THE CREATIVE USE OF THE PUT STATEMENT TO IMPROVE PRODUCTIVITY AND FLEXIBILITY IN TABLE GENERATION

Sam X. Ye
Whitehall-Robins, A.H.P.Co.

ABSTRACT

The PUT statement in data steps is a primary tool to generate tables which have complicated headings and structures. The plain use of this statement, however, could result in tedious column positioning, difficult modification and lengthy programs. This paper introduces three short macros (called the PUT macros) as a general tool to 'automatically' generate positions and PUT statements for placing strings or variables into any table. By changing a few numbers in a macro, the user is able to move a table around to achieve the desired result with ease and enjoyment. The PUT macros can be an effective programming tool to create more powerful table generating macros and programs. The concept, the programs, the usage and the applications of the PUT macros will be discussed.

INTRODUCTION

Because of the quality requirements on final statistical reports and customized data listing, generating these tables depends heavily on the PUT statement. The most tedious aspect of using the PUT statement is positioning every string and every variable into the right place in the table. This process usually involves planning, calculating and typing dozens of position numbers. Whether in the middle of creating or after generating a table, frequent modification of the positions to get the desired appearance is a common practice. Lots of time is spent in replanning, recalculating, correcting and re-correcting the position numbers.

Using position variables instead of position numbers can ease many of the problems. This paper expands this idea and develops three macros (PUTPOS, PUTVAR, and PUTCNT - I refer to them as the PUT macros) to simplify generation of tables. Comments would be greatly appreciated.

A SIMPLE EXAMPLE

Consider the following data set named TESTDAT:

<table>
<thead>
<tr>
<th>OBS</th>
<th>NAME</th>
<th>M1</th>
<th>S1</th>
<th>M2</th>
<th>S2</th>
<th>PVAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HOUR</td>
<td>2.6</td>
<td>1.6</td>
<td>1.3</td>
<td>1.4</td>
<td>.001</td>
</tr>
<tr>
<td>2</td>
<td>HOUR</td>
<td>3.2</td>
<td>1.6</td>
<td>1.7</td>
<td>1.7</td>
<td>.001</td>
</tr>
</tbody>
</table>

The table we want to generate from the data set looks like this:

PHARMACEUTICAL Co.
PROTOCOL NUMBER:

TABLE A

<table>
<thead>
<tr>
<th>Relief Variable</th>
<th>ACTIVE Mean</th>
<th>PLACEBO Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STD</td>
<td>STD</td>
</tr>
<tr>
<td>1HOUR</td>
<td>2.6</td>
<td>1.6</td>
</tr>
<tr>
<td>2HOUR</td>
<td>3.2</td>
<td>1.6</td>
</tr>
</tbody>
</table>

The following SAS program uses PUT statements to achieve this goal:

```sas
option nodate nonumber LS=78;
title;
data null;
set testdat;
file print header='b';
put @IS name @31 m1 +3 51 @47 m2 +3 s2 @63 pval;
return;
H:
put @15 'PHARMACEUTICAL Co.';
put @15 'PROTOCOL NUMBER:';
put;
put @35 'TABLE A';
put;
put @31 +2 'ACTIVE @47 +1 PLACEBO';
```
Note that:
(1) The position numbers after @ must be carefully planned and calculated to fix the paper (or font) size and column spacing.
(2) 21 position numbers (+2 or +3 are not counted to simplify the comparison) are scattered throughout the program. If there is any change in margin, column spaces, adding or dropping of variables, part or all numbers have to be recalculated and the programmer will have to find and correct them.
(3) Many numbers are repeated. In this case there are only 5 numbers which are different.

