Paper 130-30

SAS® Macro Variables and Simple Macro Programs

Steven First, Katie Ronk, Systems Seminar Consultants, Madison, WI

ABSTRACT

SAS macros can be a wonderful extension of the SAS language. This hands-on-workshop will introduce SAS users to SAS macro variables and simple macro programs. Please note that due to time constraints it will be only an introduction.

INTRODUCTION

The SAS® programming language has a rich toolbox of features that can offer a lot of power to the user. Using macro variables and macro code is difficult to learn without some training and examples that are easy to understand. This workshop will introduce the user to a few of the very powerful techniques that are possible using macro variables and macro programming.

Some of the topics to be covered include:

• What is a macro?
• What is a macro variable and what does it do?
• Examples of some of the standard SAS macro variables
• How the SAS macro processor handles macro variables
• Passing information to other steps in a program using macro variables
• Creating and invoking macro code
• How to conditionally run SAS steps based on the logic in your program

OBJECTIVES:

After completing this class students will:

- understand how the macro system fits with the rest of SAS software
- use system (automatic) macro variables
- create and use user defined macro variables
- define simple macros
- pass data from the data step to the macro system.

SAS MACRO OVERVIEW

SAS macros construct input for the SAS compiler. Some functions of the SAS macro processor are to pass symbolic values between SAS statements and steps, to establish default symbolic values, to conditionally execute SAS steps, and to invoke very long, complex code in a quick, short way. It should be noted that the macro processor is the SAS system module that processes macros and the SAS macro languages is how you communicate with the processor.

Without macros it is not easy to substitute variable text in statements such as TITLEs, to communicate across SAS steps,
to establish default values, and to conditionally execute SAS step. Macros can do this and also hide complex code that can be invoked easily.

Without macros, SAS programs are DATA and PROC steps that are scanned one statement at a time looking for the
beginning of step (step boundary). When the beginning of step is found, all statements in the step are compiled and this continues until when the end of step is found (the next step boundary), the previous step executes.

SAS step boundaries are the SAS keywords:

```
DATA            ENDSAS
PROC            LINES
CARDS           LINES4
CARDS4          PARMCARDS
DATALINES       QUIT
DATALINES4      RUN
```

The RUN statement, while not an explicit step boundary, acts as an explicit step boundary in most PROCs to start execution immediately. The use of RUN after each step is highly recommended as shown by the example below.

```
data saleexps;                <--Step, start compile
 infile rawin;
 input name $1-10 division $12
       years 15-16 sales 19-25
       expense 27-34;
 run;                         <--Step end, exec previous

proc print data=saleexps;     <--Step start, start compile
 run;                         <--Step end, exec previous
proc means data=saleexps;
 var sales expense;
 run;                         <--Step end, exec previous
```

THE SAS MACRO LANGUAGE

The SAS macro language is a second SAS programming language imbedded in SAS code that manipulates strings. Characteristics of this language are:

- strings are sequences of characters
- all input to the macro language is a string
- usually strings are SAS code, but don't need to be
- the macro processor manipulates strings and may send them back for scanning.

MACRO LANGUAGE COMPONENTS

The macro language has several kinds of components.

1. **Macro variables:**
   - are used to store and manipulate character strings
   - follow SAS naming rules
   - are NOT the same as DATA step variables
   - are stored in memory in a macro symbol table.

2. **Macro statements:**
   - begin with a % and a macro keyword and end with semicolon (;)
   - assign values, substitute values, and change macro variables
   - can branch or generate SAS statements conditionally.

