PharmaSUG2010 - Paper TT05

The Ubiquitous Clinical Trials Data Summary Table
"Summary Statistics in Rows"

John Henry King, Ouachita Clinical Data Services, Inc. Mount Ida, AR

ABSTRACT

Many if not most summary tables created from clinical trials data use basically the same format, they display a mixture of discrete (categorical) and/or continuous variables in a style that could be referred to as "summary statistics in rows". Each variable whether, discrete or continuous forms a group of rows in the table, the descriptive labels for these variables form the first left most column of the table. Continuous variable rows are made up of descriptive statistic descriptions e.g. N, Mean, Standard Deviation, Median, Minimum and Maximum. It is common for two or more of these statistics to be displayed on the same row; "Mean (SD)" and "Min, Max". Discrete variable rows are formed from the descriptive values of the levels of each discrete variable, e.g. levels of Gender (Male, Female). Usually the treatment variable from the clinical trial forms groupings in columns to the right of the row descriptions. The table's cells are either count and column percent for discrete variables or the values of summary statistics for continuous variables.

This table style is not directly obtainable in SAS® and many pharmaceutical companies and individuals have devoted great effort in programming elaborate macros to produce this table. This paper presents a simple yet dynamic program that uses SAS procedures, standard SAS metadata objects FORMAT and LABEL and a few macro variables to produce tables in the style of summary statistics in rows. Summary statistics are displayed with a decimal precision that is derived from each continuous variable's format. Each variable's LABEL is used to describe the variable in the table. The formatted value of the discrete variables is used to label the rows for each discrete variable. Complete rows and/or columns of zero counts are accommodated using SAS procedure options. This is all accomplished without a macro, using SAS procedure and data steps, yet the program is dynamic with regards to variables summarized and treatment details.

INTRODUCTION

The most common clinical trials table looks more or less like the example shown in Figure 1. Fonts and titling may be different but the table should look very familiar to anyone working with clinical trials data.

This table style is used for both safety and efficacy data summaries. It may have variables of all one type, categorical or continuous, or a mixture. The summary statistics displayed here are easily obtainable with SAS procedures SUMMARY and FREQ; it is the formatting that makes this table require extra attention, specifically the table cells that contain values from two statistics, N (%), Mean (SD) and Min, Max. Also, the row labels which usually consists of data from two variables that are displayed in one column, this will require some addition processing.

This goal of this paper is to describe a program that will produce tables of this type. This program will be dynamic but will not use a complicated macro to achieve the results. Without a complicated macro the program can be readily distributed to clients without the need to support or explain or otherwise document a complicated macro. The program uses SAS PROCEDURES and simple data steps to achieve the results. The syntax and other details are fully documented in the SAS user documentation. The program uses a simple user interface of %LET statements.

Table 14-2.1
Summary of Demographic Characteristics at Baseline

	Treatment		
	Placebo (N=13)	*Active (N=6)*	*Total (N=19)*
Gender, n(%)			
Female	7 (53.8)	2 (33.3)	9 (47.4)
Male	6 (46.2)	4 (66.7)	10 (52.6)
Ethnic Origin, n(%)			
White	7 (53.8)	4 (66.7)	11 (57.9)
Black	6 (46.2)	1 (16.7)	7 (36.8)
Hispanic	0	1 (16.7)	1 (5.3)
Other	0	0	0
Age (years)			
N	13	6	19
Mean (SD)	13.2 (1.5)	13.7 (1.6)	13.3 (1.5)
Median	13.0	13.5	13.0
Min, Max	11, 15	12, 16	11, 16
Age group, n(%)			
10 and Under	0	0	0
Pre-teen	5 (38.5)	2 (33.3)	7 (36.8)
Teen	8 (61.5)	4 (66.7)	12 (63.2)

Figure 1 Example of a Summary Statistics in Rows table.

The program uses PROC REPORT as the “print engine” to display a specially structured data set that is printed with very generic PROC REPORT syntax. Therefore most of the program is concerned with building this data set. The PROC CONTENTS output shown in Figure 2 describes the variables created by the program. Variable numbers 1-6 will be always be created, variable numbers 7 and higher are the treatment column variables. The number of COL_n variables depends on the number of levels for the treatment variable. In the example data there are three levels of treatment.

#	Variable	Type	Len	Label
1	page	Num	8	Break variable used in PROC REPORT BREAK statement
2	vorder	Num	8	Order variable used to order the analysis variables, defined in PROC REPORT as ORDER NOPRINT.
3	vname	Char	32	Analysis variable name
4	vlabel	Char	256	Analysis variable label, defined in PROCREPORT as ORDER NOPRINT, displayed with LINE statement in COMPUTE block.
5	roworder	Num	8	Order variable for row labels, defined in PROC REPORT as ORDER NOPRINT
6	rowlabel	Char	256	Detail row label, derived from categories or statistic labels
7	col_1	Char	32	Placebo~(N=13)
8	col_2	Char	32	Active~(N=6)
9	col_3	Char	32	Total~(N=19)

Figure 2 Variables information from PROC CONTENTS.

