Have it Both Ways: Macros that Produce Publication-Quality Tables and Stand-alone Code

Linda Collins, PharmaStat LLC, Newark, CA
Lisa Brooks, VaxGen Inc., Brisbane, CA
Michael Rea, VaxGen, Inc., Brisbane, CA
Alan Hopkins, PharmaStat LLC, Newark, CA

ABSTRACT

There's an intrinsic tension between setting up an elegant, modular programming system and writing simple, easy-to-understand programs. Macro libraries have enormous value as a production tool; they allow common applications to be written, validated, and reused easily. However, they often contain a great deal of obscure code designed to produce 'pretty' results. For some audiences – such as a statistician or regulatory reviewer who wants to verify that the desired procedures were coded correctly – a simple, straightforward program is preferable. The ideal program for this audience contains only the procedures necessary to produce statistical results (see CDISC Analysis Data Model Version 2.0).

In this paper, we present a technique for satisfying both needs. The framework is an application macro library that runs common SAS® statistical procedures and produces a finished report. A program run generates both a publication-quality statistical table and a simple program containing only the data pre-processing and statistical procedures. The generated code is executable as a portable, stand-alone program, which does not require access to the original macro library. The simplified code can be used for debugging, to verify correct use of the macro library, and as a record of the steps and procedures used to program the table.

INTRODUCTION

What if you could generate a simple, streamlined, standalone program automatically in the same program run that produces the statistical table? This is not, in fact, difficult. By defining an output destination for resolved macro code, we can generate such a program automatically. A library of standard macros can make use of a simple set of conventions to ensure that the resolved code contains appropriate labeling and display of the statistics generated. A small amount of post-processing aids with readability.

DRUG REGULATORY CONTEXT

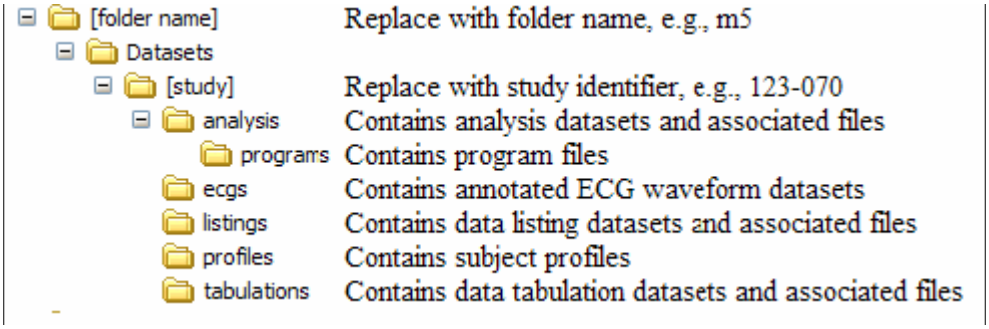
Programming activities for analysis of clinical data are inextricably linked to Good Statistical Practice (GSP). Use of good statistical practices lend greatly to credibility of the results of clinical studies. It is hard to find any single reference that explicitly defines GSP. ICH E9 (1) discusses statistical principles in drug development and provides a basis for good statistical practice from a drug regulatory perspective. The majority of the E9 document stresses GSP consists of good science (minimizing bias and maximizing precision of estimates of treatment effect) and validity and integrity (prespecified analysis and integrity of data and statistical software). The guidance reminds us that there is more to GSP than just good study design and analysis procedures:

> "The credibility of the numerical results of the analysis depends on the quality and validity of the methods and software (both internally and externally written) used both for data management (data entry, storage, verification, correction, and retrieval) and for processing the data statistically. Data management activities should therefore be based on thorough and effective standard operating procedures. The computer software used for data management and statistical analysis should be reliable, and documentation of appropriate software testing procedures should be available."

In addition to ICH E9, there are multiple drivers for a new approach to statistical practice that emphasizes documented reproducible research: CDISC data standards, the ADaM guidance version 2.0, 21 CFR Part 11, FDA guidances for electronic NDA/eCTD submissions, and ICH E3. The ADaM guidance provides metadata standards for data and analyses. This enables statistical reviewers to understand, replicate, explore, confirm, and reuse the data and analyses. The goal is clear, unambiguous communication of decisions, analysis and results

We interpret good statistical practices as a reproducible, transparent, efficient and validated approach to designing studies and to acquiring, analyzing, and interpreting clinical data. Reproducible research depends upon process transparency and also provides auditability of the statistical analysis.

Examination of the FDA electronic submission guidances defines what FDA expects to receive: multiple types of data files, documentation, and programs – components of a whole statistical environment. The specifications for organizing study datasets and their associated files in folders are summarized below. (ref 1)



Since analysis programs are recognized as part of good documentation, statistical programmers have found themselves serving two masters. Internal medical writers require publication ready tables for insertion into clinical trial reports and regulatory statistical reviewers for simple easy to read programs that produce results.

It is a common practice for some companies to handle this issue by writing a separate, stripped-down version of key table programs just for regulatory submission. This is less than optimal since the linkage in logic between the publication-ready programs and the "readable" version must be confirmed manually. We have implemented a technique which creates platform independent SAS code which is easy to read and is produced by the same program as the publication-ready tables. We develop this technique in detail in the following sections.

DISCUSSION OF TECHNIQUE

BASICS OF THE MPRINT/MFILE TECHNIQUE

The technique described here uses the SAS options MPRINT and MFILE to generate a standalone program at the same time the publishable report is run. These options write the relevant sections of resolved macro code to an external file. Since the lines written by MPRINT are resolved macro code, the resulting program can be executed without needing to reference an external macro library. By toggling MPRINT on and off at appropriate points, programmers are able to control which sections of the program are written. This allows the standalone program to omit irrelevant sections of code.