MACRO PROCESSOR FLOW

Macro statements are given to the macro processor BEFORE the compiler.
Two special triggers direct action by the macro processor. The percent sign (%.) may signify the start of a macro statement and macro variables are referred to with &. These characters might, however, be used for other purposes in text and may not be intended to do anything with the macro system

A MACRO PROBLEM

Suppose you would like to have the day of week and current date appear in a title, but SAS titles are text, not variables. A possible solution might be to use some automatic system macro variables.

```
PROC PRINT DATA=DEPTSALE;
 TITLE  "Department Sales as of &SYSDAY  &SYSDATE";
 TITLE2 "Deliver to Michael O'Malley";
RUN;
```

Automatic Macro Variables

Below is a partial list of automatic macro variables and their usage:

```
SYSBUFFR     text entered in response to %INPUT
SYSCMD       last non-SAS command entered
SYSDATE      current date in DATE6. or DATE7. format
SYSDAY       current day of the week
SYSDEVIC     current graphics device
SYSDSN       last SAS dataset built (i.e.,WORK SOFTSALE)
SYSENV       SAS environment (FORE or BACK)
SYSERR       return code set by SAS procedures
SYSFILRC     whether last FILENAME executed correctly
SYSINDEX     number of macros started in job
SYSINFO      system information given by some PROCS
SYSJOBID     name of executing job or user
SYSLAST      last SAS dataset built (i.e., WORK.SOFTSALE)
SYSLIBRC     return code from last LIBNAME statement
SYSLCKRC     whether most recent lock was successful
SYSMENV      macro execution environment
SYSMSG       message displayed with %DISPLAY
SYSPARM      value passed from SYSPARM in JCL
SYSPROD      indicates whether a SAS product is licensed
SYSPBUFF     all macro parameters passed
SYSRC        return code from macro processor
SYSSCP       operating system where SAS is running
SYSTIME      starting time of job
SYSVER       SAS version
```

Any of these variables can be used anywhere appropriate in SAS. Again, if quoted use the double quote character if resolution is desired.

For example:

```
FOOTNOTE "THIS REPORT WAS RUN ON &SYSDAY, &SYSDATE";
```

Resolves to:

```
FOOTNOTE "THIS REPORT WAS RUN ON FRIDAY, 25MAR05";
```

Displaying Macro Variables

%PUT displays macro variables to the log at compile time. There is no column control as in the DATA Step PUT statement, and quotes are not required. %PUT can display text, macro variables, or a few special variables such as _all_ which displays all macro variables. %PUT is often the easiest way to debug macros.

Example:

```
DATA NEWPAY;
 INFILE DD1;
 INPUT EMP$ RATE;
RUN;
%PUT ***** &SYSDATE *****;
```

Partial SAS Log:

```
***** 25MAR05 *****
```

Displaying All Macro Variables

%PUT can display all current macro variables by specifying _ALL_.

Example:

```
%PUT _ALL_;
```

Partial SAS Log:

```
GLOBAL MBILLYR 99
GLOBAL SSCDEV C
AUTOMATIC AFDSID 0
AUTOMATIC AFDSNAME
AUTOMATIC AFLIB
AUTOMATIC AFSTR1
AUTOMATIC AFSTR2
AUTOMATIC FSPBDV
AUTOMATIC SYSBUFFR
AUTOMATIC SYSCMD
AUTOMATIC SYSDATE 25MAR05
AUTOMATIC SYSDAY Friday
AUTOMATIC SYSDSN         _NULL_
AUTOMATIC SYSENV FORE
AUTOMATIC SYSERR 0
AUTOMATIC SYSFILRC 0
AUTOMATIC SYSINDEX 1
AUTOMATIC SYSINFO 0
AUTOMATIC SYSJOBID 0000016959
AUTOMATIC SYSLAST _NULL_
AUTOMATIC SYSMSG
AUTOMATIC SYSPARM
AUTOMATIC SYSRC 0
AUTOMATIC SYSSCP WIN
AUTOMATIC SYSSCPL WIN_32S
AUTOMATIC SYSSITE 0011485002
AUTOMATIC SYSTIME 10:35
AUTOMATIC SYSVER 6.11
AUTOMATIC SYSVLONG 6.11.0040P030596
```

User Defined Macro Variables

Problem: You reference a SAS datasetname several times in a SAS job.

```
DATA PAYROLL;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
PROC PRINT DATA=PAYROLL;
 TITLE "PRINT OF DATASET PAYROLL";
RUN;
```

Question: How can you change the name quickly in one place only AND have the datasetname appear in a title?