An example of the data contained in this data set is displayed in Figure 3. Variables COL_2 and COL_3 have been omitted here.

page	vorder	vname	vlabel	roworder	rowlabel	col_1
1	3	Age	Age (years)	1	N	13
1	3	Age	Age (years)	2	Mean (SD)	13.2 (1.5)
1	3	Age	Age (years)	3	Median	13.0
1	3	Age	Age (years)	4	Min, Max	11, 15
2	6	Height	Height (inches)	1	N	13
2	6	Height	Height (inches)	2	Mean (SD)	61.87 (4.81)
2	6	Height	Height (inches)	3	Median	62.50
2	6	Height	Height (inches)	4	Min, Max	51.3, 69.0
2	7	Weight	Weight (lbs.)	1	N	13
2	7	Weight	Weight (lbs.)	2	Mean (SD)	94.31 (17.68)
2	7	Weight	Weight (lbs.)	3	Median	98.00
2	7	Weight	Weight (lbs.)	4	Min, Max	50.5, 112.5
2	5	bmi	BMI (kg/m**2)	1	N	13
2	5	bmi	BMI (kg/m**2)	2	Mean (SD)	17.181 (1.917)
2	5	bmi	BMI (kg/m**2)	3	Median	17.772
2	5	bmi	BMI (kg/m**2)	4	Min, Max	13.49, 20.25
1	4	ageg	Age group, n(%)	1	10 and Under	0
1	4	ageg	Age group, n(%)	2	Pre-teen	5 (38.5)
1	4	ageg	Age group, n(%)	3	Teen	8 (61.5)
1	2	race	Ethnic Origin, n(%)	1	White	7 (53.8)
1	2	race	Ethnic Origin, n(%)	2	Black	6 (46.2)
1	2	race	Ethnic Origin, n(%)	3	Hispanic	0
1	2	race	Ethnic Origin, n(%)	4	Other	0
1	1	Sex	Gender, n(%)	1	Female	7 (53.8)
1	1	Sex	Gender, n(%)	2	Male	6 (46.2)

Figure 3 Example of data set produce by the program.

DETAILS

SUBJECT LEVEL DATA

The program accepts a subject level data set similar to the ADaM data ADSL. This data set has one observation for each subject, a treatment variable used to define summary table columns and analysis variables. The statements below create a data set that will demonstrate the features of the program.

```
proc format;
   value trt   1='Placebo' 2='Active' 3='Total';
   value $sex 'F'='Female' 'M'='Male';
   value race  1 ='White' 2='Black' 3='Hispanic' 4='Other';
```

```
value age    low-10 = '10 and Under' 11-12='Pre-teen' 13-high='Teen';
run;
```

These formats are used to label the output. Each categorical variable should be formatted if it is a coded variable, e.g. TRT, RACE and SEX or if there are levels of the variable that do not appear in the data. The grouping format age will be used to create and summarize age groups.

We can use SASHELP.CLASS to model ADSL, treatment (TRT) and a few other variables are created to make the output more interesting. Notice that each variable is given a LABEL and a FORMAT, these metadata objects will be used to annotate the output.

```
data ADSL;
   set sashelp.class;
   trt  = rantbl(12345,.5);
   race = rantbl(12345,.5,.4);
   ageg = age;
   bmi  = (weight*703) / height**2;
   attrib age     format=F3.0    label='Age (years)';
   attrib height  format=F7.1    label='Height (inches)';
   attrib weight  format=F7.1    label='Weight (lbs.)';
   attrib sex     format=$sex.   label='Gender';
   attrib race    format=race.   label='Ethnic Origin';
   attrib ageg    format=age.    label='Age group';
   attrib bmi     format=F7.2    label='BMI (kg/m**2)';
   attrib trt     format=trt.    label='Treatment';
   run;
```

The data created by this data step are shown in Figure 4, with the formatting removed.

Obs	Name	Sex	Age	Height	Weight	trt	race	ageg	bmi
1	Alfred	M	14	69.0	112.5	1	2	14	16.6115
2	Alice	F	13	56.5	84.0	2	1	13	18.4986
3	Barbara	F	13	65.3	98.0	1	2	13	16.1568
4	Carol	F	14	62.8	102.5	1	2	14	18.2709
5	Henry	M	14	63.5	102.5	2	2	14	17.8703
6	James	M	12	57.3	83.0	1	1	12	17.7715
7	Jane	F	12	59.8	84.5	1	1	12	16.6115
8	Janet	F	15	62.5	112.5	1	2	15	20.2464
9	Jeffrey	M	13	62.5	84.0	1	1	13	15.1173
10	John	M	12	59.0	99.5	1	1	12	20.0944
11	Joyce	F	11	51.3	50.5	1	2	11	13.4900
12	Judy	F	14	64.3	90.0	1	2	14	15.3030
13	Louise	F	12	56.3	77.0	2	1	12	17.0777
14	Mary	F	15	66.5	112.0	1	1	15	17.8045
15	Philip	M	16	72.0	150.0	2	1	16	20.3414
16	Robert	M	12	64.8	128.0	2	1	12	21.4297
17	Ronald	M	15	67.0	133.0	2	3	15	20.8285
18	Thomas	M	11	57.5	85.0	1	1	11	18.0733
19	William	M	15	66.5	112.0	1	1	15	17.8045

Figure 4 Data ADSL, example input data.