Most SAS programmers are familiar with the option MPRINT, which causes resolved macro statements to appear in the SAS log. Consider the following macro:

```
* ------------------------------------ *;
* Test macro                            ;
* ------------------------------------ *;
options mprint ;
%macro mymac ( iterate =  ) ;
  data testfile ;
     do i = 1 to &iterate ;
         a = 'Test variable' ;
         output ;
     end ;
  run ;
%mend ;
%mymac ( iterate = 3 ) ;
```

The log of this program run would look like:

```
1          * ------------------------------------ *;
2          * Test macro                           ;
3          * ------------------------------------ *;
4          options mprint ;
5
```

```
6          %macro mymac ( iterate =  ) ;
7
8            data testfile ;
9               do i = 1 to &iterate ;
10                  a = 'Test variable' ;
11                  output ;
12              end ;
13           run ;
14
15         %mend ;
16
17         %mymac ( iterate = 3 ) ;
MPRINT(MYMAC):   data testfile ;
MPRINT(MYMAC):   do i = 1 to 3 ;
MPRINT(MYMAC):   a = 'Test variable' ;
MPRINT(MYMAC):   output ;
MPRINT(MYMAC):   end ;
MPRINT(MYMAC):   run ;

NOTE: The data set WORK.TESTFILE has 3 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.01 seconds
```

In theory, one could reconstruct a complete program from the log by reading in the SAS log, selecting lines prefaced with 'MPRINT' and deleting the section before the colon. In practice, however, the MFILE option does this in a far cleaner way, by allowing the MPRINT lines to be redirected to a separate file.

By adding the FILENAME MPRINT statement to the previous program, we can cause all the MPRINT lines to be directed to the file 'mymac2.tmp':

```
* ------------------------------------ *;
* Test macro                            ;
* ------------------------------------ *;
filename mprint "mymac2.tmp";
options mprint mfile;

%macro mymac ( iterate =  ) ;
  * -------------------------------- ;
  * This is a test of MPRINT and MFILE ;
  * -------------------------------- ;
  data testfile ;
     do i = 1 to &iterate ;
         a = 'Test variable' ;
         output ;
     end ;
  run ;
%mend ;
%mymac ( iterate = 3 ) ;
```

The file 'mymac2.tmp' would look like:

```
* -------------------------------- ;
* This is a test of MPRINT and MFILE ;
* -------------------------------- ;
data testfile ;
do i = 1 to 3 ;
a = 'Test variable' ;
output ;
end ;
run ;
```

By toggling MPRINT on and off, we can control what goes into the '.tmp' file. The section shown in gray will be executed, but will not appear in the MPRINT file.

```
* ------------------------------------ *;
* Test macro                            ;
* ------------------------------------ *;
filename mprint "mymac3.tmp";
options mprint mfile;

%macro mymac ( iterate =  ) ;
  * --------------------------------- ;
  * This is a test of MPRINT and MFILE ;
  * We want this code to be captured   ;
  * --------------------------------- ;
  data testa ;
     do i = 1 to &iterate ;
         a = 'Test variable' ;
         output ;
     end ;
  run ;

  options nomprint ;
  * ------------------------------------- ;
  * This is a test of MPRINT and MFILE       ;
  * We do NOT want this code to be captured  ;
  * ------------------------------------- ;
  data testb ;
     do i = 1 to &iterate ;
         a = 'Test variable' ;
         output ;
     end ;
  run ;

  options mprint ;
  * ------------------------------------- ;
  * Now, we want this code to be captured   ;
  * ------------------------------------- ;
  data testc ;
     do i = 1 to &iterate ;
         a = 'Test variable' ;
         output ;
     end ;
  run ;
%mend ;

%mymac ( iterate = 3 ) ;
```

The file 'mymac3.tmp' is shown below. Note that the section of code where MPRINT was set off does not appear in this file.

```
* --------------------------------- ;
* This is a test of MPRINT and MFILE ;
* We want this code to be captured ;
* --------------------------------- ;
data testa ;
do i = 1 to 3 ;
a = 'Test variable' ;
output ;
end ;
run ;
options nomprint ;
* ------------------------------------ ;
```

```
* Now, we want this code to be captured ;
* ------------------------------------ ;
data testc ;
do i = 1 to 3 ;
a = 'Test variable' ;
output ;
end ;
run ;
```

USING THE MPRINT/MFILE TECHNIQUE TO MAKE A PROGRAM GENERATOR

The MPRINT/MFILE technique will reliably capture all resolved macro code to an external file, without extraneous log messages. However, a couple of refinements are needed to make this process into a robust code generator. We will handle the refinements by executing application programs from within a macro, GENSAS. The GENSAS macro is an alternative to executing the program directly. It does not require any revisions to the application program itself. It will set up the MFILE and MPRINT options, %include the application program, and perform some final cleanup on the generated program.

Requirement / Limitation	Refinement
MFILE will only capture code that is executed within a macro. This could leave out any statements in the application program that are executed in open code.	GENSAS macro %includes the application program thus executing all code within a macro.
Execution of the generated program code may overwrite outputs from the application program run.	GENSAS macro writes the generated program to a separate folder. When executed, this program will write its output to the local folder.
The resolved code is difficult to read if there are complex conditions such as nested IF statements.	GENSAS macro reads in and rewrites the generated code, inserting indentation as needed for readability.
Code containing explicit references to the execution environment or directory path, such as *libnames* or *filenames,* cannot easily be run in another environment.	References to external data locations (*librefs*) should be defined in an AUTOEXEC.SAS file. If the program is moved to a different location for execution, the AUTOEXEC.SAS is the only file that requires modification.

Figure 1 displays a typical schema for producing publication ready output, without attempting to generate a standalone program.

