SAS® Macro Techniques Useful in Development of Flexible Applications

Christian M. Forster, DZS Computer Solutions, Inc.

ABSTRACT

Building powerful, easy-to-use, flexible applications that prove useful for years can be an elusive task. Proper analysis and design as well as end-user involvement are essential to the preparation of system specifications for these applications. Breaking down the specifications into small, manageable, well-defined tasks or modules makes successful applications development possible.

This paper emphasizes the overall design considerations and development techniques used in a large, powerful SAS Macro Language application called GENTABLE. This generic application allows users to easily produce a variety of presentation-ready summary tables. The paper will document GENTABLE's main process flow and illustrate novel coding methods that may not be apparent from SAS Institute's documentation. Although GENTABLE is a reporting system, the SAS macro techniques presented should prove useful for any sort of application. Example calls to the system will demonstrate the power and flexibility that these advanced methods can provide.

INTRODUCTION

Use of flexible and powerful applications is one way corporations can distinguish their time-dependent work flow processes from their competitors. These applications should not only speed the process by providing quick results for decision support, but they should also allow for speed in terms of reacting to change. Issues involving applications development were discussed in more detail in my SUGI 18 paper entitled "Generic Applications Development Using the SAS® Macro Language."1

To demonstrate SAS Macro Language technique, this paper will discuss an application that was designed for use within the pharmaceutical industry. The application, called GENTABLE, was designed in such a manner that it would likely prove useful in any industry needing formal statistical displays. The focus of this paper will be on the user interface and extensible design of SAS macro applications.

The advent of the FRAME entry in SAS/AF has made interactive end-user applications development in the SAS system much more achievable. Interactive interfaces should be considered for most general-use SAS applications currently under development. Often, behind such applications is a "processing engine" that integrates information collected interactively from the user and performs the tasks required. This engine should be available in batch or interactive SAS execution modes and should be subjected to extensive validation and quality assurance. This paper will document techniques that can be used in development of a processing engine.

I will begin by discussing two example calls to GENTABLE to illustrate the power these techniques can provide for your applications. Particulars of the user interface and the basic process flow within GENTABLE will then be discussed by creating an example reporting system.

GENTABLE, A FLEXIBLE APPLICATION EXAMPLE

Two example calls to GENTABLE will be discussed to illustrate a powerful user interface and processing engine. In the first example (Figures 1A & 1B), the user is requesting the following statistics in the DISPLAY parameter:

- the total number of subjects (based on the SAS variable "PAT", the actual patient number):
  \[ N \ <\text{PAT}> \]

- the count and percent of patients for each category at endpoint:
  \[ \text{FREQ NEXTO PPERCENTP} \]

- a Fisher's Exact p-value and significance comparison between treatment groups (based on the SAS variable "DISC"):
  \[ \text{PVEXACT_EACH NEXTO SIG <DISC>} \]

By inspecting the DOWN and COMPARE parameters, note that the user wants treatment running down the page as well as comparisons between Drug 1 and Drug 2. Since the variable representing the groups being compared runs down the page, the actual comparisons are printed down the page (below the treatment group detail information). Also note in Figure 1B that statistics are running across the page (as per the STATSACROSS option).

Figure 1A - GENTABLE Call

```sas
%gentable(titles = Patient Status at Endpoint, display = n <pat> freq nextto ppercentp pvexact each nextto sig <disc>, compare = tmt 1 to 2, down = tmt, span = disc disc Patient Disposition at Endpoint, options = statsacross nostats solidlines blankOpt);
```

Figure 1B - GENTABLE Output

<table>
<thead>
<tr>
<th>Patient Status at Endpoint</th>
<th>Treatment</th>
<th>No of Subj</th>
<th>Adverse Reaction</th>
<th>Uncooperative</th>
<th>Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drug 1</td>
<td>5</td>
<td>2</td>
<td>(40%)</td>
<td>0</td>
<td>3 (60%)</td>
</tr>
<tr>
<td>Drug 2</td>
<td>4</td>
<td>0</td>
<td>3 (75%)</td>
<td>1</td>
<td>1 (25%)</td>
</tr>
<tr>
<td>P-values</td>
<td>0.444</td>
<td>0.048 *</td>
<td></td>
<td></td>
<td>0.524</td>
</tr>
</tbody>
</table>