We also want to display a total column this is done by creating a third treatment group which includes every observation. This method of creating the total column removes the burden from the program, which would require extra parameters and unnecessarily complicate the program. There are many situations where a simple total is not adequate or not needed so the details are better left to the user.

```
data adslT;
   set adsl;
   attrib ctrt length=$1 label='Character Treatment';
   do trt=trt,3;
      ctrt = substr('ABC',trt,1);
      output;
```

```
    end;
run;
```

The variable CTRT was created to test the program with a character treatment variable. The result of this step is a data set with 38 observations, twice as many as we started with.

PARAMETERIZATION

A dynamic program needs a user interface, for this program a few macro variables serve the purpose.

```
%let data = adslT;
%let trt  = trt;
%let cvars = bmi age-numeric-weight;
%let dvars = sex race ageg;
%let avars = sex race age ageg _page_ bmi height weight;
%let rowofn = yes;
```

These macro variables are the parameters for the program, they provide:

- DATA: the name of the input data set.
- TRT: the name of the column grouping variable, I call it TRT because that is the most often used variable, but it could be anything. This variable can be either character of numeric.
- CVARS: a list of continuous analysis variables. They must be numeric, and should be formatted with a W.D format.
- DVARS: a list of discrete categorical variables. They can be character or numeric, the formatted values of these variables will become row labels.
- AVARS: an ordered list of the variables found in CVARS and DVARS used to determine the order of each variable in the output. Include the word _PAGE_ to trigger a new page in the output. This parameter may be omitted in which case the variables are displayed in the order they appear in the input data.
- ROWOFN is a Boolean option to switch on or off the display of an extra row that displays the denominator for discrete variables.

BIG N FOR (N=NN) TREATMENT COLUMN HEADINGS

The following PROC SUMMARY counts the total subjects in each treatment group, shown in Figure 5. The PRELOADFMT and COMPLETETYPES options insure that all treatment levels are created even if they have zero counts. The LEVELS option on the output statement will create a new variable _LEVEL_, with values 1, 2, 3…, n; where n is this total number of treatments. This variable will be used later on to name the column variables, when &TRT is transposed. Numbering the column variables in this way gives a nice generic naming convention for the column variables; the formatted value of &TRT will provide column labels via the IDLABEL statement in PROC TRANSPOSE, more on that below. A simple index is created to facilitate accessing the observations.

```
proc summary data=&data nway completetypes;
   class &trt / preloadfmt;
   output out=BigN(drop=_type_ rename=(_freq_=bigN) index=(&trt)) / levels;
   run;
```

Obs	trt	_LEVEL_	bigN
1	Placebo	1	13
2	Active	2	6
3	Total	3	19

Figure 5 Data BIGN.

A BIT OF HOUSE KEEPING

There is a bit of house keeping that needs to be attended to. I had originally scattered these bits throughout the program but they are mostly easily handled in a small data step so they have been consolidated here. They include:

- Parsing ROWOFN
- Obtaining program path for use in naming output and identifying source program name on the table.

- Creating a macro variable for number of treatment columns in the report.
- Creating a macro variable for a spanning title.

ROWOFN values (YES, 1, or ON) are parsed into '01' or a blank. The macro variable ROWOFN is replaced with the value resulting from parsing. The value 01 is equal to a level of the variable _TYPE_ created by PROC FREQ, which is the number of observations used in the denominator for a discrete variable. It will be used below to select observations from PROC FREQ output.

To create a spanning column header for the treatment columns the label associated with the treatment variable is put into a macro variable. A macro variable is used because the text needs to be included in the PROC REPORT COLUMN statement. Also, define the style element CELLWIDTH for the treatment columns. Here I have chosen to allow the treatment columns to occupy 60% of the table width. The number of observations in BIGN is the number of treatment columns and is used to compute the percentage allowed for each column.

