new macro called `%GluePlus` that can remove any gaps between the tables contained in an RTF or WORD file without the constraints the %Glue has. The new macro is a small macro wrapper that will hand over a RTF file to a VBScript program (called GluePlus.vbs), which will build a stack table with titles and footnotes by properly concatenating all the tables in the file. The GlusPLus.VBS is shown in the Listing 2.

```
'  GluePlus.vbs
'  Purpose: Construct a stack table by removing the gaps between child tables
'  in an RTF file
'

      ❶On Error Resume Next
       'Attempt to get an existing running copy of Word
       Set objWord = GetObject(, "Word.Application")
       'If error occurred, then couldn't find Word, create new instance
       If Err Then
              Err.Clear
              Set objWord = CreateObject("Word.Application")
       End If
      objWord.visible = 0

       ' Open an RTF file
```

```
❷objWord.Documents.Open(Wscript.Arguments(0))

❸Set ps = objWord.ActiveDocument.Paragraphs

' Find the number of titles
 idx = 1
Do While idx <= ps.count
         set p =ps(idx)
        If not (p.range.Information(12) = 0) Then
               Exit do
        End If
        idx=idx +1
Loop

 ntitle=idx-1

 ' Find the number of footnotes
 idx = ps.count
Do While idx > 0
         set p =ps(idx)
        If not (p.range.Information(12) = 0) Then
               Exit do
        End If
        idx=idx - 1
Loop

 nfnote=ps.count - idx

 ' Find the number of gaps or paragraphs outside of child tables
 npara = 0
 idx = 1
Do While idx <=ps.count
         set p =ps(idx)
        If p.range.Information(12) = 0 Then
               npara=npara + 1
        End If
        idx=idx + 1
Loop


 ' Get the number of gaps to be removed
 ndel=npara - (ntitle + nfnote)

❹' Remove the gaps between child tables
 If ndel > 0 then
        idx = 1
        For Each p in ps
               If p.range.Information(12) = 0 Then
                      If (idx > ntitle) and ndel > 0 then
                             p.range.Delete
                             ndel = ndel - 1
                      End if
               End If
               idx=idx + 1
        Next
End if

 ❺' Save the revised RTF file
objWord.ActiveDocument.close(-1)
 ' destroy the Word automation server instance
 objWord = Nothing
```

Listing 2 - `GluePlus.vbs`

The `GluePLus.vbs` shown in Listing 2 is executed by `cscript.exe`, the command-line Window Script Host (WSH) scripting engine that is freely available on any Windows platforms [3]. By taking advantage of the power of Microsoft Word OLE automation server, the script program does the trick of removing the table gaps with following steps:

❶ Launch Microsoft Word OLE automation server and make it invisible.

❷ Open the RTF file that contains the child tables, or the building blocks of a stack table.

❸ Find out the number of titles and footnotes the stack table intends to have, and compute the number of gaps that have to be removed.

❹ Delete the gaps between the child tables.

❺ Finally save the updated file, and exit the Microsoft Word.

With `GluePLus.VBS` and `cscript.exe`, the implementation of `%GluePlus` is straightforward. The macro takes an RTF file as a required parameter and then invokes `cscript.exe` via an unnamed SAS pipe to execute the `GluePlus.vbs` for the removal of the gaps between tables in the file. This tiny macro is shown in Listing 3.

```
%Macro GluePlus(
    RTFfile    /* Full path of the RTF or Word file to be processed */
);

/* Create an unnamed pipe with the given RTF file*/
%local pipe;
%let pipe=cscript VBSriptLocationPath\GluePlus.VBS &RTFfile //nologo;

filename _glue_ PIPE %sysfunc(quote(&pipe)) console=min;

/* Remove the gaps in the RTF file via the unnamed pipe */
data _null_;
    infile _glue_ pad missover;
    input; put _infile_;
run;

%Mend;
```

Listing 3 - %GluePlus

With the %GluePlus, creating a stack table from multiple RTF tables is never been easier. For example, to have a stack table from file TwoTables.RTF, use following macro call:

```
%GluePlus(c:\TwoTables.RTF)
```

The stack table thus generated is shown in Figure 3.