With the use of the PUT macros introduced in this paper, the following program can produce the same table:

```
OPTIONS nodate nonumber mprint mautosource
sasautos='the name of directory where the macros are stored';
* see last section for explanation of sasautos ;
data_null ;
set testdat ;
file print header=h;
%PUTPOS(15 16 16 16);
%PUTVAR( name! ml +3 sl! m2 +3 s2! pval) ;
put ;
return;
run;
```

H:

```
%PUTVAR('PHARMACEUTICAL Co.');
%PUTVAR('PROTOCOL NUMBER');
%PUTCNT(78,'TABLE A') ;
put ;
%PUTVAR( '! +2 'ACTIVE! +1 'PLACEBO');
%PUTVAR( 'RELIEF! 10* ! +1 10*! =');
%PUTVAR( 'VARIABLE! 'MEAN' +3 'STD'!
'MEAN' +3 'STD'! 'P-VALUE' );
%PUTVAR('-----!' ' --- +3 ' ---!
' --- +3 ' --- ! ' ---' );
put/ ;
return; run;
```

(1) This program basically corresponds to the former program line by line except that: (a) Instead of PUT, the %PUTVAR or %PUTCNT are used. (b) All @numbers have been removed and ! is used to separate the variable(string) groups. There also is one more macro %PUTPOS(15 16 16 16) to define the positions of columns.

(2) It is easy to start generating a table. No careful plan or calculation is needed. Choose 4 numbers in the PUTPOS macro, the first number to set the start position and the other 3 numbers for column spacing. Run the program to see the table, and adjust the numbers to get the best visual result.

(3) It is easy to modify a table. For changes such as adding or dropping variables, changing column spaces or paper size, all you need for positioning is the manipulation of those few numbers at a single place in the PUTPOS macro.

(4) Further simplification and generalization can be achieved. See application section C.

THE CONCEPT AND THE PROGRAMS OF THE PUT MACROS

Among the three PUT macros (PUTPOS, PUTVAR, and PUTCNT), the PUTPOS macro and the PUTVAR macro are connected with each other and play the primary role in making the program work.

The PUTPOS(POS) macro defines the column positions that will be used by the subsequent PUTVAR macros. The parameter POS contains numbers separated by a space.

As an example, the invocation of the macro %PUTPOS(15 16 16 16) will produce the following SAS codes:

```
_p1=15;
_p2=_p1+16;
_p3=_p2+16;
_p4=_p3+16;
```

The PUTVAR(VAR) macro put variables or strings at positions previously set up by the PUTPOS macro. Parameter VAR contains variable or string groups separated by ! like (varname! mean1! mean2). The number of variable (or string) groups can be less than or equal to the number of positions in PUTPOS.

The following two lines show an example of a call of the PUTVAR macro and the SAS code it produced:

```
%PUTVAR(var1 3. ! +2 var2 $5. ! var3)
```
produces
PUT @p1 var1 3. @p2 +2 var2 $5. @p3 var3

As shown in this example, the PUTVAR macro can be used in a similar way as the PUT statement except that it uses ! (instead of @position-number) to identify the column positions for the variables (or strings). The key point of saving efforts of positioning is based on the fact that a table always has a column structure. Following the column pattern of a table, we can use the PUTPOS macro to define the starting positions of the columns, and then use the PUTVAR macro to relate the position of any variable or any string with one of these columns. We also can use +n or +(-n) to adjust positions for the variables (strings) in a given column. In this way we are able to move all column position numbers over the program to a single place that is inside the PUTPOS macro. Thus, the tedious work of position planning, calculating, and correcting can be minimized to manipulating those numbers inside the PUTPOS macro.

The PUTCNT(LSIZE, TEXT) macro puts TEXT in the center of the output file which has the line size of LSIZE. As an example, the call of the macro %PUTCNT(130, Table A) will produce the following SAS code:

PUT @61 'Table A'

This simple macro is convenient for putting a title in the table. It stands alone and has no connection with the PUTPOS macro.

Note that macros PUTVAR and PUTCNT do not produce semicolon. The user can put any other components of the put statement after the call of the macros and add semicolon thereafter. For example, the next two lines are the same and will allow you to overprint and get bold name:

%PUTVAR(name! m1! m2! pval) overprint @p1 name;
%PUTVAR(name! m1! m2! pval overprint @p1 name);

Both of above lines are equivalent to the following put statement:

PUT @p1 name @p2 m1 @p3 m2 @p4 pval
overprint @p1 name;

The programs of the PUT macros are provided below.