CALL EXECUTE statements are included to %PUT the macro variables created in this step as a debugging and documentation aid.

```
data _null_;
   if 0 then set bign(keep=&trt) nobs=nobs;

   length path $256;
   if getoption('DMS') eq 'DMS'
      then path = sysget('SAS_EXECFILEPATH');
      else path = getoption('SYSIN');

   path = substr(path,1,find(path,'.',-vlength(path)));

   length rowofn $4;
   if symexist('ROWOFN') then rowofn = upcase(symget('ROWOFN'));
   if substrN(rowofn,1,1) in('1' 'Y' 'O')
      then rowofn = "'01'";
      else rowofn = ' ';
   call symputX('PATH',path,'G');
   call symputX('ROWOFN',rowofn,'G');
   call symputX('TRTLabel',vlabel(&trt),'G');
   call symputX('TRTPercent',int(60/nobs),'G');
   call symputX('COL0',nobs,'G');
   call execute('%put NOTE: Macro variables created:;');
   call execute('%put NOTE- PATH=%nrbquote(&path);');
   call execute('%put NOTE- ROWOFN=%nrbquote(&rowofn);');
   call execute('%PUT NOTE- TRTLabel=&TRTLABEL;');
   call execute('%PUT NOTE- TRTPercent=&TRTpercent;');
   call execute('%PUT NOTE- COL0=&col0;');
   stop;
   run;
```

CATEGORIAL VARIABLE SUMMARY

Categorical/discrete variables are summarized with frequencies and column percentages. The following data and procedure steps process the variables listed in the DVARS macro variable and produces the output needed for display in the final table. An important feature of this process is that all variables are summarized together in PROC and DATA steps that process all variables at once, no looping over multiple variables calling the same procedure and passing through the data over and over again.

The variable list parameters DVARS, CVARS and AVARS can, in addition to individual variable names, accept a SAS Variable List. There is some possibility that the SAS Variable List could become ambiguous when used in a context different from that of the input data. To insure this does not happen the variable list is expanded into individual variable names. PROC TRANSPOSE provides a simple but powerful solution. When PROC TRANSPOSE is used to transpose a data set with zero observations the output from the transpose is simply a data set with one observation for each variable listed in the VAR statement. For the parameter DVARS the expanded list is used to replace the value of DVARS.

```
proc transpose data=&data(obs=0) out=dvars;
   var &dvars;
   run;
```

PROC SQL is used to recreate the macro variable with the expanded list of variable names.

```
proc sql noprint;
   select _name_ into :dvars  separated by ' ' from dvars;
   quit;
   run;
%put NOTE: DVARS=&dvars;
```

Clinical trials tables often require summary of data levels that may not exist in the data. For example if there are no subjects in a demographic variable group but we still want to see a row of zeros. The PRELOADFMT option combined with the COMPLETETYPES option in PROC SUMMARY allows us to create such observations, when the properly defined value labeling formats are associated with the CLASS variables. When ADSL was created above each categorical variable was formatted with a value labeling format.

```
proc summary data=&data completetypes chartype;
   class &dvars &trt / preloadfmt;
   types (&dvars)*&trt;
   output out=discrete(rename=(_type_=dtype));
   run;
```

PROC FREQ with a WEIGHT statement and the ZEROS option is used to compute percentages using the output data set produced by PROC SUMMARY. The ODS OUPTUT data set CROSSTABFREQS has convenient properties that allow for easy processing. Notice the use of ROWOFN in the WHERE data set option. If ROWOFN is requested the IN operator includes '01' to select the observations where _TYPE_ = '01'.

```
ods listing close;
proc freq data=discrete;
   by dtype;
   tables (&dvars)*&trt / norow nopercent;
   weight _freq_ / zeros;
   ods output CrossTabFreqs=CrossTabFreqs(where=(_type_ in(&rowofn '11')));
   run;
ods listing;
```

Using the output from PROC FREQ we create new variables from the variables in CROSSTABFREQS and BIGN that conform to the data structure we want to display with PROC REPORT. VNAME the analysis variable name is derived from CROSSTABFREQS variable TABLE. ROWLABEL is derived from the formatted value of the analysis variable. Each analysis variable is included in CROSSTABFREQS and the VVALUEX function allows the retrieval of the formatted value using the value of VNAME. VVALUEX operates on either character of numeric variables to return the formatted value as character string. The CROSSTABFREQS variables FREQUENCY and COLPERCENT are concatenated to create a new variable CELL in the N (%) style. When FREQUENCY is zero the percent is omitted, this keeps the output a little cleaner.

A variable IDLABEL is created to use in PROC TRANSPOSE IDLABEL statement in the next step. The values of BIGN are looked up for each &TRT using a SET statement with keyed access. IDLABEL is a concatenation of the formatted value of &TRT, returned from function VVALUE with (N=nn) added.

```
data CrossTabFreqs(keep=vname rowlabel cell _level_ idlabel _type_);
   length vname $32 rowlabel $64 cell $32 idlabel $64;
   set CrossTabFreqs;
   vname    = vnamex(scan(table,2,' '));
   if _type_ eq '01'
      then rowlabel = 'n';
      else rowlabel = vvalueX(vname);

   if colPercent gt 0
      then cell = catx(' ',Frequency,cats('(',vvalue(colPercent),')'));
      else cell = cats(Frequency);
   format colPercent 8.1;

   set bign key=&trt/unique;
   idlabel = catx('~',vvalue(&trt),cats('(N=',bign,')'));
   run;
```

There are two more steps to the get the data into the format needed for display with PROC REPORT. We transpose the formatted cell values into a variable for each treatment. The variable _LEVEL_ created by the LEVELS option in

PROC SUMMARY above is used to name the new variables. The formatted values of the treatment variable combined with the value of BIGN are used to create labels for the new variables using the IDLABEL statement.