Drug 1 vs Drug 2

112
In Figures 2A and 2B, the user is requesting statistics for blood pressure measurements and has chosen statistics more appropriate for continuous data. The DISPLAY parameter shows the following requests for the systolic (SYSBP) and diastolic (DIABP) blood pressures:

- the count, the mean, minimum and maximum rounded to integers, the standard deviation to one decimal place, and median:

  \[N \text{ MEAN 1. MIN 1. NEXTTO MAX 1. STD .1 MEDIAN}\]

- the LSMEANS p-value next to its significance from the GLM procedure comparing the two treatment groups:

  \[PVLSMEAN NEXTTO SIG <SYSBP DIABP>\]

The ACROSS, DOWN and COMPARE parameters show that the user wants treatment going across, week running down the page, and comparisons between Drug 1 and Drug 2. Since the variable representing the groups being compared runs across the page, the comparisons are printed as separate columns to the right of the treatment group detail information. Note also that the variables are running down the page in the first column (because VARSDOWN is specified in the OPTIONS parameter).

**Figure 2A - GENTABLE Call**

```
igentable(Titles = Vital Signs - Blood Pressure Statistics, Display = n mean 1. min 1. nextto max 1. std .1 median pvlsmean nextto sig <sysbp diabp>,
Across = tmt,
Down = week,
compare = tmt 1 to 2,
Stattext = min BP\max Range,
Options = varsdown solidlines rsub nodate nonumber);
```

**Figure 2B - GENTABLE Output**

<table>
<thead>
<tr>
<th>Variable</th>
<th>STATISTIC</th>
<th>Treatment Group</th>
<th>Drug 1 vs Drug 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Systolic BP</td>
<td>1 N Mean</td>
<td>112 120 7.9</td>
<td>113 139 3.8</td>
</tr>
<tr>
<td></td>
<td>BP Range</td>
<td>110 118 130</td>
<td>111 111</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>113 139 130</td>
<td>111 111</td>
</tr>
</tbody>
</table>

| Diastolic BP | 1 N Mean | 92 76 8.2 | 92 98 8.0 |
|             | BP Range | 71 68 6.0 | 71 68 6.0 |
|             | Median   | 71 68 6.0 | 71 68 6.0 |

**USER INTERFACE ISSUES**

It should be apparent from the above examples that a fair amount of time was spent on the batch user interface of this application. Many factors come into play when designing this important part of an application. These factors include ease of use, meeting system goals expressed in the design document, and allowing for the application's evolution beyond the current specifications. How often the processing engine is called outside of its interactive interface has a large impact on the sophistication and the time spent designing and implementing the batch interface. Below are some key aspects of the design of the GENTABLE user interface:

- **Keyword parameters** - Use of positional parameters should be avoided in end-user interfaces. Expecting the user to remember which positional parameter to set to a certain value, and including null entries for unspecified parameters is unreasonable. It is more convenient to set default values for macro variables using keyword parameters. Keyword parameters can also have meaningful names like TITLES, DISPLAY, ACROSS, and DOWN.

- **Appropriate delimiters** - Within a keyword parameter several different delimiters may be needed. Consistent use of delimiters and the use of spaces as delimiters are key factors to consider. However, using spaces as delimiters in parameters whose values may contain embedded spaces is not feasible. For example, title lines in the TITLES parameter are delineated by a backslash since the lines are likely to contain embedded spaces.

In the DISPLAY parameter, statistic keywords act on SAS variable names. Less-than and greater-than signs were chosen to serve as brackets for the variable names on which the requested statistics are to be performed. Optional rounding factors can follow each statistic keyword.