Analysis of Average Change From Baseline in T_4 (µg/dL)
Treatment Period

		Baseline		Average Change from Baseline	
Treatment Group	**N***	**Mean**	**SD**	**LS Mean**	**95 % CI for LS Mean**
Drug A	114	3.8	0.32	-0.01	(-0.04, 0.00)
Drug B		3.7	0.28	-0.03	(-0.08, 0.01)
Placebo		3.8	0.3	0.02	(-0.00, 0.08)
Between-treatment Comparisons from ANOVA Model					
Comparison	**Difference in LS Means**	**%95 CI for Difference**		**p-value**	
Drug A vs. Drug B	-0.07	(-0.14, -0.02)		0.0001	
Drug B vs. Placebo	-0.10	(-0.14, -0.03)		0.7123	
Drug A vs. Placebo	0.01	(-0.05, 0.06)		0.9993	

No gap between two child tables now

**** N is the number of patients used in the ANOVA model analysis.***

Figure 3. - A stack table produced by %GluePlus

TABLE PROGRAMMING WITH PRINCIPLE OF SEPARATION OF CONCERNS

As a table integration tool, `%GluePlus` paves the road for the ultimate generation of a stack table from multiple RTF tables; however, creating a stack table usually takes significant time and effort. Many concerns (or issues) will arise during the development process, especially when you write stack table programs for complex clinical analysis reports. The direct concerns you may have include: stack table contents, titles, footnotes, font, and layout. You may also have crosscutting concerns, such as having a consistent look and feel across the child tables, following the same stack table style as other programs have, adhering to the company SOP for the table reports, etc. Those concerns often lead to tangled code within a particular SAS program, or scattered code among multiple SAS programs. For example, some special visual requirement of footnotes for a set of stack tables may cause the code pieces to scatter across a group of SAS programs, which will make the future code modification a substantial amount of effort. Based on the well-established software engineering principle of separation of concerns [4][5], We therefore propose following programming guidelines for stack table construction with SAS language:

1. *Separate data from representation,*
2. *Separate visual appearance from table layout,*
3. *Separate local visual appearance from global visual appearance, and*
4. *Separate data-dependent visual appearance from data-independent visual appearance*

The above guidelines actually apply to both regular table and stack table development. The main purpose is to manage the design and programming issues of stack table locally, one at a time, at different organized levels, and in separated code segments. Therefore, programs for stack tables will be much easier to write, understand, reuse, and modify.

A PROGRAMMING FRAMEWORK FOR STACK TABLES

Based on the above programming guidelines, a programming workframe for a stack table with *n* child tables are presented. The workframe has four steps:

Step 1. Create style definitions with 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 visual appearances from the code for table layout so that you can encapsulate almost all global concerns about visual appearances in a single place. This makes it very easy to modify the color, font, border and other visual elements across a stack table, or in a group of stack tables, which often is the trickiest part in the table programming. Besides, the inheritance mechanism 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. The typical code pattern for the organization of the style definitions will look like this:

```
PROC TEMPLATE;
        DEFINE STYLE  styles.stack_table;
                PARENT  Styles.RTF or any project/company-wise style definition;
        …;
        DEFINE STYLE styles.child_table1;
                PARENT  Styles.stack_table;
        …;
        DEFINE STYLE Styles.child_table2;
                PARENT  Styles.stack_table;
        …;
        ...
        DEFINE STYLE Styles.child_tablen;
                PARENT Styles.stack_table;
        …;
        End;
Run;
```

With the code pattern, the style definition for a stack table (`styles.stack_table`) inherits attributes from `Styles.RTF`, (or a permanent project/company style definition you choose), and all the style definitions for child tables (`styles.child_table`$_1$, `styles.child_table`$_2$, ..., `styles.child_table`$_n$) inherit attributes from `styles.stack_table`, thus minimizing the redundancy of the code for table visual appearances and maximizing the capability of understanding, modifying and reusing those style definitions. The style definitions should be handled in he same way as SAS formats. If a set of style definitions is shared by a group of stack tables, it is better to put them in a executable separated file, so that they can be run once and referenced many times by many other programs.

Step 2. Create *n* reporting datasets (such as *ReportDSN*$_1$, *ReportDSN*$_2$, ..., and *ReportDSN*$_n$) with Data steps and PROC steps. Make sure that each reporting dataset be ready for the table generation by Proc Report. This step separates the data from representation.