Solution:

- Use a macro variable.
- You can define macro variables with %LET.
- You refer to the variables later with &variable.
- Macro will substitute value for all occurrences of &variable.

Syntax:

```
%LET variable=value;
```

```
%LET NAME=PAYROLL;
DATA &NAME;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
PROC PRINT DATA=&NAME;
 TITLE "PRINT OF DATASET &NAME";
RUN;
```

The Generated SAS Code

```
DATA PAYROLL;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
PROC PRINT DATA=PAYROLL;
 TITLE "PRINT OF DATASET PAYROLL";
RUN;
```

To later assign a new value, use another %LET to assign a different value.

```
%LET NAME=NEWPAY;
DATA &NAME;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
PROC PRINT DATA=&NAME;
 TITLE "PRINT OF DATASET &NAME";
RUN;
```

The Generated SAS Code

```
DATA NEWPAY;
 INPUT EMP$ RATE;
DATALINES;
 TOM 10
 JIM 10
  ;
PROC PRINT DATA=NEWPAY;
TITLE "PRINT OF DATASET NEWPAY";
RUN;
```

Assigning SAS Statements

To include special characters including semicolon, SAS statements etc, %STR allows you to enclose special characters within parentheses..

```
%LET NAME=NEWPAY;
%LET CHART=%STR(PROC CHART;VBAR EMP;RUN;);
DATA &NAME;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
&CHART
PROC PRINT DATA=&NAME;
 TITLE "PRINT OF DATASET &NAME";
RUN;
```

The Generated SAS Code

```
DATA NEWPAY;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
PROC CHART;VBAR EMP;RUN;
PROC PRINT DATA=NEWPAY;
 TITLE "PRINT OF DATASET NEWPAY";
RUN;
```

Nesting of Macro Variables

Macro variables can contain other macro variables.

```
%LET NAME=NEWPAY;
%LET CHART=%STR(PROC CHART DATA=&NAME;VBAR EMP;RUN;);
DATA &NAME;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
&CHART
PROC PRINT DATA=&NAME;
 TITLE "PRINT OF DATASET &NAME";
RUN;
```

The Generated SAS Code

```
DATA NEWPAY;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
PROC CHART DATA=NEWPAY;VBAR EMP;RUN;
PROC PRINT DATA=NEWPAY;
 TITLE "PRINT OF DATASET NEWPAY";
RUN;
```

Other Macro Debugging Options

SAS gives several options that control log output at various points in the compilation and execution of your SAS program.

- SYMBOLGEN/NOSYMBOLGEN controls display of variable resolution
- MPRINT/NOMPRINT displays generated statements given to the compiler
- MLOGIC/NOMLOGIC displays tracing information during macro execution.

SAS Macros

Stored text including logic that can be inserted anywhere in a SAS program and expanded.

Macros can include:
- constants such as literals, variables, names, and statements
- assignments to macro variables
- macro programming statements
- macro language functions
- invocations of other functions
- nested macro definitions
- LOGIC to conditionally generate SAS code.

Defining and Using Macros (Macro Programs)

%MACRO and %MEND define macros, %macroname will invoke it later.