(1) The PUTPOS macro

%MACRO PUTPOS( POS );

%GLOBAL _NPOS;
%LOCAL I J;

%* The next counts the total number of positions in POS
%* and assigns it to a global macro variable _NPOS *;
%LET I=1;
%DO %WHILE (%SCAN(&POS, &I) NE ) ; %LET I=%EVAL(&I+1) ;
%END;
%LET _NPOS=%EVAL(&I-1) ;

%* The next creates SAS code to calculate and assign
%* position values to variables _P1 - _Pn which will be
%* used in macro PUTVAR *;
_P1=%SCAN(&POS, 1) ;
%DO I=2 %TO &NPOS ;
%LET J=%EVAL(&I-1) ;
_P&I=_P&J+%SCAN(&POS, &I) ;
%END;
%AMEND PUTPOS;

(2) The PUTVAR macro

%MACRO PUTVAR( VAR ) ;
%GLOBAL _NPOS;
%LOCAL I NVAR ;

%% The next counts the total number of text groups
%% separated by ! in VAR and assign it to a macro
%% variable NVAR **;
%LET I=1 ;
%DO %WHILE (%LENGTH(%SCAN(&VAR, &I,!))
NE 0 ) ; %** the %LENGTH here is necessary;
%LET I=%EVAL(&I+1) ;
%END;
%LET NVAR=%EVAL(&I-1) ;

%% The next creates the PUT statement using
%% variables _P1 - _Pn to position strings or
%% variables(given by VAR) and do error checking ;
%IF &NVAR <= & NPOS %THEN %DO ;
PUT %DO I=1 %TO &NVAR ;
@ _P&I %SCAN( &VAR, &I,!)
%END;
%END ;
%ELSE %DO ;
Error 'Error Message from the PUTVAR macro: # of'
'positions < # of variables OR required %PUTPOS'
'macro did not be used prior %PUTVAR';
%END;
%AMEND PUTVAR;

%GLOBAL _NPOS
(3) The PUTCNT macro

%MACRO PUTCNT(LSIZE, TEXT);
  PUT @((&LSIZE-%LENGTH(&TEXT))/2) &TEXT
%MEND PUTCNT;

THE APPLICATION OF THE PUT MACROS

The PUT macros bring a new and effective tool for programming in table generation. Below are some examples.

A. Table Head Floating

The report of a clinical study contains dozens of statistical and data listing tables. These tables often have the same study heading such as company name, protocol number, investigator's name and protocol title. It would be more efficient to create a head macro (or a few head macros) to set the heading and have all table generating programs use it. The difficulty here is that the starting position of the heading should be consistent throughout different tables.

PUTVAR (and PUTCNT) provide a good solution to the creation of such head macros because the starting positions of PUTVAR are floating automatically with the PUTPOS that is given in the table generating programs. The following is an example of the head macro:

%MACRO HD1234;
  %PUTVAR('PHARMACEUTICAL CO.');
  %PUTVAR('PROTOCOL #: 1234');
  %PUTVAR('INVESTIGATORS:');
  %PUTCNT(176,' COMMON TITLE');
%MEND;

All (or part) of the tables of a study can share this head macro. The position of the first three lines is p1 which will be defined in the program wherever the head macro is called. This will guarantee the consistence of the report tables of a study.

B. Multi-tables Setting

Consider the example in the section "A SIMPLE EXAMPLE" again. Suppose we want to create a macro which can generate different tables automatically depending on the number of drug treatments. This scenario involves several table settings and regular methods usually cause a lot of confusion when modifications are being made. PUTPOS and PUTVAR provide a nice framework to make a program more concise and easier to modify. The basic idea is to use a single table generating procedure to achieve several table outcomes.

Suppose the necessary data are provided in TESTDAT with variables M1 - Mn and S1 - Sn depending on the number of treatments given in the macro variable N1RT. Below is an example of the macro program which can generate two tables, one for 2 treatments (N1RT=2) and one for 3 treatments (N1RT=3):

%MACRO TAB(N1RT=2);

%* set macro variables for N1RT=2;
%let vars = m1 +3 s1! m2 +3 s2 !
%let head1= 'ACTIVE1'!
%let head2= 10*'=!
%let head3= 'MEAN' +3 'STD'! 'MEAN' +3 'STD'!
%let head4= '-' +3 '-'! '-' +3 '-'!