Fig 1: Schema of a Typical Table-producing Application Program

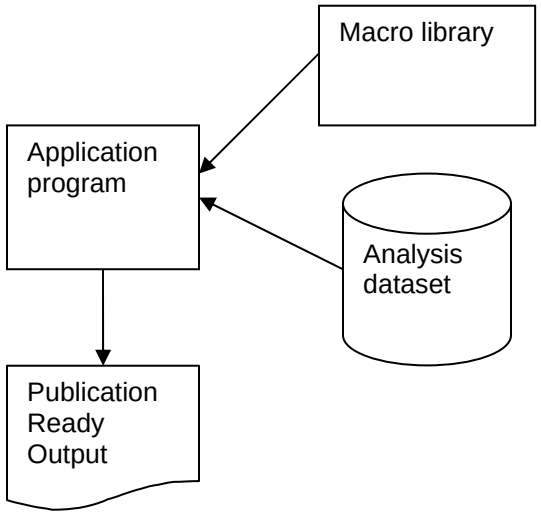
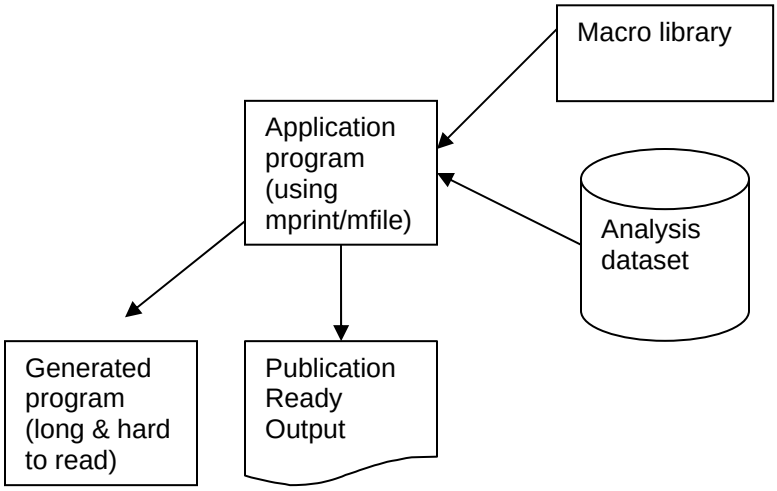


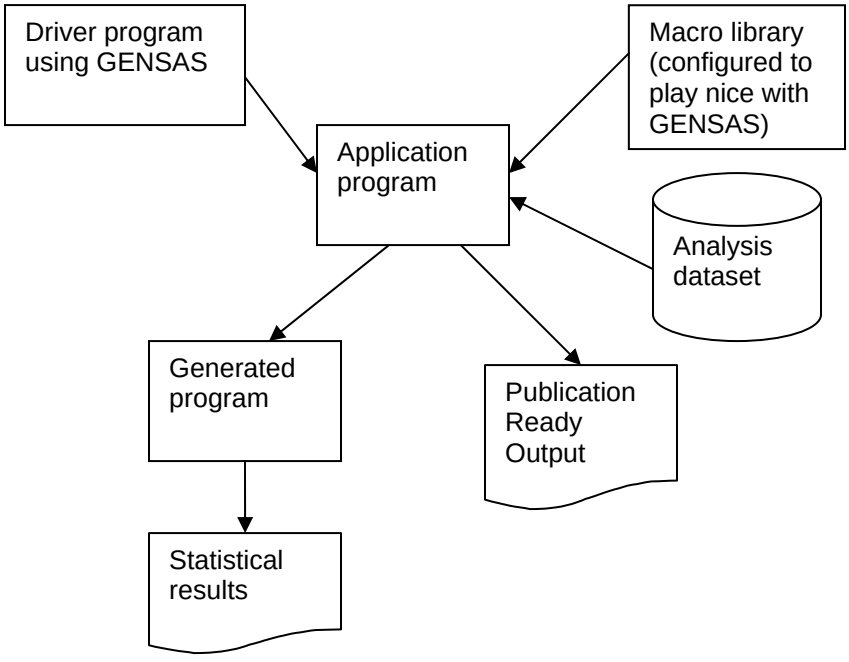
Figure 2 shows a modified application program In this schema we use the MPRINT/MFILE technique to generate a standalone program. This method has a minor problem in that it will generate all code executed with mprint on. That means that all code executed in the macro library will show up in the generated program. Unless there is a way to turn mprint on and off in the macro library we are stuck with a long and difficult to read generated program.

Fig 2: Schema of a Table Program, Modified to produce a Generated Program



Let's take a look at the revised schema for programs executed through GENSAS. Figure 3 displays the use of a driver program that uses the GENSAS macro and a modified Macro library that is configured to toggle mprint on and off at strategic locations based on a macro variable in GENSAS (&_codegen):

Fig 3: Schema of an Application Program executed through GENSAS



The GENSAS macro is an alternative technique that can be used for the subset of programs that need standalone code generated. The GENSAS macro sets a switch that controls whether any of the special processing within the macros should be turned on. This special processing consists of turning MPRINT on and off at the appropriate points in the program, and writing PROC PRINT steps that will be executed when the standalone program is run.

By using GENSAS, the application program does not need to be written with special commands to produce a standalone program. The application program can be run in the normal mode, without being invoked by GENSAS. In this mode, none of the special processing for GENSAS will be done. The macro code will not override any of the user's SAS options such as MPRINT. This is important for debugging purposes when it may be necessary to use SAS options to control the type of diagnostic information that appears in the log.

The GENSAS technique allows the user to decide at any time which application programs will require generated programs.

ANATOMY OF A TYPICAL APPLICATION PROGRAM USING MACROS

In an environment where a common macro library is used, application programs typically contain a minimum of native SAS code. The SAS code that is used is generally limited to a small amount of data subsetting, and possibly some data prep to get ready for the macro calls.