Step 3. Create child tables with a series of Proc Report steps. Use Column statement to define the layout of each child table, and then interweave it with the corresponding dataset, and style definition. Within Proc Report, you can also change the local facade of a child table on the fly with some ODS style options, such as

1. Defining the special visual appearance for a column in a child table with `Define` *`variable`*`/style=` statement,

2. Creating data-dependent visual appearance with `Call Define` statement , and

3. Creating special visual effects on titles, headers, cells and footnotes by embedding RTF control symbols and words by inline formatting commands such as `^R/RTF"`*`raw rtf code`*`", ^{…}`, and `^S={...}`. For more information, please see [6][7][8].

The linear coding pattern can be summarized as follows:

```
ODS RTF file = "StackTable.RTF" bodytitle Style=styles.stack_table;

ODS RTF Style=Styles.Child_Table1
Proc Report Data=ReportDSN1 nowd ...;
     Column  ...;
     Define  ...;
     ...;
Run;
```

```
ODS RTF Style=Styles.Child_Table2
Proc Report Data=ReportDSN2 nowd ...;
     Column  ...;
     Define  ...;
     ...;
Run;

...

ODS RTF Style=Styles.Child_Tablen
Proc Report Data=ReportDSNn  nowd ...;
     Column  ...;
     Define  ...;
     ...;
Run;

ODS RTS Close;
```

Step 4. Call `%GluePlus` to create the stack table from the RTF output file. For example,

```
%GluePlus(StackTable.RTF)
```

The framework provided above can be recapitulated with following formula:

Stack Table = [Proc Template (∑Style)] + ∑Dataset + ∑(Proc Report) + %GluePlus

Note 1: if you have defined and stored style definitions in a central repository, it is optional to create style definitions with Proc Template in a stack table program. Doing so often makes stack table programs much shorter, easy to understand and reuse.

Note 2: Apart from concatenating child tables generated by Proc Report under ODS RTF settings, %GluePlus can be used to construct stack tables from the Word tables generated by other applications.

Note 3: If you need a stack table in PDF format, you can convert it by running an RTF-to-PDF converter right after `%GluePlus` generates the stack table.

BENEFITS IN USING THE PROGRAMMING FRAMEWORK

There are several obvious benefits from this programming framework:

1. You don't have to create a stack table from scratch. Creating a stack table is simply equivalent to creating *n* regular child tables.

2. Almost all advanced Proc Report features, such as traffic lightings, images, hyperlinks, and bookmark, can be implemented in the stack tables with this approach.

3. Standard macros can be developed for a particular group of stack tables. The code will be much easier to write, modify and reuse.

4. Little or no learning curve for those SAS programmers who know the popular Proc Report procedure.

5. The programming technique can be easily shared by different sites within an organization, or transferred across different organizations.

A COMPLETE EXAMPLE

In order for you to fully understand the programming technique for stack tables, we will show you how to generate the stack table in Figure 1. The example code is divided into two parts. The first part is `style.sas` in Listing 4, which defines all the style definitions used by the stack table.

```
/** Step 1: Style Definitions */
❶Proc template;
      define style styles.Stack_Table_Style;
      parent=styles.RTF;
      /* Control table appearance */
```

```
      style table from table /
            cellspacing=0 rules=all
            /* border styles are only available in SAS 9 */
            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;
```

Listing 4. - The program that defines all the style definitions for the stack table in Figure 1.

The second is `StackTable.SAS`, which implements the stack table.

```
/** Step 2: Set up Data */
❷data 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 3: Weaving */
option nomprint nosymbolgen nomlogic nodate nonumber;
```

```
ods listing close;
❸ods rtf file = "c:\StackTable.rtf" bodytitle style=styles.Stack_Table_Style;
ods escapechar = '^';

title1 "Analysis of Average Change From Baseline in YYY^{sub 1}(L)";
title2 "Treatment Period";

❹ods rtf style=styles.child_table1;
proc report nowindows missing data=summary;

  column trt n ('Baseline' mean sd)
         ('Average Change from Baseline' lsmean ci);
  define trt/display 'Treatment Group'
               style(column)=[cellwidth=1.8in];
  define n/display "N^R/RTF""{\super \'86}""";
  define mean/display 'Mean';
  define sd/display 'SD';
  define lsmean/display 'LS Mean';
  define ci/display '95 % CI for LS Mean';
run;

ods rtf style=styles.child_table2_style;
proc report nowindows missing data=analysis;

  column ('^{\~} ' ('Between-treatment Comparisons from ANOVA Model' comp diff ci
pvalue));
  define comp/display 'Comparison'
            style(column)=[cellwidth=1.27in];
  define diff/display 'Difference in LS Means' format=5.2;
  define ci/display '%95 CI for Difference';
  define pvalue/display 'p-value' format=pvalfmt.;
  compute pvalue;
       if pvalue < 0.001 then do;
            call define (_row_, 'style',
            'style=[background=yellow]');
       end;
  endcomp;
run;

ods rtf style=styles.child_table3;
proc report nowindows missing noheader data=footer ;

  column footer;
  define footer/display ' '
     style={font_style=roman font_weight=bold just=l};
run;

ods rtf close;
ods listing;

❺%GluePlus(C:\StackTable.rtf)
```

