Paper 107-31

Intermediate and Advanced SAS® Macros

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

Abstract

This hands-on workshop presents some intermediate-to-advanced macro topics such as macro referencing environments, macro interfaces (SYMGET, SYMPUT, EXECUTE, RESOLVE, PROC SQL), macro quoting, and macro functions. Good practices and alternatives to macros are also discussed.

After a short lecture, attendees will perform hands-on exercises until the end of the session.

Introduction

The SAS® programming language has a rich toolbox of features that can offer a lot of power to the user. The SAS macro language can be used to generate and alter SAS code. By combining these two languages the user can create some very dynamic systems. Combining these two languages can be daunting however. This workshop will concentrate on the interfaces available between the SAS macro language and other system components.

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.

User Defined Macro Variables

%LET can define a macro variable that can be used later in the program.

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

RUN;
```

A More Involved Macro Application

As logic is required, a SAS macro can be used to generate SAS code conditionally and then invoke the macro later.

Example: Generate several SAS dataset names.

```
%MACRO DSNAMES(PREFIX,FIRST,LAST);
   %LOCAL N;
    %DO N=&FIRST %TO &LAST;
      &PREFIX&N
    %END;
   %MEND DSNAMES;

data
%dsnames(day,1,4)
;
...
```

Generates:

```
data day1 day2 day3 day4;
```

SAS DATA Step Interfaces

SYMGET, SYMPUT, and other interfaces using 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.

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
```

Other Interfaces With The Macro Facility

Interfaces:

- are not part of the macro facility,
- are rather a SAS software feature
- enable another portion of SAS to interact with macro at execution.

Interfaces exist between macro and:

- the DATA step
- SCL (SAS Component Language)
- PROC SQL
- SAS/CONNECT
- the host operating system

DATA Step Interfaces

There are eight interfaces that interact with macros when DATA steps execute.
You can use DATA step interfaces to:

- pass information to later steps via macro variables
- generate and submit SAS statements
- invoke a macro when DATA step executes
- resolve macro elements when DATA step executes
- delete a macro variable
- pass information about macro variables to DATA step.

Interface	Description
CALL EXECUTE routine	resolves argument and executes macros immediately and generated code at next step boundary
RESOLVE function	resolves text expression at execution
CALL SYMDEL routine	deletes a macro variable
SYMEXIST function	test for macro variable existence

SYMGET function	returns macro variable at execution
SYMGLOBL function	tests for global scope
SYMLOCAL function	tests for local scope
CALL SYMPUT routines	assigns DATA step values to macro variables

Retrieving Macro Variable Values

SYMGET places values of macro variables into the PDV at execution time.

Syntax:

DATAstepvariable=SYMGET(argument);

argument can be:
a character literal (Ex. 'MDEPT')
a DATA step character variable (Ex. DEPT)
a character expression (Ex. 'MDEPT' !! '01')

```
%LET LANGUAGE=SAS;
%LET COURSE=MACRO;
%LET ENV=INTERACTIVE;

DATA COURSE;
       INPUT WORDS $;
       NAME = SYMGET(WORDS);
DATALINES;
ENV
LANGUAGE
COURSE;
RUN;
PROC PRINT DATA=COURSE;
   TITLE1 'COURSE DATASET';
RUN;
```

Storing Macro Variable Values

SYMPUT copies values of DATA step variables into macro variables.

Syntax:

CALL SYMPUT(argument1,argument2);

Where:
Argument1 is the macro variable where the value is placed.
Argument2 is the DATA step value that will be assigned.
Arguments can be character literals (Ex. 'MDEPT'), DATA step character variables (Ex. DEPT), or character expressions (Ex. 'MDEPT' !! '01').
Argument1 must result in a character string that is legal for macro variable names.

```
DATA SWITCH;
   INPUT VAR1 $ VAR2 $;
   CALL SYMPUT(VAR1,VAR2);
   DATALINES;
FNAME RICHARD
MNAME MILHOUSE
LNAME NIXON
;
```

```
RUN;
%PUT FNAME=&FNAME;
%PUT MNAME=&MNAME;
%PUT LNAME=&LNAME;
```

Partial log:

```
FNAME=RICHARD
MNAME=MILHOUSE
LNAME=NIXON
```

SYMPUT With a Numeric Value

Numeric values must be converted to character and leading and trailing blanks trimmed, or a conversion message will display.

```
%LET SAMPSIZE=5;
DATA SAMPLE;
   DO N=1 TO &SAMPSIZE;
       OBSNO=CEIL(UNIFORM(0)*TOTOBS);
       SET POP POINT=OBSNO NOBS=TOTOBS;
       OUTPUT;
   END;
CALL SYMPUT('MTOTOBS',LEFT(PUT(TOTOBS,8.)));
STOP;
RUN;
```

CALL SYMPUTX

SYMPUTX will automatically convert and trim numbers.