Consider a table like the following:

Fig. 1: Table of Change From Baseline by Lab Test and Time Point

LABTEST: Serum Glucose (mg/dl)		Treatment A		Treatment B		P-value
		Actual Value	Change from Baseline	Actual Value	Change from Baseline	Treatment A vs B
Lab Value						
Time Point 0	N	657		651		
	Mean	227.9		224.1		
	Std Dev	105.1		104.6		
	Median	228.0		225.0		
Time Point 1	N	657	657	655	651	
	Mean	217.4	-10.5	215.6	-8.9	0.783
	Std Dev	100.1	107.7	100.7	107.4	
	Median	203.0	-12.0	210.0	-12.0	
Time Point 2	N	657	657	654	650	
	Mean	207.7	-20.1	207.9	-16.5	0.548
	Std Dev	95.0	110.1	94.2	110.3	
	Median	194.0	-19.0	192.5	-18.5	

A macro library is a logical way to handle generating the descriptive and inferential statistics, and then reformat them into a readable presentation. A typical analysis file structure for this type of data would have one record per subject, test, and time point. For example:

Fig. 2: Input Data for Table of Change From Baseline

```
usubjid  txgrp          labtest              visit        labval  baseline  labchg

    1       2     Serum Glucose (mg/dl)   Time Point 0      93       93         .
            2     Serum Glucose (mg/dl)   Time Point 1     127       93        34
            2     Serum Glucose (mg/dl)   Time Point 2     130       93        37

    2       1     Serum Glucose (mg/dl)   Time Point 0     140      140         .
            1     Serum Glucose (mg/dl)   Time Point 1     205      140        65
            1     Serum Glucose (mg/dl)   Time Point 2     192      140        52
```

(Note that the actual file contains many other lab tests as well.)

The macro library we have available will do the descriptive statistics using PROC UNIVARIATE and the inferential statistics using PROC GLM. In order to prepare for these macro calls, the application program will subset for the lab

values to be used, and then reformat slightly so that both the actual value of the test and the change from baseline are on separate records, with a variable designating which output column they should be summarized in.

```
**  ------------------------------------------------------  ;
**  Prep data for analysis: set up column variable          ;
**  ------------------------------------------------------  ;

data lab ;
   set outdata.alab ;

   where labtest = 'Serum Glucose (mg/dl)' ;
   if txgrp = 1 then colvar = 1 ;
   if txgrp = 2 then colvar = 3 ;

   labtyp = 'Actual Value         ';
   output ;

   if visit > 1 then do ;
       labtyp = 'Change from Baseline';
       labval = labchg ;
       colvar = colvar + 1 ;
       output ;
   end ;
run ;
```

The macros do the actual work of producing the analysis:

```
%aptprep (
    infile   = lab  ,
    column   = colvar ,
    byvar    = labtest     ) ;

%aptdesc (
    brkvar  = visit  ,
    stats   =       N   | mean | std      | median
    statlabl= %str( N   | Mean | Std Dev  | Median )     ) ;

%aptpval (
    varname = labval ,
    brkvar  = visit  ,
    statanov= prob ,
    compare = 2_4     ) ;

ods rtf file="&prog..rtf" style=tempstyle bodytitle ;
title1 "Analysis Report of Change From Baseline by Lab and Time Point";
title2 "With P-values for Selected Pairs of Column Groups";

%aptprint (
    spanhdr1= %str(Treatment A  | Treatment B  | P-value ),
    spancol1=      1_2            | 3_4        | 5_5      ,
    headers = %str(Actual~Value | Change from~Baseline |
                   Actual~Value | Change from~Baseline |
                   Treatment~A vs B ),
    nequals = N    );

ods rtf close;
```

In the code above:

- APTPREP defines overall parameters for the report: name of input file and globally-used variables such as those that define the columns for the report (COLVAR), and page-by categorical variable LABTEST.
- APTDESC performs descriptive statistics on the variable LABVAL, broken out by variable VISIT (in addition to the global breakout variables LABTEST and COLVAR.
- APTPVAL generates comparative statistics between columns 2 and 4 only.
- ODS and TITLE statements redirect the output to an RTF file and assign titles for the report.

- APTPRINT formats and prints the output report. Parameters to this routine control spacing on the report, column headings, and other characteristics of the report appearance.

Frequently, a set of code is run repeatedly with variations in parameters by using either an external macro or one defined within the program (changes to code are in bold):

```
%macro doreport ( report = 1, labtest = ) ;
   data lab ;
      set outdata.alab ;

      where labtest = "&labtest" ;
      if txgrp = 1 then colvar = 1 ;
      if txgrp = 2 then colvar = 3 ;
      labtyp = 'Actual Value         ';
      output ;
      if visit > 1 then do ;
          labtyp = 'Change from Baseline';
          labval = labchg ;
          colvar = colvar + 1 ;
          output ;
      end;
   run;

   %aptprep ( ...    ) ;
   %aptdesc ( ...    ) ;
   %aptpval ( ...    ) ;

   ods rtf file="&prog._&report..rtf" style=tempstyle bodytitle ;
   title1 "APT Analysis Report of Change From Baseline by Lab and Time Point";
   title2 "With P-values for Selected Pairs of Column Groups";
   title3 "Lab Parameter: &labtest";

   %aptprint ( ...    ) ;

   ods rtf close;
   %aptreset ;

%mend ;

%doreport ( report = 1, labtest = %str(Serum Glucose (mg/dl)) ) ;
%doreport ( report = 1, labtest = %str(Serum Albumin (mg/dl)) ) ;
```

In this example, the set of code is called once for each lab parameter, and each call to %doreport produces a separate output file. Note(The macro call %aptreset has been added to clean up the working environment so that work datasets or global macro variables are not carried over from one report to the next.)

WHAT'S WANTED, AND WHAT'S NOT WANTED, IN A STANDALONE PROGRAM

We have determined that the only code we want to see in our generated program is the sections responsible for producing statistical results. In this example, then, we would want to see all of the processing involved in the data preparation before going into the standard macros. Once in the macro code, we would want to omit any diagnostic or preparatory steps that do not affect the input data, then capture the code that produces the statistics. Having produced the statistics, any steps that do manipulations for presentation should also be skipped.