Listing 5. - The program for implementing the stack table in Figure 1.

That's it. Let's see how the individual code pieces work.

❶ Create the program `Style.SAS` that defines one style definition for the whole stack table and three for its child tables. 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 default `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 borderline style to the double line, you can add a new attribute `borderbottomstyle=double` in the table style element. (Note: both `borderbottomwidth=` and `borderbottomstyle=` are new style elements in SAS 9.)

❷Create datasets and formats 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.

❸Create ODS RTF destination with style definition "*Styles.Stack_Table*". The symbol "^" is defined as an escape character.

❹Create child tables with a series of Proc Report steps. 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 can change them on the fly with The `DEFINE` *`variable`*`/Style(column/header)=` statements, `Call define` statements, or inline formatting commands.

❺Finally combine all child tables together by the call to %GluePlus, which forms the stack table shown in Figure 1.

CONCLUSION

I hope that this paper has given you a simple way to construct all kinds of stack tables with SAS programs. The technique described in this paper represents a new frontier for SAS report programming. Furthermore, it represents an opportunity for SAS programmers looking for ways to enhance and extend the functionality of their existing table macros. Imagine if you could "LEGOize" a various type of tables with this magic "glue" to form a stack table. You might just find yourself creating the tables you never thought possible.

DISCLAIMER: The contents of this paper are the work of the authors and do not necessarily represent the opinions, recommendations, or practices of Celgene Corporation and Amgen Inc.

REFERENCES

1. Zhang, Lei. 2005. "Constructing Stack Tables With Proc Report and ODS RTF," *Proceedings of PharmaSUG '05*, Phoenix, AZ.

2. Peszek, I., Song, C., Kuznetsova, O. 1999. "Producing Tabular Reports in SAS in the Form of Word Tables," *Proceedings of PharamaSUG '99*, New Orleans, LA.

3. Microsoft, Inc. "VBScript Primer", http://www.microsoft.com/technet/scriptcenter/guide.

4. Dijkstra, E. W. 1976. *A Discipline of Programming*. Prentice Hall, Englewood Cliffs, NJ.

5. Parnas, D.L. 1972. "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, 15(12): 1053-1058.

6. David Shannon, "To ODS RTF and Beyond," *Proceedings of the Twenty-seventh Annual SAS Users Group International Conference*, Orlando, FL.

7. Gupta, Sunil K. 2004. "Using Styles and Templates to Customize SAS ODS Output," *Proceedings of the Twenty-ninth Annual SAS Users Group International Conference*, Montreal, Canada.

8. Haworth, Lauren. 2004. "SAS with Style: Creating your own ODS Style Template for RTF Output", *Proceedings of the Twenty-ninth Annual SAS Users Group International Conference*, Montreal, Canada.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact authors at:

Lei Zhang
Celgene Corporation.
86 Morris Avenue
Summit, NJ 07901
Phone: (908) 673-9000

George Li
Amgen Inc.
One Amgen Center Drive
Thousand Oaks, CA 91320
Phone: (805)-447-1000

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 22-27 ci $29-41;
datalines;
Drug A  114 3.8 0.32 -0.01  (-0.04, 0.00)
Drug B  115 3.7 0.28 -0.03  (-0.08, 0.01)
Placebo 113 3.8 0.30  0.02  (-0.00, 0.08)
;
Run;

data analysis;
input  comp $1-18 diff 20-24 ci $26-39 pvalue 41-46;
datalines;
Drug A vs. Drug B  -0.07 (-0.14, -0.02) 0.0001
Drug B vs. Placebo -0.10 (-0.14, -0.03) 0.7123
Drug A vs. Placebo  0.01 (-0.05,  0.06) 0.9993
;
Run;
```