The advantage of transposing the treatments into an enumerated list makes it easy to refer to all the treatment columns using a SAS Variable List. We are no longer concerned with the attributes of the treatment variable or how many treatment levels there are.

```
proc transpose data=CrossTabFreqs out=Discrete(drop=_name_) prefix=Col_;
   by vname rowlabel _type_ notsorted;
   var cell;
   id _level_;
   idlabel idlabel;
   run;
```

To keep the rows of each categorical variable in the proper order a companion variable is created ROWORDER.

```
data Discrete;
   set Discrete;
   by vname notsorted;
   if first.vname then roworder = 0;
   roworder + 1;
   if _type_ eq '01' then roworder = 0;
   drop _type_;
   run;
```

The categorical variable data summaries are now ready for display they will be combined with the continuous data below.

CONTINUOUS VARIABLE SUMMARY

Once again we use PROC TRANSPOSE to expand the list of names, this time the names in CVARS. PROC SQL is used to put the expanded list back into the macro variable CVARS.

```
proc transpose data=&data(obs=0) out=cvars;
   var &cvars;
   run;
proc sql noprint;
   select _name_
      into :cvars separated by ' '
      from cvars;
   quit;
   run;
%put NOTE: CVARS=&cvars;
```

The continuous variable statistics are displayed with formats based on the formats associated with the original variables. Some statistics are formatted with the same number of decimal places some with one more, some statistics like N are formatted with zero decimal places as integers. PROC TRANSPOSE will be used to convert the numeric statistic values to characters using the associated formats but we need to create macro variables to hold the FORMAT statement operands. The variable FMT0 holds the user assigned format for each variable with a little added to the W portion. This is just to insure that the width is enough to display the statistics. The variable FMT1 holds the user assigned format with 1 added to the D part for display of statistics with one more decimal place than the original data. Because the formatted table cells are ultimately displayed with center justification in the RTF output the W portion of the format is not important as long as it is wide enough to prevent any W.D format too small message.

```
data cvars;
   set cvars end=eof;
   vformatN = vformatNX(_name_);
   vformatD = vformatdX(_name_);
   vformatW = vformatWX(_name_);
   length fmt0 fmt1 $32.;
   fmt0     = cats(vformatN,max(12,vformatW+2),'.',vformatD);
   fmt1     = cats(vformatN,max(12,vformatW+2),'.',vformatD+1);
   return;
   set &data(keep=&cvars);
```

```
   drop &cvars;
   run;

%let FMT0 =;
%let FMT1 =;
proc sql noprint;
   select
      catx(' ',_name_, fmt0),
      catx(' ',_name_, fmt1)
      into
         :fmt0 separated by ' ',
         :fmt1 separated by ' '
      from cvars;
   quit;
   run;
%put NOTE: FMT0=&fmt0;
%put NOTE: FMT1=&fmt1;
```

The preceding two steps create the FORMAT statement operands shown in Figure 6

```
NOTE: FMT0=bmi F12.2 Age F12.0 Height F12.1 Weight F12.1
NOTE: FMT1=bmi F12.3 Age F12.1 Height F12.2 Weight F12.2
```

Figure 6 Format statement operands.

With statistic formats established we summarize the continuous variables and continue on to formatting the statistics for display. PROC SUMMARY the alter ego of PROC MEANS is chosen for speed and features. It requires no sorting and the PRELOADFMT option will provide output for all levels of the class variables even it they are not present in the data. The default output data set created by PROC SUMMARY has a data structure that suites further processing however it does not include all the statistics that we need. That is remedied with another output statement.

```
proc summary data=&data nway completetypes;
   class &trt / preloadfmt;
   var &cvars;
   output out=stats (drop=_type_);
   output out=median(drop=_type_) median=;
   run;
```

Combine the output from PROC SUMMARY into one data set and assign _STAT_ a value for the MEDIAN. A dummy character variable is created to be used with PROC TRANPOSE.

```
data stats;
   set stats median(in=in2);
   by &trt;
   if in2 then _STAT_ = 'MEDIAN';
   retain dummy '12345678';
   run;
```

The next PROC TRANSPOSE steps are used to format each statistic value according to the format associated with it. The technique involves transposing numeric variables along with a character variable, which produces a result that is very similar to VVALUE function. When numeric variables are transposed with a character variable the numeric variables are converted to character using any associated format or the default format BEST12. The technique is very powerful and allows many variables to be processed without knowing anything but the variable names; the same statements are used for 1 or 100 or more variables.