%* set macro variables for N1RT=3;
%if &N1RT=3 %then %do;
%let vars = &vars m3 +3 s3 !
%let head1= &head1 'ACTIVE2'!
%let head2= &head2 10*'=!
%let head3= &head3 'MEAN' +3 'STD'!
%let head4= &head4 '-' +3 '-'!
%end;

%* using a single table generating procedure;
data _null_; set testdat;
  file print header=h;

%* define table columns for different N1RT;
%PUTPOS(15 16 16 16);
%if &N1RT=3 %then %do;
%PUTPOS(7 12 15 15 15);
%end;

%** put variables;
%PUTVAR(name! &vars! pval)
return;

h:
%** put variable headings;
%PUTVAR(' ' ! &head1 +1 'PLACEBO');
%PUTVAR('Relief ' ! &head2 10*'=');
%PUTVAR('Variable'! &head3 'P-value');
%PUTVAR('------!' &head4 '------');
return;
%MEND;
The invocation of %TAB generates exactly the same table as TABLE A, and the invocation of %TAB(NTRT=3) produces similar table but has 3 treatment columns (ACTIVE1, ACTIVE2, and PLACEBO).

As stated before, the user needs only manipulate those few numbers in PUTPOS to modify these two tables. It is easy to expand this macro to handle NTRT=4 or more without changing this single "data _ null_" structure.

C. Expanding the Power of Table Generating Macros

With the help of the PUT macros, writing a macro to generate tables becomes much easier, and the power of the table generating macros can be surprising. For example, it is easy to write a table generating macro with the following functions: the user can freely set or change the number and the positions of variables, the names of variables, the table titles, the variable headings, and the footnotes just by assigning the parameters of the macro. The following macro is an example for simple data listing:

%MACROTABSL(DATA=_LAST_, POS==, VARS=, STDHDL=, TABHD1=, TABHD2=, TABHD3=, TABHD4=, TABHD5=, ARHD1=, VARHD1=, VARHD2=, VARHD3=, VARHD4=, VARHD5=, FOOIN1=, FOOIN2=, FOOIN3=, FOOIN4=, FOOIN5=);
%local j ;
data _null_; set &data end=last ;
file print header=h;
%PUTPOS(&POS); %PUTVAR(&VARS)/;
if last then do ; put//;
  %do j=1 %to 5 ;
    %if %length(&&FOOTN&j) ne 0 %then
      %PUTVAR(&&FOOTN&j) %str(();
  %end; end; return;

h: %&STDHDL ; %do j=1 %to 5 ;
  %if %length(&&TABHD&j) ne 0 %then
    %PUTCMT(78, &&TABHD&j) %str(();
  %end;
%do j=1 %to 5 ;
  %if %length(&&VARHD&j) ne 0 %then
    %PUTVAR(&&VARHD&j) %str(();
  %end;
  put //;
return;
%MEND TABSL;

This simple macro is quite flexible. Any straightforward data listing table can be done by a single call of the macro with the power of choosing any variables, headings, footnotes, and the ability of modifying the column positions. For example, the TABLE A can be generated by an invocation of the TABSL macro:

%TABSL(DATA=testdat, POS=15 16 16 16, VARS=name! m1 +3 s1! m2 +3 s2! pval, TABHD1=TABLE A', VARHD1= ' +2 'ACTIVE1' +1 'PLACEBO', VARHD2= 'RELIEF ! 10**'=1! 10**-=, VARHD3= 'VARIABLE! MEAN +3 'STD'! MEAN +3 'STD'! P-VALUE', VARHD4= '-----'! -----+3 '-----'! -----+3 '-----'! -----');

D. Using Line Pointer (#) in the PUTVAR Macro

The PUTVAR macro is very simple and effective, but if you need to use line pointer (#) inside the macro, a modification should be made. For example, assume following PUT statement is wanted:

PUT #40 @_p1 VAR1 @_p2 VAR2 @_p3 VAR3;

To obtain this statement, the macro call should be read as:

%PUTVAR( #40 @_p1 VAR1 ! VAR2! VAR3);

In other words, you need to insert an additional @_p1 between #40 and VAR1. It will produce:

PUT @_p1 #40 @_p1 VAR1 @_p2 VAR2 @_p3 VAR3;

The PUT statement here has the same results as the desirable one. A modified version of the macro named PUTVAR_M can handle line pointer(#) better. The invocation of

%PUTVAR_M( #40 VAR1! VAR2! VAR3);

will produce the required PUT statement. No insertion is needed. This macro is more complicated and less efficient if no line pointers are used. The autocall library supplied by SAS Institute should be available in the system (because the %VERIFY macro is used in this version).