If we look again at our analysis program (with some refinements added), we can see the sections where we will want the code captured or not captured (sections in gray are not wanted in the generated program):

```
** ------------------------------------------------------ ;
** Program Name: demo_basechg.sas                        ;
** Description: Demonstrate APT macros                   ;
** ------------------------------------------------------ ;
proc format ;
    value timept
    1 = 'Time Point 0 '
    2 = 'Time Point 1 '
    3 = 'Time Point 2 '
    4 = 'Time Point 3 '
    5 = 'Time Point 4 '
    ;
run;
%macro doreport ( report = 1, labtest = ) ;
** ------------------------------------------------------ ;
** Prep data for analysis: set up column variable ;
** ------------------------------------------------------ ;
  data lab ;
      set outdata.alab ;
      where labtest = "&labtest" ;
      if txgrp = 1 then colvar = 1 ;
      if txgrp = 2 then colvar = 3 ;
      labtyp = 'Actual Value         ';
      output ;
      if visit > 1 then do ;
          labtyp = 'Change from Baseline';
          labval = labchg ;
          colvar = colvar + 1 ;
          output ;
      end;
    format visit timept. ;
  run;
   %aptprep ( ...
              /* File prep */
              /* diagnostics */
                 ) ;
   %aptdesc ( segment = 1, varname = labval, seglabl = Lab Value ,
               ...
              /* Generate descriptive statistics */
              /* Reformat statistics work dataset for printing */
                 ) ;
   %aptpval ( segment = 1, varname = labval,
               ...
              /* Generate inferential statistics */
              /* Reformat statistics work dataset for printing */
                 ) ;
   ods rtf file="&prog._&report..rtf" style=tempstyle bodytitle ;
   title1 "APT Analysis Report of Change From Baseline by Lab and Time Point";
   title2 "With P-values for Selected Pairs of Column Groups";
   title3 "Lab Parameter: &labtest";

   %aptprint ( ...    ) ;

   ods rtf close;
   %aptreset ;

%mend ;

%doreport ( report = 1, labtest = %str(Serum Glucose (mg/dl)) ) ;
%doreport ( report = 1, labtest = %str(Serum Albumin (mg/dl)) ) ;
```

When the program is run through GENSAS, the resulting generated program would look like the following (for space considerations, assume that only the first %doreport macro call is executed) :

```
** ------------------------------------------------------------------------ **;
** Program:      gen_demo_basechg.sas                                        ;
** Source:       demo_basechg.sas                                            ;
** Generated:    02JUL06 21:11                                               ;
** Generated by: APT Reporting Tools V5.0  (c) 2006 Hygia Biostat, Inc.      ;
** Generated code begins below                                              ;
** ------------------------------------------------------------------------ **;
** ------------------------------------------------------ ;
** ;
** Program Name: demo_basechg.sas ;
** ;
** Description: Demonstrate APT macros ;
** ------------------------------------------------------ ;

proc format ;
    value timept 1 = 'Time Point 0 ' 2 = 'Time Point 1 ' 3 = 'Time Point 2 ' 4 = 'Time Point 3 '
5 = 'Time Point 4 ' ;
run;
** ------------------------------------------------------ ;
** Prep data for analysis: set up column variable ;
** ------------------------------------------------------ ;

data lab ;
    set outdata.alab ;
    where labtest = 'Serum Glucose (mg/dl)' ;
    if txgrp = 1 then colvar = 1 ;
    if txgrp = 2 then colvar = 3 ;
    labtyp = 'Actual Value         ';
    output ;
    if visit > 1 then do ;
        labtyp = 'Change from Baseline';
        labval = labchg ;
        colvar = colvar + 1 ;
        output ;
    end ;
    format visit timept. ;
run ;
title3 "Data Prepped for Analysis ";

proc print data = lab (obs = 30) ;
    by usubjid ;
    id usubjid ;
run ;

* ------------------------------------------------------------------------------- ;
* NOTE: Executing APT Macro Library Version 5.0 ;
* NOTE: Release date: April 2006 ;
* NOTE: APT Reporting Tools (c) 2000-2006 Hygia Biostat Inc. All Rights Reserved ;
* NOTE: Unauthorized use is forbidden ;
* ------------------------------------------------------------------------------- ;
title1 "-------------------------------------------------------------------------" ;
title2 "APT Table Generation from Dataset: LAB                                  " ;
title3 "Column Variable: COLVAR Subject ID Variable: USUBJID By Variable: LABTEST" ;
title4 "-------------------------------------------------------------------------" ;

data _aptin ;
    set lab ;
    dummy = 1 ;
    output ;
run ;
* ---------------------------------------------------------------- ;
* Beginning of Macro APTDESC V5.0 (19Apr06) P-values and Confidence Limits;
* ---------------------------------------------------------------- ;
title5 " APTDESC Report Segment 1 " ;
title6 " Variable: LABVAL  Segment Name: Lab Value " ;
** ------------------------------------------------ ** ;
** generate univariate statistics ** ;
```

```
** ------------------------------------------------ ** ;

proc sort data = _aptin ;
    by LABTEST visit COLVAR ;
run ;
title7 " Univariate Statistics for Variable LABVAL ";
title8 "-------------------------------------------------------------" ;

proc univariate data = _aptin ;
    by LABTEST visit COLVAR ;
    var labval ;
    output out = _aptdes1 N = N MEAN = MEAN STD = STD MEDIAN = MEDIAN ;
run ;

proc print data = _aptdes1 ;
    by LABTEST visit COLVAR ;
    id LABTEST visit COLVAR ;
run;
* ------------------------------------------------------------------ ;
* End of Macro APTDESC ;
* ------------------------------------------------------------------ ;
* ------------------------------------------------------------------ ;
* Beginning of Macro APTPVAL V5.0 (19Apr06) P-values and Confidence Limits;
* ------------------------------------------------------------------ ;
title5 " APTPVAL Report Segment 1 " ;
title6 " Variable LABVAL  Segment Name: Lab Value " ;

data _aptpval ;
    set _aptin ;
    if COLVAR not in ( 2 4 ) then delete ;
    dummy = 1 ;
run ;

proc sort data = _aptpval ;
    by LABTEST visit dummy COLVAR;
run ;
title7 " PROC GLM Comparative Statistics for Variable LABVAL ";
title8 "Note: Labels of contrasts (e.g. 1 v 2) refer to the order of values of variable COLVAR,
not the values themselves";
title9 "The order of values may be different from the actual values of COLVAR if only selected
values are used in the proced
ure";
title10 "-------------------------------------------------------------" ;

proc glm data = _aptpval outstat = _aptpst alpha = .05 ;
    format COLVAR ;
    by LABTEST visit dummy ;
    class COLVAR ;
    model labval = COLVAR ;
    lsmeans COLVAR / stderr ;
    CONTRAST "1:1v2   " COLVAR 1 -1 ;
run ;
* ------------------------------------------------------------------ ;
* End of Macro APTPVAL ;
* ------------------------------------------------------------------ ;
** ------------------------------------------------------------------------ **;
**
** End of generated program gen_demo_basechg.sas
**
** ------------------------------------------------------------------------ **;
```