Example: Define a macro to run PROC CHART and later invoke it.

```
%MACRO CHART;
  PROC CHART DATA=&NAME;
    VBAR EMP;
RUN;
%MEND;


%LET NAME=NEWPAY;
DATA &NAME;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
RUN;
%CHART
PROC PRINT DATA=&NAME;
 TITLE "PRINT OF DATASET &NAME";
RUN;
```

The Generated Code

```
DATA NEWPAY;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
RUN;
PROC CHART DATA=NEWPAY;
 VBAR EMP;
RUN;
PROC PRINT DATA=NEWPAY;
 TITLE "PRINT OF DATASET NEWPAY";
RUN;
```

Positional Macro Parameters

Positional macro parameters are defined in order after the macro name. Values for those parameters can be set when the macro is invoked. Keyword parameters are also allowed, and can set default values.

```
%MACRO CHART(NAME,BARVAR);
  PROC CHART DATA=&NAME;
    VBAR &BARVAR;
  RUN;
%MEND;

%CHART(PAYROLL,EMP)
```

Resolves to:

```
PROC CHART DATA=PAYROLL;
  VBAR EMP;
RUN;
```

Nested Macros

Macros can call other macros.

```
%MACRO CHART(NAME,BARVAR);
  PROC CHART DATA=&NAME;
    VBAR &BARVAR;
     RUN;
%MEND;

%MACRO PTCHART(NAME,BARVAR);
%CHART(PAYROLL,EMP)
  PROC PRINT DATA=&NAME;
    TITLE "PRINT OF DATASET &NAME";
  RUN;
%MEND;

%PTCHART(PAYROLL,EMP)
```

The Generated SAS Code

```
PROC CHART DATA=PAYROLL;
 VBAR EMP;
RUN;
PROC PRINT DATA=PAYROLL;
  TITLE  "PRINT OF DATASET PAYROLL";
RUN;
```

Conditional Macro Compilation

%IF can conditionally pass code to the compiler.

Example: Run PROC PRINT only if PRTCH=YES.

```
%MACRO PTCHT(PRTCH,NAME,BARVAR);
  %IF &PRTCH=YES %THEN PROC PRINT DATA=&NAME;
       ;
  PROC CHART DATA=&NAME;
   VBAR &BARVAR;
  RUN;
%MEND;

%PTCHT(YES,PAYROLL,EMP)
```

The Generated SAS Code

```
PROC PRINT DATA=PAYROLL ;
PROC CHART DATA=PAYROLL;
  VBAR EMP;
RUN;
```

The %DO Statement

%DO allows many statements to be conditionally compiled.

Example: Submit as before, but include titles.

```
%MACRO PTCHT(PRTCH,NAME,BARVAR);
  %IF &PRTCH=YES %THEN
      %DO;
          PROC PRINT DATA=&NAME;
          TITLE "PRINT OF DATASET &NAME";
          RUN;
      END;
  PROC CHART DATA=&NAME;
   VBAR &BARVAR;
  RUN;
%MEND;

%PTCHT(YES,PAYROLL,EMP)
```

The Generated SAS Code

```
PROC PRINT DATA=PAYROLL;
 TITLE  "PRINT OF DATASET PAYROLL";
RUN;
PROC CHART DATA=PAYROLL;
 VBAR EMP;
RUN;
```

Iterative Macro Invocation

%DO can also vary a value.

Example: Run PROC PRINT &PRTNUM times.

```
%MACRO PRTMAC(PRTNUM,NAME);
 %DO I= 1 %TO &PRTNUM;
     PROC PRINT DATA=&NAME&I;
      TITLE "PRINT OF DATASET &NAME&I";
RUN;
 %END;
%MEND;

%PRTMAC(4,PAYROLL)
```

The Generated SAS Code

```
PROC PRINT DATA=PAYROLL1;
  TITLE "PRINT OF DATASET PAYROLL1";
RUN;
PROC PRINT DATA=PAYROLL2;
  TITLE "PRINT OF DATASET PAYROLL2";
RUN;
PROC PRINT DATA=PAYROLL3;
  TITLE "PRINT OF DATASET PAYROLL3";
RUN;
PROC PRINT DATA=PAYROLL4;
  TITLE "PRINT OF DATASET PAYROLL4";
RUN;
```

SAS DATA Step Interfaces

SYMGET, SYMPUT, and macro variables can transfer values between SAS steps.

