A technique for generating repetitious SAS® code (such as format, label, and rename lists) is presented through three examples of increasing complexity. The more experienced macro programmer will recognize the wide applicability of the method to many of his own problems involving repetitious lists. The novice macro programmer will increase his understanding through seeing basic macro coding tools (SYMPUT subroutine and macro variable arrays) applied to a class of related problems.

Usually macros should be used to manipulate the SAS data set output by a statistical procedure because the same processing is done many times with only minor modifications. The technique presented in this paper helps the programmer to formalize the development of this type of macro. Our third example uses PROC CORR to illustrate this process.

INTRODUCTION

Suppose a recurring job consists of a series of steps, say, a DATA step followed by a PROC, then another DATA step to process the output of the PROC. Each time the job is done, it uses different data sets with different variable names. The technique presented in this paper can be used to build a system through the use of the macro facility to generate the programs automatically and avoid modifying code.

The method consists of three steps:

1) Obtain variable names from the output data set of PROC CONTENTS.
2) Use the SYMPUT subroutine in a data step to convert variable names, etc., to values of macro variables.
3) Generate the SAS code, using one or more %DO loops, containing the macro variables.

Note:
The technique, as explained in this paper, depends on the OUT option of PROC CONTENTS, available in Version 5 of the SAS system. Anyone using an earlier release may write to the authors for a macro to produce the needed data set from PROC CONTENTS.

The first two examples presented below are relatively simple and intended to make clear the three steps of the technique. The third is an example of a system for combining DATA and PROC steps, where code modifications are done automatically. The Conclusion mentions some other useful applications.

Some features of the macro facility which are demonstrated include:

1) The SYMPUT subroutine.
2) Construction of macro variable names and values through concatenation.
3) The use of the double ampersand in macro variable arrays.
4) The %DO loop.

EXAMPLE 1

We have two data sets with the same variables. One has format information that we want to add to the other. For example, let JUL85 be a member of the library with libref CHILD.

<table>
<thead>
<tr>
<th>OBS</th>
<th>CHILD</th>
<th>AGE</th>
<th>HEIGHT</th>
<th>WEIGHT</th>
<th>NUTRIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LCH</td>
<td>9</td>
<td>51</td>
<td>60</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>MEB</td>
<td>7</td>
<td>46</td>
<td>47</td>
<td>4</td>
</tr>
</tbody>
</table>

Each of the variables has been assigned a format. The following code would add the format information:

```sas
PROC DATASETS DDNAME = CHILD ;
MODIFY AUG85 ;
FORMAT
CHILD $FULLNAM.
AGE AGE_CLS.
HEIGHT HT_CLS.
WEIGHT WT_CLS.
NUTRIT NUTR_CLS.
;
RUN ;
```

The code is simple but repetitive. It would be very tedious if we had 100 variables and many data sets to run on. Our goal is to obtain this code automatically. The process is described as three separate steps for simplicity. But, wherever this technique can be applied, you will find three similar steps combined in a macro.

Step 1

We must first obtain the variable names and formats on JUL85 as SAS data. This can be done using the OUT option of PROC CONTENTS:

```sas
PROC CONTENTS DATA = CHILD,JUL85 OUT = DRCTRY NOPRINT ;
RUN ;
```

The resulting data set DRCTRY has the variable NAME, TYPE, and FORMAT (among others). For our example, we obtain:
Numeric variables are TYPE = 1, and character variables are TYPE = 2. TYPE is needed in order to prefix character formats with '$'. Note the suffix '.' is needed for all formats.

Step 2

The SAS data obtained in step 1 must be converted into macro variable values so that it can be manipulated as code. The SYMPUT subroutine is used in the data step below to accomplish the task.

```sas
DATA NULL
RETAIN N 0
IF EOFC THEN
  CALL SYMPUT ( 'N' , LEFT(PUT(N,4.)) )
SET DRCTRY END = EOFC
IF FORMAT = 1 THEN DELETE
N + 1
CALL SYMPUT ( 'NAME' II LEFT(PUT(N,4.)) , NME )
IF TYPE = 2 THEN
  CALL SYMPUT ( 'FMT' II LEFT(PUT(N,4.)) , TRIM(FORMAT) II '.' )
  CALL SYMPUT
RUN
```

The macro variable names are generated by concatenating:

1) 'NAME' or 'FMT'
2) the character equivalent of the counter N.