- **Provide a high degree of flexibility** - The application has a number of features which can be turned "on" or "off". The most straightforward method, for both developers and users, is to implement an OPTIONS parameter that behaves like the SAS OPTIONS statement. Macro variables within GENTABLE are initialized to either zero or one, defining the default value of an option. For example, the macro variable called "NUMBER" is initialized to "1" to indicate that page numbering is enabled. To disable page numbering, the user simply specifies NONUMBER in the OPTIONS parameter.

- **Allow parameter values to be insensitive to case.** Parameters whose actual values appear on the printed page should have their case preserved; all other parameters can be converted to uppercase prior to processing.

- **Derive as much information from the SAS environment as possible.** The output dataset from the CONTENTS procedure can be read to discover variable types, labels, and user-assigned formats. See my SUGI18 paper for one method to accomplish this.
MAIN PROCESS FLOW DESCRIPTION

To focus on the basic concepts behind GENTABLE, a barebones example application called SUGITAB will be built and illustrated. The actual SUGITAB application is of little utility other than that of a demonstration tool. Portions of code revealing interesting techniques will be examined.

The specifications for SUGITAB are as follows:

- Develop a SAS macro that can produce simple printouts of selected statistics from the MEANS and UNIVARIATE procedures. Potentially, any number of statistics in any order should be able to be requested for any number of SAS variables.

- Provide a parameter to specify title lines.

- Enable the user to make statistic requests followed by the SAS variable names to perform these statistics on in the DISPLAY parameter. The SAS variable names will be bracketed by less-than and greater-than symbols.

- Allow grouping variables down the left side of the report (specified in the DOWN parameter).

- Provide options such as debugging and labelling columns with the name of the statistic.

The system’s main macro routine (%SUGITAB) handles high-level tasks, default setting initialization, and scope of operation for the entire application. The portion of the SUGITAB macro shown in Figure 3 performs the following tasks:

- The assignment of the default input dataset (_LAST_).

- SUGITAB execution, version number, and environment announcement.

- A check that the environment is one under which the application has been validated. Issue warnings in unproven execution environments. (Note: SUGITAB has not been validated since ~ is only for demonstration purposes!)

- Make a copy of the input dataset to execute against.

- Convert appropriate parameters to uppercase and define constants.

The available statistic keyword set is defined in the main routine as a series of lists. The concept of using lists makes it very easy to add new statistic requests and categorize them. Statistics that are generated in the same manner, such as through the execution of the same procedure, can be considered as being in the same category. The next portion of the main routine (Figure 4) performs the following:

- Defines the available statistic keywords in STATNAME. STATNAME is then broken up into individual global macro variables by the BREAKUP macro (see SUGI 18 paper). The macro variables will be named CMD1 through CMD##, where ## is the number of supported statistics.

- Sets the default text for each statistic supported in STATTEXT. Note the mixed case and embedded spaces in the text for some statistics.

- Defines STATSFIX to hold the suffixes that are used to create new SAS variable names for the resulting statistics. The statistic SAS variable names will be of the form "V", followed by the SAS variable number, followed by the suffix for that statistic. For example, if the N statistic was requested for the first SAS variable appearing in the DISPLAY parameter, that statistic's SAS variable name would be V1.

- Initializes STATNEED, which will contain a "1" for each respective statistic keyword that appears in the DISPLAY parameter, and a "0" for unneeded statistics. To avoid errors due to empty datasets, N from the MEANS procedure is always generated but may or may not be printed. STATNEED is buffered by asterisks.

There is a one-to-one correspondence among the members of the STATNAME, STATTEXT, STATSFX and STATNEED lists.
Defines the &MEANSTAT and &UNIVSTAT lists that categorize the available statistics into those retrieved from the MEANS and UNIVARIATE procedures respectively. These lists are buffered with asterisks so that the INDEX function can be used against them and the results will not be ambiguous.

For example, to find out if the statistic "N" is produced from the MEANS procedure, the application can check the results of an index function call that resolves to %index("N·MIN·MAX·...·N'); the only exact match is the first member of the buffered string.