- SYMGET returns macro variable values to the DATA step
- Macro variables created with SYMPUT, can be referenced via & in the NEXT step.

Example: Display the number of observations in a dataset in a title.

```
%MACRO OBSCOUNT(NAME);
  DATA _NULL_;
   SET &NAME NOBS=OBSOUT;
   CALL SYMPUT('MOBSOUT',OBSOUT);
   STOP;
  RUN;
  PROC PRINT DATA=&NAME;
   TITLE "DATASET &NAME CONTAINS &MOBSOUT OBSERVATIONS";
  RUN;
%MEND;

%OBSCOUNT(PAYROLL)
```

The Generated SAS Code

```
DATA _NULL_;
 SET PAYROLL NOBS=OBSOUT;
 CALL SYMPUT('MOBSOUT',OBSOUT);
 STOP;
RUN;
PROC PRINT DATA=PAYROLL;
 TITLE  "DATASET PAYROLL CONTAINS 50 OBSERVATIONS";
RUN;
```

A SAS Macro Application

Data Set COUNTYDT

Obs	COUNTYNM	READING
1	ASHLAND	125
2	ASHLAND	611
3	BAYFIELD	101
4	BAYFIELD	101
5	BAYFIELD	222
6	WASHINGTON	143

Problem: Each day you read a SAS dataset containing data from counties in Wisconsin. Anywhere between 1 and 72 counties might report that day. Do the following:

1. Create a separate dataset for each reporting county.
2. Produce a separate PROC PRINT for each reporting county.
3. In the TITLE print the county name.
4. Reset the page number to 1 at the beginning of each report.
5. In a footnote print the number of observations processed for each county.

Question: How do you do it?
Solution: A Data Step and a SAS Macro.

A data step and a macro to generate the PROC PRINTs.

The data step goes through the data and:
- counts counties
- counts observations per county, puts in macro variables
- puts countynms into macro variables
- puts total counties reporting into a macro variable.

The Data Step Code

```
DATA _NULL_;
SET COUNTYDT END=EOF;           /* READ SAS DATASET      */
 BY COUNTYNM;                   /* SORT SEQ              */
 IF FIRST.COUNTYNM THEN DO;     /* NEW COUNTY ?          */
    NUMCTY+1;                   /* ADD 1 TO NUMCTY       */
    CTYOBS=0;                   /* OBS PER COUNTY TO 0   */
    END;
 CTYOBS+1;                      /* ADD ONE OBSER FOR CTY */
IF LAST.COUNTYNM THEN DO;       /* EOF CTY, MAKE MAC VARS*/
    CALL SYMPUT('MCTY'||LEFT(PUT(NUMCTY,3.)),COUNTYNM);
    CALL SYMPUT('MOBS'||LEFT(PUT(NUMCTY,3.)),LEFT(CTYOBS));
    END;
IF EOF THEN
    CALL SYMPUT('MTOTCT',NUMCTY);/* MAC VAR NO DIF CTYS  */
RUN;
%PUT *** MTOTCT=&MTOTCT;         /* DISPLAY NO OF CTYS   */
```

The Generated Macro Variables
One for each countynm, obs/county, and total num of counties.

```
SYMBOL TABLE

NAME          VALUE

MCTY1         ASHLAND
MOBS1         2
MCTY2         BAYFIELD
MOBS2         3
MCTY3         WASHINGTON
MBOS3         1
MTOTCT        3
```

The Macro to Loop Around PROC PRINT

```
%MACRO COUNTYMC;                 /* MACRO START           */
%DO I=1 %TO &MTOTCT;             /* LOOP THRU ALL CTYS    */
 %PUT *** LOOP &I OF &MTOTCT;    /* DISPLAY PROGRESS      */
 PROC PRINT DATA=COUNTYDT;       /* PROC PRINT            */
  WHERE COUNTYNM="&&MCTY&I";     /* GENERATED WHERE       */
  OPTIONS PAGENO=1;              /* RESET PAGENO          */
  TITLE "REPORT FOR COUNTY &&MCTY&I";
                                 /* TITLES AND FOOTNOTES  */
  FOOTNOTE "TOTAL OBSERVATION COUNT WAS &&MOBS&I";
 RUN;
%END;                            /* END OF %DO            */
%MEND COUNTYMC;                  /* END OF MACRO          */
%COUNTYMC                        /* INVOKE MACRO          */
```