The left justification, LEFT(PUT(N,4.)) is required in making the macro variable name because it cannot have any spaces. The TRIM function prevents spaces in the format name. For safety, the left justification is also done in the end of file call, creating the macro variable N, because this macro variable is often used in macro concatenation in the third step. It is important to place a RUN after this data step in order to force the data step to execute before any reference to the macro variables created during the step's execution.

The result of the data step is the set of macro variables:

```
NAME1 - NAME5 and FMT1 - FMT5
```
Again we find that the code has the basic characteristics -- simplicity and repetition. The repetitious part of the program can be generated automatically using the same technique illustrated in the first example:

1) Obtain the variable names from SAS data sets using the OUT option in PROC CONTENTS.
2) Create macro variable to hold the variable names.
3) Use one or more %DO loops to generate the repetitious code.

For the macro to solve the general problem in this example, see Figure 2. For the data shown, the call would be:

%MERGDAT (CHILD, JUL85, AUG85, JULAUG85, CHILD)

EXAMPLE 3

We needed to present correlation coefficients, output by PROC CORR, ranked in numerically descending order for each variable in the VAR statement. The RANK option could not be used since it orders by absolute value. Since many runs of proc CORR were required for the project, we applied the technique described in this paper.

Example

Typical correlation data were produced by code similar to:

PROC CORR DATA = EXAMPLE OUTP = CORROUT NOPRINT;
VAR U X W;
WITH X Y Z W;
RUN;

The VAR statement determines the variables that will become actual variable names in the output data set, CORROUT. The WITH statement determines the variables that will become values of the variable _NAME_ in CORROUT:

<table>
<thead>
<tr>
<th>OBS</th>
<th><em>TYPE</em></th>
<th><em>NAME</em></th>
<th>U</th>
<th>X</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MEAN</td>
<td></td>
<td>4.5</td>
<td>3.39527</td>
<td>64.6875</td>
</tr>
<tr>
<td>2</td>
<td>STD</td>
<td></td>
<td>2.88675</td>
<td>4.23991</td>
<td>40.6201</td>
</tr>
<tr>
<td>3</td>
<td>N</td>
<td></td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>CORR</td>
<td>X</td>
<td>0.600932</td>
<td>1</td>
<td>0.773511</td>
</tr>
<tr>
<td>5</td>
<td>CORR</td>
<td>Y</td>
<td>-0.973168</td>
<td>-.555179</td>
<td>-.941597</td>
</tr>
<tr>
<td>6</td>
<td>CORR</td>
<td>Z</td>
<td>0.95847</td>
<td>0.597878</td>
<td>0.951125</td>
</tr>
<tr>
<td>7</td>
<td>CORR</td>
<td>W</td>
<td>0.956971</td>
<td>0.773511</td>
<td>1</td>
</tr>
</tbody>
</table>

The macro, RANKCORR, in Figure 3, uses this data set to produce the SAS data set RANKCOEF, shown below:

<table>
<thead>
<tr>
<th>OBS</th>
<th>NAME1</th>
<th>NAME2</th>
<th>COEFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>U</td>
<td>Z</td>
<td>0.95847</td>
</tr>
<tr>
<td>2</td>
<td>U</td>
<td>W</td>
<td>0.95697</td>
</tr>
<tr>
<td>3</td>
<td>U</td>
<td>X</td>
<td>0.600932</td>
</tr>
<tr>
<td>4</td>
<td>U</td>
<td>Y</td>
<td>-0.97317</td>
</tr>
<tr>
<td>5</td>
<td>W</td>
<td>Z</td>
<td>0.951125</td>
</tr>
<tr>
<td>6</td>
<td>W</td>
<td>X</td>
<td>0.77351</td>
</tr>
<tr>
<td>7</td>
<td>W</td>
<td>Y</td>
<td>-0.94160</td>
</tr>
</tbody>
</table>

Producing RANKCOEF requires getting three categories of information from CORROUT:

1) The 'VAR' variable names, which are obtained from the output data set of PROC CONTENTS, run on CORROUT.
   (We kept track of these in Step 3 in an array called _NAME_, and assigned the values in a DO OVER loop to a variable called NAME1.)
2) The 'WITH' variables, obtained by reading the values of _NAME_ for each record of CORROUT.
   (We renamed _NAME_ to NAME2 to make it parallel to NAME1.)
3) The coefficients, which are read and output, creating as many records in RANKCOEF as are needed to list the correlations as they appear there.
   (We created an array _COEFFS_ for the coefficients, then read and output them in a DO OVER loop.)