**Figure 4 - SUGITAB Macro continuation**

```
/* Comments removed - covered in text */
%let statname=N MIN MAX MEAN STD STDERR VAR MEDIAN MODE:
%breakup(&statname.ncmds.cmd.&bspace)
%let stattext=NIMinlMaxlMeanlStdlStd ErrorlVarlHedianlMode:
%breakup(&stattext,ncmds.statt,&bslash)
Xlet statsfix-N MN HX ME SO SE V HD HD:
%breakup(&statsfix,ncmds,sfix.&bspace)
%let statneed=l*O*O·O*O·O*O*O·O;
1* assume N is needed */
%let meanstat~*N*HIN*HAX*HEAN*STD*STDERR*VAR*:
%let univstat=*HEDIAN*HODE*;
```

Now we break up any SAS variables specified in the DOWN parameter and call the PARSDISP macro to parse the DISPLAY parameter into a separate statistic list for each SAS variable (Figure 5). After %PARSDISP executes, the following global macro variables will be defined:

NVARS: Number of SAS variables in the DISPLAY parameter.

VAR1 - VAR##: Macro variables holding the actual SAS variable names in the DISPLAY parameter (## is the total number of SAS variables).

NSTAT1 - NSTAT##: Number of requested statistics for each SAS variable.

STAT1N1 - STAT##N#: The statistic keywords for each SAS variable in the DISPLAY parameter (## is the number of the SAS variable, # is the statistic number for the SAS variable).

Next, the OPTIONS parameter is broken up, the pieces are checked for known options, and appropriate macro variable switches are set. A PROC CONTENTS is executed against the table dataset, and the CONTENTS procedure output is read and loaded into macro variables by the FINDTYPE macro (see SUGI 18 paper).

In Figure 6, %STATNEED is called to update the STATNEED macro variable that holds information on which statistics were requested. A "1" will appear in the slot if the statistic is needed; otherwise the slot will remain "0". The STATNEED macro variable is broken up into individual macro variables in the form SNEED##. For each member of the STATNAME list that appears in the DISPLAY parameter, the respective SNEED macro variable holds a "1" to indicate the need for that statistic's generation.

The DOMEANS macro, which produces statistics from the MEANS procedure, is always called to begin the creation of the _STATOUT dataset. Please see "Spotlight on %DOMEANS" below for more detail. The DOUNIV macro variable can then be used as a prefix to SAS statements that are executed only while debugging calls.

**Figure 5 - SUGITAB Macro continuation**

```
/* Comments covered in text */
%** Parse DOWN by variables;
%breakup(&down,ndown,down,&bspace)
%** Invoke DISPLAY parsing macro;
%parsdisp;
%** Parse OPTIONS;
%breakup(&options,ntopts,optn,&bspace)
%do x=1 Xto &ntopts;
  %if &&optn&x=DEBUG %then %do;
  1*
  %print diagnostics? /*
  %let diag=&bspace;
  %end;
  %else
    %if &&optn&x=NOSTATS %then %do;
      /* Print STATISTIC labels? */
      %let nostats=1;
    %end;
    %end;
%end;
&diag options mprint; &diag run;
proc contents data=&data noprint out= cfcont· run;
%** Find formats, types, and labels of variables;
%findtype(&nvars, var,vtype,vfmt,vlab); ,
%if &diag NE &star Xthen %do i=1 %to &nvars;
  %put &&vtype&i &&vfmt&i &&vLab&i;
%end;
%if &diag NE &star Xthen %do;
  * Diagnostic printing of macro variables to log;
  %put *** DIAGNOSTICS were requested ***;
  %put ndown=&ndown options=&options nvars=&nvars;
  %do x=1 Xto &ndown;
    %put down&x=&&down&x;
  %end;
  %do x=1 Xto &nvars;
    %put nstat&x=&&nstat&x;
  %end;
  %do x=1 Xto &nvars;
    %put var&x=&&var&x;
    %put stat&x.n&y=&&stat&x.n&y;
  %end;
%end;
```

In Figure 6, %STATNEED is called to update the STATNEED macro variable that holds information on which statistics were requested. A "1" will appear in the slot if the statistic is needed; otherwise the slot will remain "0". The STATNEED macro variable is broken up into individual macro variables in the form SNEED##. For each member of the STATNAME list that appears in the DISPLAY parameter, the respective SNEED macro variable holds a "1" to indicate the need for that statistic's generation.