When this program is executed, it will reproduce all of the numeric results that the production program does, albeit in a less ‘pretty’ format. The code contained in the program is short, clear, and strictly focused on the statistics.

The generated program also contains internal documentation, including user comments from the original program as well as macro-generated ones that allow a programmer to identify the sections of the original program that generated them.

Partial print of the output of the generated program:

```
------------------------------------------------------------------------------------------------
                              APT Table Generation from Dataset:LAB
Column Variable: COLVAR  Subject ID Variable: USUBJID By Variable: LABTEST  Break Variable: VISIT

                                    APTDESC Report Segment 1
                                Variable: LABVAL  Segment Name: Lab Value
                                Univariate Statistics for Variable LABVAL
------------------------------------------------------------------------------------------------

            labtest              visit        colvar     N         MEAN      STD      MEDIAN

     Serum Glucose (mg/dl)    Time Point 0       1      657     227.887    105.085     228.0

     Serum Glucose (mg/dl)    Time Point 0       3      651     224.091    104.560     225.0

     Serum Glucose (mg/dl)    Time Point 1       1      657     217.393    100.103     203.0

     Serum Glucose (mg/dl)    Time Point 1       2      657     -10.495    107.701     -12.0

(additional output)
{page}
------------------------------------------------------------------------------------------------
                              APT Table Generation from Dataset:LAB
   Column Variable: COLVAR  Subject ID Variable: USUBJID By Variable: LABTEST   Break Variable:
                                          VISIT

                                    APTPVAL Report Segment 1
                                Variable LABVAL  Segment Name: Lab Value
                          PROC GLM Comparative Statistics for Variable LABVAL
     Note: Labels of contrasts (e.g. 1 v 2) refer to the order of values of variable COLVAR,
                                      not the values themselves
          The order of values may be different from the actual values of COLVAR if only
                                selected values are used in the procedure
------------------------------------------------------------------------------------------------

------------------- labtest=Serum Glucose (mg/dl) visit=Time Point 1 dummy=1 --------------------

                                        The GLM Procedure

                                     Class Level Information

                                   Class         Levels    Values
                                   colvar             2        2 4


                                Number of Observations Read        1420
                                Number of Observations Used        1308
```

APT Table Generation from Dataset:LAB
Column Variable: COLVAR Subject ID Variable: USUBJID By Variable: LABTEST Break Variable: VISIT

APTPVAL Report Segment 1
Variable LABVAL Segment Name: Lab Value
PROC GLM Comparative Statistics for Variable LABVAL
Note: Labels of contrasts (e.g. 1 v 2) refer to the order of values of variable COLVAR,
not the values themselves
The order of values may be different from the actual values of COLVAR if only
selected values are used in the procedure

labtest=Serum Glucose (mg/dl) visit=Time Point 1 dummy=1

The GLM Procedure

Dependent Variable: labval

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	878.48	878.48	0.08	0.7830
Error	1306	15113020.66	11571.99		
Corrected Total	1307	15113899.14			

R-Square	Coeff Var	Root MSE	labval Mean
0.000058	-1111.420	107.5732	-9.678899

Source	DF	Type I SS	Mean Square	F Value	Pr > F
colvar	1	878.4792247	878.4792247	0.08	0.7830

Source	DF	Type III SS	Mean Square	F Value	Pr > F
colvar	1	878.4792247	878.4792247	0.08	0.7830

BENEFITS OF GENERATED CODE IN A PRODUCTION ENVIRONMENT

PROGRAM DELIVERABLES – THE REVIEWER'S VIEW

FDA Statistical Reviewers and clinical trial sponsor companies often request the statistical analysis code with the purpose of reproducing the results and completing an independent analysis. This has proven difficult for companies that use complicated macro libraries oriented toward publication quality tables. Challenges include transfer and installation of the software and training on its use. Furthermore, licensing issues may arise. Generating simple statistical code with GENSAS is an elegant solution. The programs are self contained and easily transportable to a different environment. Usually all that is needed is to modify an initialization procedure such as an autoexec.sas to point to the location of the input files.

Reviewers are usually selected for their statistical expertise, not their programming prowess. Their time is valuable, and not well spent in figuring out how to do data manipulation to replicate an analysis. A generated program can give them a quick 'jump-start' to reproducing results, and doing any variations they deem necessary.

PROGRAM VALIDATION – THE PROGRAMMER'S VIEW

A macro library can be a double-edged sword for a programmer. There's great power and efficiency to be gained, but transparency is lost. It's tempting to assume that the application program is correct because the macro library is well tested and validated. However, ***no program, however well tested, is immune to abuse***. Programmers must have a way of checking that the input data are appropriately set up for the macro calls, and that the macro parameters are appropriate for the requirements of the analysis.

With a well-designed macro system, it is fairly easy to do a code review of the application program, as long as it is done by someone familiar with the assumptions of the macros. The danger is that the assumptions may be subtle and easily overlooked. Reviewing a generated program, by contrast, will make the implications of these assumptions more clear because you can see the exact syntax of the statistical procedures. A generated program is much easier to desk check.