Our example includes a WITH statement but the macro RANKCORR shown in Figure 3 can be called with the parameter WITH=NO to allow for no WITH statement. In this case the 'WITH' and 'VAR' lists match so that the upper and lower triangles of coefficients are equal. Thus we chose to output only the lower triangle. This was accomplished by a RETURN in the DO loop introduced by the macro variable RSWITCH.

The macro call to produce the data set RANKCOEF was:

%RANKCORR(DATA = CORROUT, OUT = RANKCOEF)

CONCLUSION

Formalization of the technique described in this paper has aided us to:

1) Recognize where the technique should be applied.
2) Standardize the application code.
3) Speed up the development of an application.

Count files from PROC FREQ provide a natural source to apply the technique. In particular, the difficulty of reading Nth order cross tabs for N > 2 makes it desirable to produce specially tailored reports. Estimate files from PROC REG are another source for the technique. We used it to produce COBOL file descriptions for implementing regression formulas in a COBOL environment. Still another application arises when using PROC FSEDIT in a user friendly system to present a list of variables for selection.
The authors were working at Reader's Digest Association when they wrote this paper. Comments are welcome. For additional information write to:

H. Ian Whitlock
ORI, Inc.
122 C Street, N.W.
Washington, D.C. 20001

SAS is a registered trademark of SAS Institute Inc., Cary, NC, USA.

/* Figure 1 */

%MACRO GETFMTS (LIB, FMISORCE, FMTADD);
%*-------------------------------------------------------------------------------------;
%* MACRO PARAMETERS ARE, IN ORDER: THE LIBREF, THE MEMBER WITH FORMATS, AND THE MEMBER TO WHICH THE MACRO WILL ADD FORMATS;
%*-------------------------------------------------------------------------------------;
%*-------------------------------------------------------------------------------------;
%* GET VARIABLE NAMES---------------------------------------------------------------------*/
PROC CONTENTS DATA = &LIB FMISORCE OUT = DRCTRY NOPRINT;
RUN;
%* ONE FOR MACRO DELIMITER
%* AND ONE TO GO BETWEEN LIBREF AND MEMBER NAME
/*--------------------CREATE MACRO VARIABLES------------------------------------------*/
DATA _NULL_;
RETAIN NO;
IF EOFC THEN
   CALL SYMPUT ('N', LEFT(PUT(N,4.)));
SET DRCTRY END = EOFC;
IF FORMAT = ' ' THEN DELETE;
N + 1;
CALL SYMPUT ('NAME' II LEFT(PUT(N,4.)), NAME);
IF TYPE = 2 THEN
   CALL SYMPUT ('FMT' II LEFT(PUT(N,4.)), TRIM(FORMAT) II '.');
ELSE
   CALL SYMPUT ('FMT' II LEFT(PUT(N,4.)), TRIM(FORMAT) II ' ');
RUN;
/*-------------------------------GENERATE SAS CODE---------------------------------------*/
PROC DATASETS DDNAME = &LIB;
MODIFY &FMTADD;
FORMAT ZDO I = 1 XTO &N;
&NAMEAI &FMTSI
ZEND;
RUN;
%XEND GETFMTS;
/* Figure 2 */

%MACRO MERGDAT (LIB, MEM1, MEM2, OUT, BYVAR ) ;

PROC CONTENTS DATA = &LIB..\&MEM1
OUT = CONTI NOPRINT ;
RUN ;
PROC CONTENTS DATA = &LIB..\&MEM2
OUT = CONT2 NOPRINT ;
RUN ;

PROC CONTENTS DATA = &LIB..\&MEM1
OUT = CONTI NOPRINT ;
RUN ;
PROC CONTENTS DATA = &LIB..\&MEM2
OUT = CONT2 NOPRINT ;
RUN ;

/*----------------------GET VARIABLE NAMES--------------------------*/

DATA _NULL_;
RETAIN CNT1 CNT2 0
IF EOF THEN DO ;
CALL SYMPUT('CNT1', LEFT(PUT(CNT1, 4.»)
CALL SYMPUT('CNT2', LEFT(PUT(CNT2, 4.»)
END ;
SET CONTI ( IN = - MEM1 )
CONT2 ( IN = - MEM2 ) END - EOFC ;