The DOMEANS macro, which produces statistics from the MEANS procedure, is always called to begin the creation of the _STATOUT dataset. Please see "Spotlight on %DOMEANS" below for more detail. The DOUNIV macro variable can then be used as a prefix to SAS statements that are executed only while debugging calls.
called if statistics from the UNIVARIATE procedure are necessary. The _STATOUT dataset will eventually hold all the requested statistics for all the SAS variables in the DISPLAY parameter. Since a protocol exists for routines that add statistics to the _STATOUT dataset, new statistic-generating routines can be added without much difficulty. Producing all statistics for all variables makes construction of the statistical procedure calls very straightforward. It is easier to "weed out" unnecessary statistics during construction of the call to the REPORT procedure.

The MAKEVAR macro is called to assign labels and formats (if needed) to the statistic SAS variables. MAKEVAR completes all preparations prior to the call to the REPORT procedure. Finally, the PREREPORT macro is called to execute the REPORT procedure producing the printed output.

**Figure 6 - End of SUGITAB Macro Definition**

```plaintext
/* Comments removed - Covered in text */
 global assess needed stats - set 0 & 1 flags in STATNEED **;
 global xstatneed;
 xdo i=1 xto &ncmds;
 xif &&sneed&i xthen xput stat &&cmd&i is requested.;
 xend;
 x% a proc means is always done to get at least N **;
 %domeans
 %diag proc print data=_statout;
 %diag title "Means - Statout";
 run;
 x** check if DOUNIV macro will need to be called **;
 xlet douniv=0; /* Assume DOUNIV is not needed */
 xdo i=1 xto &ncmds;
 xif xindex(UNIVSTAT,&&star.&&cmd&i.&&star) GT 0
 and &&sneed&i xthen xput Stat &&cmd&i is requested.
 xend;
 x** Univariate stats are needed. Generate them **;
 xif &&douniv xthen xdo;
 %domeans
 data _statout; merge _statout _univar; by &down;
 %diag proc print data=_statout;
 %diag title "Statout after DOUNIV";
 run;
 xend;
 x** Assign labels, formats. Prep for PROC REPORT **;
 %makevar
 x** Call PROC REPORT... **;
 %report
 xend sugitab;

**SPOTLIGHT ON %DOMEANS**

Most portions of a well-designed, modular application should appear straightforward, almost simple to implement. This is partially because the complexity of the application is spread across many logically defined modules, reducing the complexity of each individual module of function. Hopefully, this is apparent after inspecting SUGITAB's main macro. A detailed look at a typical SUGITAB subroutine called DOMEANS should make this more clear.

The function of %DOMEANS is to initialize the SAS dataset that will hold all requested statistics for the call and to contribute any statistics coming from the MEANS procedure. In the development of most of the modules, including DOMEANS, it was useful first to outline the actual SAS code that the module needs to generate. For example, if N and MEAN were requested and the two SAS variables appearing in the DISPLAY parameter were AGE and SEX, the generated code would be:

```plaintext
proc sort data=sugitab; by <DOWN SAS 'BY' variables>
proc means data=sugitab noprint missing; by <BY vars>
var AGE SEX;
output out=_statout
N = V1 N V2N
MEAN = V1ME V2ME;
run;
```

The construction of the SAS variable names for statistics (V1N, V1ME, etc.) was discussed above. Inspecting the DOMEANS macro (Figure 7), the role of the statistic on/off macro variables (SNEED#, where # is the number of the statistic in STATNAME) and the category lists (MEANSTAT, UNIVSTAT, etc.) becomes more clear. The scan of all statistic keyword on/off switches, and adjoining check of &MEANSTAT (list of statistics from this procedure) automatically generates the correct PROC MEANS parameter. This is because the SNEED (needed statistics) switches, the CMD (statistic keywords) list, and SFIX (statistic suffix) list share the same indexing.

**Figure 7 - DOMEANS Macro Definition**