Each PROC TRANSPOSE subsets data STATS according to the statistics that belong to a formatting group. The statistic N is formatted as an integer, the MIN and MAX are formatted using the same format as the original data, and the MEAN, STD and MEDIAN are formatted with one more decimal place than the original data. You will recall the formats with the additional decimal places were calculated in data CVARS and the format statement operands were written to a macro variable.

```
proc transpose data=stats out=stat1;
   by &trt _stat_ notsorted;
   where _stat_ eq 'N';
```

```
   var &cvars dummy;
   format &cvars 12.;
   run;
proc transpose data=stats out=stat2;
   by &trt _stat_ notsorted;
   where _stat_ in('MIN','MAX');
   var &cvars dummy;
   format &fmt0;
   run;
proc transpose data=stats out=stat3;
   by &trt _stat_ notsorted;
   where _stat_ in('MEAN','STD','MEDIAN');
   var &cvars dummy;
   format &fmt1;
   run;
```

The summary statistics will need row labels to identify them. We also need to establish the order they are displayed and create groups for processing the statistics displayed in single cells; Mean (SD) and MIN, MAX. A FORMAT and two INFORMATS provide the needed objects to accomplish this.

```
proc format;
   invalue statord(upcase just) 'N'=1 'MEAN'=2 'STD'=3  'MEDIAN'=4  'MIN'=5 'MAX'=6;
   invalue statgrp(upcase just) 'N'=1 'MEAN','STD'=2  'MEDIAN'=3  'MIN','MAX'=4;
   value   statgrp               1='N' 2='Mean (SD)'  3='Median'  4='Min, Max';
   run;
```

With each of the three statistic group's values formatted and converted to character by PROC TRANSPOSE the data are recombined while removing the DUMMY character variable. New variables are created to facilitate further processing. VNAME is created from _NAME_ with length 32 to insure consistency. ROWORDER is created from _STAT_ using the STATGRP INFORMAT, ORDER is created to order the values of _STAT_ within ROWORDER for rows that display more than one statistic, and finally COL1 created above by the transposes is left justified.

```
data stats;
   length vname $32;
   set stat1 stat2 stat3;
   where upcase(_name_) ne 'DUMMY';
   attrib roworder length=8 label='Order';
   attrib order    length=8;
   Vname    = _name_;
   roworder = input(_stat_,statgrp.);
   order    = input(_stat_,statord.);
   col1     = left(col1);
   drop _NAME_ _LABEL_;
   run;
proc sort data=stats;
   by vname &trt roworder order _stat_;
   run;
```

At this point in the processing the continuous variables and the statistics for each of those variables are strung out into individual records as shown in Figure 7. The data has one record per variable per statistic per level of treatment. Notice that ROWORDER groups the MEAN and STD and the MIN and MAX.

Obs	vname	trt	_STAT_	COL1	roworder	order
1	Age	Placebo	N	13	1	1
2	Age	Placebo	MEAN	13.2	2	2
3	Age	Placebo	STD	1.5	2	3
4	Age	Placebo	MEDIAN	13.0	3	4
5	Age	Placebo	MIN	11	4	5
6	Age	Placebo	MAX	15	4	6
7	Age	Active	N	6	1	1
8	Age	Active	MEAN	13.7	2	2
9	Age	Active	STD	1.6	2	3
10	Age	Active	MEDIAN	13.5	3	4

11	Age	Active	MIN	12	4	5
12	Age	Active	MAX	16	4	6
13	Age	Total	N	19	1	1
14	Age	Total	MEAN	13.3	2	2
15	Age	Total	STD	1.5	2	3
16	Age	Total	MEDIAN	13.0	3	4
17	Age	Total	MIN	11	4	5
18	Age	Total	MAX	16	4	6
19	Height	Placebo	N	13	1	1
20	Height	Placebo	MEAN	61.87	2	2

Figure 7 A few observations from combined PROC TRANPOSE output.

The next step is to format the table cells. PROC TRANPOSE is used transpose to combine the observations for the statistics that are displayed together in one cell MEAN (STD) and MIN, MAX. The sorted data is transpose by VNAME &TRT ROWORDER. There are now two new variables COL1 and COL2 that contain the formatted statistic values as show in Figure 8.

```
proc transpose data=stats out=stats(drop=_name_);
   by vname &trt roworder;
   var col1;
   run;
```

Obs	vname	trt	roworder	COL1	COL2
1	Age	Placebo	1	13	
2	Age	Placebo	2	13.2	1.5
3	Age	Placebo	3	13.0	
4	Age	Placebo	4	11	15
5	Age	Active	1	6	
6	Age	Active	2	13.7	1.6
7	Age	Active	3	13.5	
8	Age	Active	4	12	16
9	Age	Total	1	19	
10	Age	Total	2	13.3	1.5
11	Age	Total	3	13.0	
12	Age	Total	4	11	16
13	Height	Placebo	1	13	
14	Height	Placebo	2	61.87	4.81
15	Height	Placebo	3	62.50	
16	Height	Placebo	4	51.3	69.0
17	Height	Active	1	6	
18	Height	Active	2	63.35	6.11
19	Height	Active	3	64.15	
20	Height	Active	4	56.3	72.0

Figure 8 Data from TRANSPOSE with statistics displayed in one table cell combined into one observation.