For example, imagine an application program that is designed to produce p-values through varying statistical methods. The application program is designed to use the macro keyword for a Fisher's exact p-value when &method = 1. Instead the production programmer accidentally assigns &method to a value of 2. The generated code would produce a proc frequency procedure with an option of "chisq". The error can be caught on code review.

After the validation programmer has conducted an initial code review, the code can then be executed independently of the base program to ensure that the results shown in the output are correct. In the example above, the results of the generated code will also make the error apparent.

Sometimes questions arise about whether the production macro is behaving correctly. Generated code can help to isolate problems by giving the programmer feedback about how macro parameters resolve into procedures. This can be invaluable in identifying whether the problem lies in the production macro or in its usage.

Many companies have internal standards that require completely independent programming for program validation. A generated program cannot be considered independent programming, since it is programmatically identical to the original program. However, it does provide a useful 'white-box' facility. Whether this is used as a substitute or an adjunct to independent programming is a policy decision to be made by the company.

EXPLORATORY ANALYSIS – THE STATISTICIAN'S VIEW

The generated program is likely to be useful to the project statistician in several ways. First, a statistician may want a simple, clear confirmation that the programmer implemented the analysis plan correctly. If there is data preprocessing such as subsetting in the original program, the statistician can confirm that these steps were done as intended.

By looking at the statistical procedures without the 'noise' of extraneous processing, a statistician can confirm that the correct procedure was used. In the code above, the statistician can confirm that the CLASS, MODEL, LSMEANS and CONTRAST statements will perform the intended statistical tests.

Second, a statistician can use the generated program to experiment with alternative statistical tests. If he or she wants to investigate the effects of using different coding of the statistical procedure, the generated program provides an easy starting point. The data preprocessing steps in the beginning of the program ensure that the alternative program is starting from the same data setup as the production program. This is a timesaver for statisticians, many of whom are not experts in programming data manipulation steps.

Third, a statistician may want to look in more detail at some of the additional information provided by the native SAS output of statistical procedures. The default PROC GLM output provides much more information than a p-value, although the p-value is all that typically appears in the final report. The statistician may want to examine the diagnostics and auxiliary statistics to confirm that the assumptions inherent in the procedure are satisfied by the data used for this analysis. The standalone program provides an easy way to do this. It is worth noting that, in the macro system used here, the original program can route the native SAS output to a debugging file for this purpose; macro switches are provided that allow for this. However, using the generated program allows the statistician to look at this output without having to search through any other debugging output.

FOR ALL USERS - A BLACK BOX WITH A WINDOW

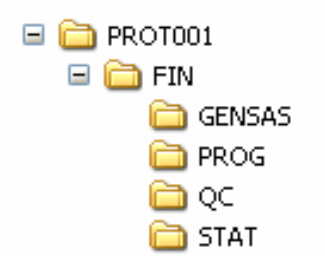
The most important drawback to macro libraries is their obscurity and complexity. The generated code option creates a 'window' into the black box, making the important sections transparent. Selectively routing the statistical sections to the generated code makes it possible to focus on the critical logic. Making the generated code executable gives us a 'comfort level' that we understand exactly what we are getting.

IMPLEMENTING MACRO LIBRARY WITH GENERATED CODE IN A PRODUCTION ENVIRONMENT

The secret to a successful production statistical analysis environment is organization and standards that are well documented and adhered to. The first step in implementing a macro library with a code generator is to evaluate how the generated code would be used. Use dictates structure. There have been three uses outlined in this paper so far: program deliverables, program validation and exploratory analysis by a statistician.

ORGANIZATION AND DIRECTORY STRUCTURE

Below is a directory structure for final analysis of a clinical trial protocol PROT001 that segregates the different uses of generated code.



Code generated from GENSAS resides here

Production code and output resides in PROG

QC contains validation code developed or copied from GENSAS

STAT has code copied from GENSAS or developed by the Statistician

The PROG folder contains the autoexec.sas (which has librefs and libnames and points to the macro library), production code and its generated statistical output. This is where programmers use the macro library to create publication ready output.

The GENSAS folder is where programmers can use the GENSAS macro to create the generated programs apart from the production environment. This folder has its own autoexec.sas, the program that calls the GENSAS macro, the resulting generated programs and outputs from the generated program. These programs can serve as deliverables to a FDA Statistical Reviewer or a sponsor company. They can also be copied over to the QC and/or STAT folder and modified for validation and exploratory analysis.

The folder for QC is the working area for the validation programmer. It is here that programs are independently written to check the work being completed in PROG. This area can also be used by the production programmer in order to check his or her own work. This folder contains an autoexec.sas, validation programs and corresponding output.

The statisticians can also have their own corner to verify the work of programmers and delve further into analysis methods. The generated program is a good starting point. This folder contains an autoexec.sas, statistical programs and corresponding output.

STANDARDS

Finally, there must be standards that are followed in the production code in order for GENSAS to produce a generated program that contains what is wanted, and omits what is not wanted. A standards team that has representatives from programming, statistics and quality control should develop and formalize rules that will meet everyone's needs. Production programmers will need to be trained on the techniques in this paper and the standards to ensure generated code meets expectations.