```plaintext
** DOMEANS - Comment Block removed - covered in text *
** COPYRIGHT 1994. DZS Computer Solutions. Inc. By C.Forster *
** PUBLIC DOMAIN VERSION 1.0 - FREELY DISTRIBUTE AS LONG AS *
** ABOVE COPYRIGHT NOTICE IS RETAINED *
%macro domeans:
 %local 1;
 proc sort data=sugitab; by &down;
 proc means data=sugitab noprint missing; by &down;
 var &do i=1 &do j=1 &nvars; &svar&j. &aspace &end;
 %end;
 %** Vars added from DISPLAY parameter **;
 %end;
 %** Variables from MEANS **;
 %do j=1 %to &nvars:
 %if &domeans xthen %do:
 %end;
 %end;
 %** Univariate stats are needed. Generate them **;
 %if &&domeans xthen xdo:
 %domeans
 data _statout; merge _statout _univar; by &down;
 %diag proc print data=_statout;
 %diag title "Statout after DOUNIV";
 run;
 %end;
 %** Assign labels, formats. Prep for PROC REPORT **;
 %makevar
 %** Call PROC REPORT... **;
 %report
 %end;
 %end sugitab;
```

**CONCLUSION**

In this paper I have tried to share some techniques you may find useful in developing applications. The methods shown use the simple macro functions available to facilitate higher-order functionality, such as a form of list processing. Through intelligent use of the SAS Macro Facility one can create very useful and time-saving applications that are easy to build.
upon. Spending more time on the design and initial implementation of an application can result in significant savings in later stages of development. The main objective has been to encourage modular design in applications development by showing the power and flexibility it can yield.

CREDITS

Christian M. Forster from DZS Computer Solutions is the designer and developer of GENLIST.

Christian M. Forster and Amy Matcho from DZS Computer Solutions are the designers and developers of GENTABLE.

REFERENCES

'Forster, Christian M. "Generic Applications Development Using the SAS® Macro Language", SUGI 18 Proceedings.

SAS is a registered trademark of the SAS Institute, Cary, NC, USA in the USA and other countries. ® indicates USA registration.

CONTACTS

Doron Z. Steger, President or Christian M. Forster, Medical Systems Manager/Analyst
DZS Computer Solutions, Inc. (908) 356 - 6961
1661 Route 22 West
Bound Brook, NJ 08805

APPENDIX 1 - SUGITAB Example Calls

>>>>>>>>>>>> PORTIONS OF SAS JOB

** SUGITAB example calls;
proc format;
  value tmtf 1='Drug 1' 2='Drug 2';
/* Construct fake data for test calls. SysBP is supposed to be Systolic BP and DiaBP Diastolic BP. Although I assign WEEK to 1 and 2, these have no real meaning. In the 1st call, I use WEEK 1 as Endpoint. In the next call, WEEK 1 and WEEK 2 are used as is.

Once again, the data used in these calls is just for demo purposes to produce simple of reports. */
data test;
  week=1; tmt=1; pat=1; wt=180; sysbp=120; diabp=78;
/* Assign FORMATS & LABELS that SUGITAB will read and use. */
format tmt tmtf.;
label pat='Subjects Present' wt='Weight (lbs)' sysbp='Systolic BP' diabp='Diastolic BP';
run;
/* Call SUGITAB to print the N of Patients, and the MEAN MIN MAX MEDIAN of Weight */