The Generated Code

```
*** MTOTCT=3
*** LOOP 1 OF 3
   PROC PRINT DATA=COUNTYDT;
   WHERE COUNTYNM="ASHLAND";   OPTIONS PAGENO=1;
   TITLE "REPORT FOR COUNTY ASHLAND";
   FOOTNOTE "TOTAL OBSERVATION COUNT WAS 2";   RUN;
*** LOOP 2 OF 3
   PROC PRINT DATA=COUNTYDT;
   WHERE COUNTYNM="BAYFIELD";   OPTIONS PAGENO=1;
   TITLE "REPORT FOR COUNTY BAYFIELD";
   FOOTNOTE "TOTAL OBSERVATION COUNT WAS 3";   RUN;
*** LOOP 3 OF 3
   PROC PRINT DATA=COUNTYDT;
   WHERE COUNTYNM="WASHINGTON";   OPTIONS PAGENO=1;
   TITLE "REPORT FOR COUNTY WASHINGTON";
   FOOTNOTE "TOTAL OBSERVATION COUNT WAS 1";  RUN;
```

The Generated Output

```
REPORT FOR COUNTY ASHLAND       1

OBS    COUNTYNM    READING

 1     ASHLAND       125
 2     ASHLAND       611


TOTAL OBSERVATION COUNT WAS 2
```

```
REPORT FOR COUNTY BAYFIELD      1


OBS    COUNTYNM    READING

 3     BAYFIELD      101
 4     BAYFIELD      101
 5     BAYFIELD      222

TOTAL OBSERVATION COUNT WAS 3


REPORT FOR COUNTY WASHINGTON    1

OBS     COUNTYNM     READING

 6     WASHINGTON      143


TOTAL OBSERVATION COUNT WAS 1
```

A Solution Via PROC SQL

PROC SQL can create macro variables.
Using the INTO clause, PROC SQL can easily insert a number of values into corresponding SAS macro variables.

```
PROC SQL;
  SELECT  LEFT(PUT(COUNT(DISTINCT COUNTYNM),3.))
           INTO:MTOTCT FROM COUNTYDT;
                                  /* NO OF UNIQUE CTYS     */
  SELECT  DISTINCT COUNTYNM     /* EACH CTY NAME         */
          INTO:MCTY1-:MCTY&MTOTCT FROM COUNTYDT;
  SELECT  COUNT(*)              /* OBS PER               */
          INTO:MOBS1-:MOBS&MTOTCT FROM COUNTYDT
  GROUP BY COUNTYNM;

%PUT *** MTOTCT=&MTOTCT;        /* DISPLAY NO OF CTYS    */

%MACRO COUNTYMC;                /* MACRO START           */
%DO I=1 %TO &MTOTCT;            /* LOOP THRU ALL CTYS    */
 %PUT *** LOOP &I OF &MTOTCT;   /* DISPLAY PROGRESS      */
 PROC PRINT DATA=COUNTYDT;      /* PROC PRINT            */
  WHERE COUNTYNM="&&MCTY&I";    /* GENERATED WHERE       */
  OPTIONS PAGENO=1;             /* RESET PAGENO          */
  TITLE "REPORT FOR COUNTY &&MCTY&I";
                                /*TITLES AND FOOTNOTES  */
 FOOTNOTE "TOTAL OBSERVATION COUNT WAS &&MOBS&I";
 RUN;
%END;                           /* END OF %DO            */
%MEND COUNTYMC;                 /* END OF MACRO          */
%COUNTYMC                       /* INVOKE MACRO          */
```

The SSCFLAT Macro

A general purpose macro:
- converts any SAS ds to a comma delimited file for input to spreadsheets etc.
- will run on all platforms
- automatically reads dictionary tables to get ds definition
- honors SAS formatting
- can create a header line
- dropping, reordering variables etc. can be done in an earlier SAS step.