This next data step formats the table cells for the continuous variables according to the statistic group ROWORDER. Statistics contained in variables COL1 and COL2 that are displayed in one cell are concatenated or the value of COL1 is simply assigned to CELL. The labels for the statistics are created from the formatted value of ROWLABEL using the STATGRP format. IDLABEL is created from data BIGN and _LEVEL_ the generic form of the treatment values are looked up from data BIGN.

```
data stats;
   set stats;
   length cell $32 idlabel $256;
   select(roworder);
      when(2)     cell = catx(' ',col1,cats('(',col2,')')); /* mean(std)*/
```

```
      when(4)     cell = catx(', ',col1,col2);              /* min, max */
      otherwise   cell = col1;
      end;
   attrib rowlabel length=$64 label='Statistic';
   rowlabel = put(roworder,statgrp.);
   set bign key=&trt/unique;
   idlabel = catx('~',vvalue(&trt),cats('(N=',bign,')'));
   run;
```

The generic treatment values contained in the variable _LEVEL_ are ready to be transposed into variables named by with ID statement. The IDLABEL will provide variable labels for the variables created and named by the ID statement. These variable labels will become the column header labels when the data are displayed with PROC REPORT.

```
proc sort data=stats;
   by vname roworder rowlabel _level_;
   run;
proc transpose data=stats out=Continuous(drop=_name_) prefix=col_;
   by vname roworder rowlabel;
   var cell;
   id _level_;
   idlabel idlabel;
   run;
```

FINAL DATA PREPERATION STEPS

At this point in the processing both the categorical variable and continuous variables have been summarized and the summary statistics formatted for display. Then next step is to determine the order to display each variable down the page. This information is provided in the AVARS macro variable, or if AVARS is blank the order the variables appear in the input data is used to determine the variable order down the page.

AVARS was define above as

```
%let avars = sex race age ageg _page_ bmi height weight;
```

The special variable name _PAGE_ is used to mark page breaks in AVARS but this variable does not exist in the input data so a data step view is created to add the variable. Since no observations are needed the view is created with zero observations. Using the view the variables are transposed into a data set with one row for each name listed in AVARS. This is the same technique used to process DVARS and CVARS.

```
data avarsV / view=avarsV;
   stop;
   set &data;
   retain _PAGE_ 0;
   run;

proc transpose name=vname label=vlabel data=avarsV out=avars;
   var _PAGE_ &avars %sysfunc(ifC(%superQ(avars) eq,%nrstr(_all_),%nrstr()));
   run;
```

The output from PROC TRANSPOSE is used to create two new variables to be added to the combined data sets CONTINUOUS and DISCRETE. PAGE is an ORDER variable that PROC REPORT uses to produce page breaks. VORDER also an order variable is used to order the variables in PROC REPORT. The data set is indexed on VNAME so the information in this data can be retrieved easily using keyed data access. VLABEL the analysis variable label will be used in PROC REPORT to identify the variable.

```
data avars(index=(vname));
   set avars;
   if vname eq '_PAGE_' then do;
      page + 1;
      delete;
      end;
   vorder + 1;
   run;
```

The two summary data sets created above CONTINUOUS and DISCRETE are concatenated and the variable ordering and pagination information is looked up from AVARS. Data DISPLAY is the final data set ready for PROC

REPORT. The row labeling and order variables are defined with ATTRIB statement to assure they are ordered in the data set in a logical order form left to right. The variables are labeled to describe their function.

```
data display;
   attrib page       length=8
          label='Break variable used in PROC REPORT BREAK statement';
   attrib vorder     length=8
          label='Order variable used to order the
                 analysis variables, defined in PROC REPORT as ORDER NOPRINT.';
   attrib vname      length=$32  label='Analysis variable name';
   attrib vlabel     length=$256
          label='Analysis variable label, defined in PROC REPORT
                 as ORDER NOPRINT, displayed with LINE statement in COMPUTE block';
   attrib roworder   length=8
          label='Order variable for row labels, defined in PROC REPORT
                 as ORDER NOPRINT';
   attrib rowlabel   length=$256
          label='Detail row label, derived from categories or statistic labels';

   set Continuous(in=in1) Discrete(in=in2);
   set avars key=vname/unique;
   if in2 then vlabel = catx(', ',vlabel,'n(%)');
   run;
```

An example of this data set is given in Figure 3.