%sugitab(titles=Patient Weight at Endpoint, display=n <pat> mean min max median <wt>, down=tmt, options=diag);
run;
/* More demonstration data is added */
/* Call SUGITAB to print the N of readings, and MEAN of SYStolic & DIAstolic BP */
%sugitab(Titles=Blood Pressure Statistics by Week, Display=n mean <sysbp diabp>, Down=tmt week)
run;

Executing SUGITAB ver 1.00 by Chris Forster and Amy Matcho, (c)1994 DZS Computer Solutions, Inc. for SUGI 19 demonstration under OS2 running SAS 6.06.

WARNING: SUGITAB is executing in an environment that has not been validated under.

*** DIAGNOSTICS were requested ***
rdnum=1 options=diag nvars=2
down1=TMT nstat1=1
nstat2=4
var1=PT
stat1n1=N
stat2n=MAX
stat2n2=MEDIAN
MPRINT(DONEANS): PROC SORT DATA=_SUGITAB; BY TMT;
NOTE: The data set WORK._SUGITAB has 10 observations and 6 variables.
NOTE: The PROCEDURE SORT used 1.03 seconds.
MPRINT(DONEANS): PROC MEANS DATA=_SUGITAB NOPRINT MISSING;
MPRINT(DONEANS): BY TMT;
MPRINT(DONEANS): VAR PAT WT ;
MPRINT(DONEANS): OUTPUT OUT=_STATOUT N= V1N V2N MIN= V1MN V2MN MAX= V1MX V2MX MEAN= V1ME V2ME ;
NOTE: The data set WORK._STATOUT has 2 observations and 11 variables.
NOTE: The PROCEDURE MEANS used 7.00 seconds.
MPRINT(DOUNIV): PROC UNIVARIATE DATA=_SUGITAB NOPRINT;
MPRINT(DOUNIV): BY TMT;
MPRINT(DOUNIV): VAR PAT WT ;
MPRINT(DOUNIV): OUTPUT OUT=_UNIVAR MEDIAN= V1MD V2MD ;
MPRINT(DOUNIV): RUN;
NOTE: The data set WORK._UNIVAR has 2 observations and 3 variables.
NOTE: The PROCEDURE UNIVARIATE used 6.44 seconds.
MPRINT(SUGITAB): DATA _STATOUT;
MPRINT(SUGITAB): MERGE _STATOUT _UNIVAR;
MPRINT(SUGITAB): BY TMT;
NOTE: The data set WORK._STATOUT has 2 observations and 13
variables.
NOTE: The DATA statement used 2.72 seconds.

MPRINT(MAKEVAR): DATA _STATOUT;
MPRINT(MAKEVAR): SET _STATOUT;
MPRINT(MAKEVAR): LABEL V1N="Subjects Present";
MPRINT(MAKEVAR): LABEL V2ME= "Mean";
MPRINT(MAKEVAR): LABEL V2MN= "Min";
MPRINT(MAKEVAR): LABEL V2MX= "Max";
MPRINT(MAKEVAR): LABEL V2MD= "Median";
MPRINT(MAKEVAR): RUN;

NOTE: The data set WORK._STATOUT has 2 observations and 13 variables.
NOTE: The DATA statement used 14.77 seconds.

MPRINT(PREPORT): DATA STATDUT;
SET STATOUT;
LABEL V1N="Subjects Present";
LABEL V2ME= "Mean",
LABEL V2MN= "Min";
LABEL V2MX= "Max";
LABEL V2MD= "Median";
RUN;

NOTE: The PROCEDURE REPORT used 3.31 seconds.

Poster Weight at Endpoint

<table>
<thead>
<tr>
<th>Treatment</th>
<th>Subjects Present</th>
<th>Weight (lbs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drug 1</td>
<td>6</td>
<td>155.0</td>
</tr>
<tr>
<td>Drug 2</td>
<td>4</td>
<td>157.5</td>
</tr>
</tbody>
</table>

**Diastolic BP**

<table>
<thead>
<tr>
<th>Treatment</th>
<th>Week</th>
<th>N</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drug 1</td>
<td>1</td>
<td>6</td>
<td>122.5</td>
</tr>
<tr>
<td>Drug 1</td>
<td>2</td>
<td>6</td>
<td>121.3</td>
</tr>
<tr>
<td>Drug 2</td>
<td>1</td>
<td>4</td>
<td>112.5</td>
</tr>
<tr>
<td>Drug 2</td>
<td>2</td>
<td>4</td>
<td>114.0</td>
</tr>
</tbody>
</table>

END OF APPENDIX