The SSCFLAT Macro Partial Source

```
/*********************************************************/
/* MACRO SSCFLAT VERSION 1.4                             */
/* CREATES A COMMA DELIMITED FILE FROM ANY SAS DATA SET   */
/* IT CAN BE THEN DOWNLOADED & IMPORTED INTO A SPREADSHEET*/
/*                                                       */
/* SAMPLE WINDOWS CALL:                                  */
/*  %SSCFLAT(MSASDS=MAIL.TEMPMAIL.MPREFIX=C:\TEMP\)      */
/*                                                       */
/* SAMPLE MVS CALL:                                      */
/*  %SSCFLAT(MSASDS=WORK.XYZ.MPREFIX=MYUSER)             */
/*                                                       */
/* STEVEN FIRST                                          */
/* (C) SYSTEMS SEMINAR CONSULTANTS 1999   608 278-9964    */
/*                                                       */
/* PERMISSION GIVEN TO FORMER SSC SAS STUDENTS TO USE     */
/*     IN PERSONAL SAS JOBS.                             */
/* FOR PERMISSION TO DISTRIBUTE TO OTHERS, OR FOR         */
/*     COMPANY WIDE USE, CONTACT SSC AT 608 278-9964      */
/*                                                       */
/*********************************************************/
%MACRO SSCFLAT(MSASDS=,            /*INPUT SAS DS (REQUIRD*/
   MPREFIX=&SYSPREF..,             /*OUT PREF, OR DIR OUT */
   MFLATOUT=&MPREFIX&MMEMNAME..DAT,/*FLATFILE OUT         */
   MHEADER=YES,                    /*FLD NAMES IN 1ST REC */
   MLIST=YES,                      /*PRINT FLT FILE IN LOG*/
   MTRIMNUM=YES,                   /*TRIM NUM TRAIL BLANKS*/
   MTRACE=NO,                      /*DEBUGGING OPTION     */
   MMISSING=“.”,                   /*MISSING VALUE CHAR   */
   MLRECL=6160,                    /*LARGEST RECORD LEN   */
    MVSOPT=UNIT=3390,             /*MVS UNIT OPTIONS     */
   MSPACE=1,                       /*MVS SPACE IN CYLS    */
   );

%PUT ***** SSCMAC COPYRIGHT (C) 1999 SYSTEMS SEMINAR;
%PUT ***** CONSULTANTS 608 278-9964;
```

A SSCFLAT Macro Example
In windows, convert WORK.ADDRDATA to a FLAT file.

```
            ADDRDATA

OBS    NAMES  DEPT   AGE    RATE

1      STEVE  ACCT   43     12.22
2      DAVID  PAYR          11.21
```

```
%INC ‘insert file ref\sscflat.sas.’;
%SSCFLAT(MSASDS=WORK.ADDRDATA,
mprefix=c:\temp\)   *invoke macro;
```

The c:\temp\addrdata.dat flat file created:

```
“NAME”,“DEPT”,“AGE”,“RATE”
“STEVE”,”ACCT ”,43,12.22
“DAVID”,”PAYR “,,11.21
```

The flat file is now available for download and/or import to spreadsheets, etc.

CONCLUSION

The SAS macro language is an integral part of the SAS system and while not always easy to understand and use, it can be a wonderful tool for the SAS programmer.

CONTACT INFORMATION

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

Steven First
Systems Seminar Consultants
2997 Yarmouth Greenway Drive
Madison, WI 53716
608 278-9964 (Work)
sfirst@sys-seminar.com
www.sys-seminar.com