IF MEM1 = 1 THEN DO ;
IF NAME = "&BYVAR" THEN DELETE
CNT1 + 1 ;
CALL SYMPUT('D1_.' II LEFT(PUT(CNT1, 4.»), TRIM(NAME)) ;
END ;
ELSE IF MEM2 = 1 THEN DO ;
IF NAME = "&BYVAR" THEN DELETE
CNT2 + 1 ;
CALL SYMPUT('D2_.' II LEFT(PUT(CNT2, 4.»), TRIM(NAME)) ;
END ;

RUN ;

/*----------------------GENERATE SAS CODE---------------------------*/

DATA &LIB..OUT ;
MERGE &LIB..\&MEM1
(RENAME = (
ZDO I = 1 XTO &CNT1 ;
&D1_&I = &D1_&I.._1
ZEND ;
)
&LIB..\&MEM2
(RENAME = (
ZDO I = 1 XTO &CNT2 ;
&D2_&I = &D2_&I.._2
ZEND ;
)
);
BY &BYVAR ;
RUN ;
ZMEND MERGDAT ;
/* Figure 3 */

%MACRO RANKCORR (DATA = , OUT = , WITH = YES);
%----------------------------------------------------------
%* AUTHOR: H. IAN WHITLOCK DATE: FEB. 27, 1985
%* REVISED FOR VER 5.08 AUG. 2, 1985
%* RANKCORR - PRODUCES REPORT SIMILAR TO PROC CORR
%* USING THE RANK OPTIONS BUT NEGATIVE CORRELATIONS ARE LOW
%* PARAMETER
%* DATA - OUTPUT OF PROC CORR USING OUTP OPTION
%* OUT - OUTPUT OF THIS MACRO
%* DATASETS GENERATED
%* WORK.DRCTRY
%* VARIABLE NAMES TO AVOID
%* NAME1 NAME2 COEFF (VARS OUTPUT BY RANKCORR)
%* __ NAMES __ COEFFS __ NMI, __ NM2, ...
%* (VARS USED WITHIN RANKCORR)
%* END
%----------------------------------------------------------

LOCAL RSWITCH;
IF &WITH = YES THEN LET RSWITCH = 1;
ELSE LET RSWITCH = RETURN;
/* 1) GET CORRELATION VARIABLE NAMES */
PROC CONTENTS DATA = DATA OUT = DRCTRY NOPRINT;
RUN;
PROC SORT DATA = DRCTRY BY NPOS;
/* SORT TO POSITION ORDER */
BY NPOS;
/* FOR '---' NOTATION */
RUN;
/* 2) GENERATE MACRO VARIABLES */
DATA _NULL_; RETAIN N 0;
IF ENDD THEN CALL SYMPUT ( 'NUMCOR' , LEFT(PUT(N,8.)) ) ;
SET DRCTRY END = ENDD;
/* DELETE NON CORRELATION VARIABLES */
IF NAME = "_TYPE_" OR NAME = "_NAME_" THEN DELETE;
N + 1;
CALL SYMPUT ( 'SN' 'I' LEFT(PUT(N,8.)) , NAME ) ;
RUN;
/* 3) READ CORRELATION DATA */
DATA &OUT ( KEEP =
   NAME1 /* 'VAR' VARIABLES FROM PROC CORR */
   NAME2 /* 'WITH' VARIABLES */
   COEFF /* CORRELATION COEFFICIENTS */
   LABEL = 'OUTPUT OF RANKCORR' ) ;
LENGTH NAME1 NAME2 $ 8 COEFF 8;
/* RETAIN THE VARIABLE NAMES IN AN ARRAY */
ARRAY __ NAMES $ 8 __ NMI - __ NM&NUMCOR;
/* RETAIN __ COEFFS __ NAMES __ NMI, __ NM2, ...
(VARS USED WITHIN RANKCORR)
DO J = 1 %TO &NUMCOR
   NAME1 - __ NAMES
   COEFF - __ COEFFS
   IF NAME1 - NAME2 THEN OUTPUT ELSE &RSWITCH
   END;
/* RETURN FOR NO WITH STMT */
RUN;
PROC SORT DATA = &OUT
   BY NAME1 DESCENDING COEFF;
RUN;
%MEND RANKCORR;