Below are standards to consider:

- Titles: Title statements carry over from procedure to procedure unless they are explicitly reset. The generated program will have assignments of titles that will take effect when the generated program is run. If a section of the code is intentionally omitted, so should the applicable titles. The location of title statements in the program can control this. A programming standard can remind the programmer to keep this mind. Also, the application program should set titles for the production output after all statistics are generated to avoid conflicts.
- Comments in generated program: MPRINT/MFILE does not capture the /* */ form of comments and pass to the generated program. Only comments beginning with * and ending with ; are resolved in the generated program. Having a standard concerning comment form which intentionally omits some and includes others can prove advantageous when the generated program is a deliverable to a reviewer or client. Another thing to consider is that MPRINT/MFILE will place the entire comment (from * to ;) on a single line in the generated program, removing any hard returns. Very long comments may prove difficult to read without the wrapping. This can be controlled by ending each line of a comment block with a semicolon. For example:

Comment Form	Generated Program
* This is a one line comment;	* This is a one line comment;
*This is a; *two line comment;	*This is a; *two line comment;
/* this one doesn't show up */	
%* this one doesn't show up either;	

- Overrides of SAS options: Any standard SAS options may be set in the standard initialization procedure for debugging purposes. When the GENSAS run is done, the application program should not have any overrides to SAS options.
- Interface with GENSAS: The macro library is set up to use a switch to determine whether it should override the MPRINT option and other SAS options that affect what is displayed in the log. In normal mode, it should not touch the SAS options; the overriding is done only when the application program is invoked from GENSAS. The normal SAS initialization procedure (typically autoexec.sas or and include file) must include this switch to notify the macro library that the normal mode is *not* to produce generated code. Therefore the initialization routine will contain the line:

```
%let _codegen = N ;
```

CONCLUSION

Good statistical practice emphasizes documented reproducible research. The ability to provide easy to read software code that generated the statistical results of a clinical study provides the transparency required to easily reproduce or audit the methodologies used. This leads to additional credibility of results reported.

The FDA Critical Path Initiative calls for new tools to accelerate the drug development process. Nowhere is the need for new tools and standards more evident than in the statistical processing of clinical trials data. We think that tools like GENSAS represent a small but significant step in streamlining the review of clinical trials results.

REFERENCES

1. FDA Guidance for Industry. E9: Statistical Principles for Clinical Trials. September 1998. http://www.fda.gov/cder/guidance/ICH_E9-fnl.PDF.

2. Guide to documents relating to the CDISC Analysis Data Model (ADaM) Version 2.0. Retrieved from <http://www.cdisc.org/models/adam/V2.0/index.html>.

3. FDA Guidance Document (2005).Study Data Specifications, Version 1.2. Retrieved from <http://www.fda.gov/cder/regulatory/ersr/Studydata-v1.2.pdf> March 2006.

CONTACT INFORMATION

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

Linda Collins
PharmaStat LLC
39899 Ballentine Drive, Suite 109
Newark, CA 94560
Work Phone: (510) 656-2080
Fax: (510) 656-2081
Email: lcollins@pharmastat.com
Web: www.pharmastat.com

Appendix1: Driver program using GENSAS

```
* Generate standalone programs for key tables ;
%gensas ( program = demo_basechg , outprog = gen_demo_basechg  ) ;
%gensas ( program = demo_demog   , outprog = gen_demo_demog  ) ;
%gensas ( program = demo_ae      , outprog = gen_demo_ae  ) ;
```

Appendix 2: Code for program GENSAS

```
%macro gensas   ( dir     = ..\prog\  ,
                  program = ,
                  outdir  = ..\gensas\ ,
                  outprog =
          ) ;


    %let program = %scan(&program,1,%str(.)) ;
    %let outprog = %scan(&outprog,1,%str(.)) ;
    %let _codegen = Y ;

    * ------------------------------------------------ *;
    * Direct the log file to the normal log of the     ;
    * program being processed                         ;
    * ------------------------------------------------ *;
    proc printto log="&program..log" new;
    run;

    * ------------------------------------------------ *;
    * Turn on MPRINT and run the program                ;
    * ------------------------------------------------ *;
    filename mprint "&program..tmp";
    options mprint mfile ;

    %include "&program..sas";
    options nomprint ;

    * ------------------------------------------------ *;
    * Copy the captured code to the generated program  ;
    * file, cleaning up indentation as we go           ;
    * and adding comments at the beginning and end     ;
    * ------------------------------------------------ *;

    proc printto print="&outdir.&outprog..sas" new;
    run;

    data _null_;
        infile "&dir.\&program..tmp" lrecl=200 pad  ;
        file print notitles pagesize=32000  ;
        length char1 $1 word1 word $25 ;
```

```
        retain indent 1 opendo 0 ;
        input text $200.;
        word1 = upcase(scan(left(text),1,' ;')) ;
        word1 = compress(word1,'0123456789') ;
        char1 = substr(word1,1,1) ;
        do i = 1 to length(text) ;
           word = upcase(scan(left(text),i,' ;')) ;
           if word = 'DO' then gotdo = 1 ;
        end ;
        if _n_ = 1 then do ;
           put "** ------------------------------------------------------------ **; ";
           put "** Program:      &outprog..sas                                  ;   ";
           put "** Source:       &program..sas                                  ;   ";
           put "** Generated:    &sysdate &systime                              ;   ";
           put "** Generated code begins below                                  ;   ";
           put "** ------------------------------------------------------------ **; ";
        end ;

        if word1 = 'MPRINT' then delete ;
        if substr(upcase(compress(text,' ;')),1,13) in ('OPTIONSNOMPRI','OPTIONSMPRINT')
        then delete ;
        if left(text) = ';' then delete ;

        if word1 in ('DATA','PROC','RUN','OPTIONS','TITLE','FOOTNOTE')
        or char1 in ('/','*')
        then indent = 1 ;
        if word1 in ('LABEL') then indent = 4 ;
        if word1 in ('ELSE','END') and opendo then indent = indent - 4 ;
        if word1 in ('END')  then opendo = opendo - 1 ;
        if word1 in ('RUN')  then opendo = 0 ;


        if word1 in ('DATA','PROC') then put ;
        put @indent text  ;

        if word1 in ('DATA','PROC','DO','SELECT')
            or gotdo then indent = indent + 4 ;
        if gotdo then opendo = opendo + 1 ;

        if indent < 1 then indent = 1 ;
    run ;

    data _null_;
        file print notitles pagesize=32000  ;
            put "** ------------------------------------------------------------ **; ";
            put "**                                                               ";
            put "** End of generated program &outprog..sas                       ";
            put "**                                                               ";
            put "** ------------------------------------------------------------ **; ";
    run ;
    proc printto;
    run;
    x "del &program..tmp";

%mend ;
```