ODS RTF SETUP

The output is produce as an RTF file which seems to be popular, in the pharmaceutical industry. The options statement specifies the page layout, portrait, and turns off the default page numbering and date. Date and page number are handled with RTF options. The center option is used to unsure the table is centered on the page.

```
options
   Orientation =  Portrait
   Date        =  0
   Number      =  0
   Center      =  1
   ;
```

The SAS system options for margin settings are not used for certain style templates in SAS Version 9.1.3. I believe this have been changed in version 9.2. This bit of PROC TEMPLATE code creates a new style template from the Journal style and sets the margins.

```
ods path temp(update) sashelp.tmplmst(read);
proc template;
   define style statsinrows;
      parent=Styles.Journal;
         style body from document /
         leftmargin     =1.5in
         rightmargin    =1  in
         topmargin      =1  in
         bottommargin   =1  in;
      end;
   run;
```

Turn off the LISTING destination, set the ODS escape character and open the RTF destination using the path and filename we derived from the program above.

```
ods listing close;
ods ESCAPECHAR = '!';
ods rtf file="&path.rtf" style=statsinrows;
```

TITLES AND FOOTNOTES

Usually TITLES and FOOTNOTES are the domain of some type of meta data system, there is usually a program that accepts as input an identifier and the program produces title and footnote statements. In this example we will include them as follows.

```
title1 j=Left   h=10pt 'ABC Pharmaceutical' j=r 'Page !{pageof}';
title2 j=left   h=10pt 'Protocol: ABC2010-103, A Phase III Study';
title3 j=center h=14pt 'Table 14-2.1';
title4 j=center h=14pt 'Summary of Demographic Characteristics at Baseline';
footnote1 j=left h=8pt "&path.sas" j=right "%sysfunc(datetime(),datetime)";
```

PROC REPORT

The data is displayed with a generic PROC REPORT. This PROC REPORT step will remain unchanged with additional analysis variables or additional levels of the treatment variable. Notice that there is no reference to any variable that existed in the input data. All analysis variable names are now values of variables owned by the program (VORDER, VNAME and VLABEL). Similarly we control the levels of the categorical variables and the summary statistics computed for the continuous variables with ROWORDER and ROWLABEL. The treatment variable has been converted to a format we control in the COL_n enumerated variable list.

The variables ROWLABEL and COL_: are the only variables that do not have the NOPRINT attribute. The pretext style attribute is used to indent the values of ROWLABEL variable because VLABEL will share this column. The indentation will highlight the subordinate relationship of ROWLABEL to the variable description in VLABEL. A COMPUTE block and LINE statements are used to produce a blank line and display the value of VLABEL.

Notice there is only one DEFINE statement for the COL_: variables. PROC REPORT allows the use of the SAS Variable List in the DEFINE statement we take advantage of that and define all the treatment column variables using one DEFINE statement. Pagination is controlled with a BREAK statement using the ORDER variable PAGE.

```
proc report list nowd missing data=display split='~';
   column page vorder vname vlabel roworder rowlabel
          ("!R'\brdrb\brdrs\brdrw1' &TRTLabel" col_1-col_%eval(&col0-1)) col_&col0;
   define page       / order order=internal noprint ' ';
   define vorder     / order order=internal noprint ' ';
   define vname      / order order=internal noprint ' ';
   define vlabel     / order order=internal noprint ' ';
   define roworder   / order order=internal noprint ' ';
   define rowlabel   / display ' '
      style(column)=
         [just=left cellwidth=39% protectspecialchars=off pretext="\li240 "];

   define col_:      / display style(column)=[just=center cellwidth=&trtPercent.%];

   break before page / page;
   compute before page;
      f = 0;
      endcomp;
   compute before vlabel / style=[just=left];
      blankline = ' ';
      if f eq 0 then do;
         l1 = 0;
         f  = 1;
         end;
      else l1 = 1;
      l2 = length(vlabel);
      line blankline $varying10.-l l1;
      line vlabel $varying200.-l l2;
      endcomp;
   run;
ods rtf close;
ods listing;
```

CONCLUSION

The program presented here demonstrates that a dynamic program to produce clinical trials tables does not need to be a complicated macro. This is because very little code generation is required if the proper SAS PROCS and other features are used. In other words most of the processing we might have thought needed macro language can be done easily using standard SAS code. The program is reasonably efficient it processes all categorical variables in one group as well as processing the continuous variables together in a group. Additional summary statistics could easily be added and the display styles could be modified as needed. Any reasonable number of variables and levels of categorical variables can be used. The code is simple requiring no library of macros and using no macro language feature other than a few macro variables. The code is obvious and transparent; programs written using this or similar code require no complex documentation or support.

REFERENCES

- Problem Note *5468:* ODS RTF ignores LEFTMARGIN, RIGHTMARGIN, TOPMARGIN, and BOTTOMMARGIN system options http://support.sas.com/kb/5/468.html
- Usage Note *24033:* How can I set the margins for the ODS RTF destination? http://support.sas.com/kb/24/033.html
- SAS Variable Lists http://support.sas.com/documentation/cdl/en/lrcon/61722/HTML/default/a000695105.htm

CONTACT INFORMATION

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

John Henry King
Ouachita Clinical Data Services, Inc.
24 Loblolly Circle
Mount Ida, AR 71957
501-351-0432
ouachitaclinicaldataservices@gmail.com