The PUTVAR_M macro is shown below.

%MACROTABSL VAR );
%global __npos;
%local i nvar c len fig;
%LET I=1;
%DO %WHILE (%length(%SCAN(&VAR, &I, !)) NE 0);
  %LET I=%EVAL(&I+1);
%END;
%LET NVAR=%EVAL(&I-1);
%IF &NVAR <= &__NPOS %THEN %DO;
  %LET fig=0;
  %LET c1=1;
  %LET c2=1;
%END;
%** search for a #;
%do %until(&c1=%length(&var) or &fig=1);
  %if "%substr(&var, &c1, 1)="" %then %do;
    %let c1=%eval(&c1+1);
    %let c2=%verify(%substr(&var, &c1),
       '1234567890 ');
    %let len=%eval(&c2-&c1);
    %let c1=%eval(&c1-1);
    %let fig=1;
  %end;
%else %do;
  %let c1=%eval(&c1+1);
%end;
%end;
%** if there is a #, then do following;
%if &fig=1 %then %do;
  PUT
  %substr(&VAR, &c1, &len) @_PI
  %SCAN(%substr(&VAR, &c2, 1, !))
%DO I=2 %TO &NVAR;
    @ _P&I %SCAN( &VAR, &I, !)
%END;
%end;
%** if there is no #, then do following;
%if &fig=0 %then %do;
  PUT
  %DO I=1 %TO &NVAR;
    @ _P&I %SCAN( &VAR, &I, !)
%END;
%end;
%ELSE %DO;
  Error '!! Error Message from Macro PUTVAR: # of ' 'positions < # of variables OR required macro'
  '%PUTPOS did not be used prior %PUTVAR';
%END;

%MEND PUTVAR_M;

ADDITIONAL NOTES

A. The Method of Making The Macros Available.

The best way to make macros ready to be called is to utilize the SAS macro autocall facility. For example, under UNIX or VMS system this can be done as shown below.

Place each macro separately in a SAS file with the same name as the name of macro it contains. Keep these files in a directory and inform this directory to the SAS system by setting MAUTOSOURCE and SASAUTOS= options.

If you set these options in your autoexec.sas file in your home directory you will be able to call these macros in any of your programs. You also can set these options in the program in which you intend to call these macros if you want other users to run that program.

The following are examples of the option setting.

Under UNIX operating system:
OPTIONS mautosource sasautos="/user/maclib";

Under VMS operating system:
OPTIONS mautosource sasautos="[group:user.maclib];";

The other way to make macros ready to use is to place all macros in a single file and then include it in the program using %INC.

B. The PUTVAR macro uses ! as a delimiter because ! can not be a part of a string to be included in a table in my working environment. If the user does need ! in a string, then the usual PUT statement should be used with appropriate _p position variables or the user can change the macro PUTVAR and choose another character they do not use often as a delimiter.

CONCLUSION

The PUT statement in data steps is a primary tool to generate complicated tables. The plain use of this statement, however, could result in tedious column positioning, difficult modification and lengthy programs. The PUT macros introduced in this paper can ease these problems and can be an effective programming tool to create more powerful table generating macros and programs.
REFERENCES


ACKNOWLEDGEMENTS

Special Thanks to Yingcong Zheng, Helen-Jean Talbott, and Elizabeth C. Foulds for valuable comments and suggestions.

SAS is a registered trademark of SAS Institute Inc. Cary, NC, USA. Other brand and product names are registered trademarks or trademarks of their respective companies.

AUTHOR CONTACT

Sam X. Ye
Whitehall-Robins Inc. AHP Corp.
Dept. of Clinical Research
5 Giralda Farms
Madison, NJ 07940