Syntax:

CALL SYMPUTX(argument1,argument2, <,symbol-table>);

SYMPUTX is:

- similar to SYMPUT but will automatically convert and trim
- works for both character and numeric values
- uses up to 32 characters with best format
- allows you to specify a specific symbol table
- was added in Version 9.

Example:

```
%LET SAMPSIZE=5;
DATA SAMPLE;
   DO N=1 TO &SAMPSIZE;
       OBSNO=CEIL(UNIFORM(0)*TOTOBS);
       SET POP POINT=OBSNO NOBS=TOTOBS;
       OUTPUT;
   END;
CALL SYMPUTX('MTOTOBS',TOTOBS);
STOP;
RUN;
```

CALL EXECUTE

Resolves an argument and executes the resolved value at step boundary. This is a very powerful tool can can use the power of the data step to generate code.

Syntax:

CALL EXECUTE(argument);

EXECUTE::

if argument is single quoted string, resolves during data step execution.
If double quoted string and a macro element, resolves during data step compile.
If a DATA step variable, must contain a text expression or SAS statement to generate
can also be a expression that is resolved to a macro text expression or SAS statement.
CALL EXECUTE can be a good way to avoid complicated macros.

A Call Execute Example
Generate a PROC PRINT for each county in our file.

```
data pass2;
 set perm.countydt;
  by countynm;
  if first.countynm then ctyobs=0;
  ctyobs+1;
  if last.countynm then
     call execute('PROC PRINT DATA=PERM.COUNTYDT; '      !!
                  'WHERE COUNTYNM="'                     !!
                   countynm                              !!
                  '";'                                   !!
                  'OPTIONS PAGENO=1;'                    !!
                  'TITLE "REPORT FOR COUNTY '            !!
                  countynm                               !!
                  '";'                                   !!
                  'FOOTNOTE "TOTAL OBSERVATION COUNT WAS ' !!
                  put(ctyobs,2.)                         !!
                  '";'                                   !!
                  'RUN;'
                  );
run;
```

The resulting SAS Log:

```
NOTE: CALL EXECUTE generated line.
1   + PROC PRINT DATA=PERM.COUNTYDT;
      WHERE COUNTYNM="ASHLAND";
      OPTIONS PAGENO=1;
      TITLE "REPORT FOR COUNTY ASHLAND";
      FOOTNOTE "TOTAL OBSERVATION COUNT WAS  2";RUN;

NOTE: CALL EXECUTE generated line.
2   + PROC PRINT DATA=PERM.COUNTYDT;
      WHERE COUNTYNM="BAYFIELD";
      OPTIONS PAGENO=1;
      TITLE "REPORT FOR COUNTY BAYFIELD";
      FOOTNOTE "TOTAL OBSERVATION COUNT WAS  3";RUN;
       . . .
```

Call Execute Using a Macro
Define a macro to do most of the work, then call it.

```
%macro printcty(mcountynm,mctyobs);
 PROC PRINT DATA=PERM.COUNTYDT;
```

```
   WHERE COUNTYNM="&mcountynm";
   OPTIONS PAGENO=1;
   TITLE "REPORT FOR COUNTY &mcountynm";
   FOOTNOTE "TOTAL OBSERVATION COUNT WAS &MCTYOBS";
   RUN;
%mend printcty;

data pass2;
 set perm.countydt;
  by countynm;
  if first.countynm then ctyobs=0;
  ctyobs+1;
  if last.countynm then
     call execute('%printcty('    !!
                   countynm       !!
                   ','            !!
                   put(ctyobs,2.) !!
                   ')'
                 );
run;
```

The Resulting Log

```
NOTE: CALL EXECUTE generated line.
1   + PROC PRINT DATA=PERM.COUNTYDT;
      WHERE COUNTYNM="ASHLAND";
      OPTIONS PAGENO=1;
      TITLE "REPORT FOR COUNTY ASHLAND";
      FOOTNOTE "TOTAL OBSERVATION COUNT WAS 2";
      RUN;

2   + PROC PRINT DATA=PERM.COUNTYDT;
      WHERE COUNTYNM="BAYFIELD";
      OPTIONS PAGENO=1;
      TITLE "REPORT FOR COUNTY BAYFIELD";
      FOOTNOTE "TOTAL OBSERVATION COUNT WAS 3";
      RUN;


   ...
```

The RESOLVE Function

Resolves the value of a text expression during DATA step execution.

Syntax:

variable=RESOLVE(argument);

Notes:

if argument is single quoted string, resolves during data step execution.
If double quoted string and a macro element, resolves during data step compile.
RESOLVE allows you to delay resolution until data step executes.
    For example: to use macro variables created in data step.
RESOLVE Comparisons
RESOLVE resolves values during DATA step execution, where macro references resolve during scan.
RESOLVE accepts a wider variety of arguments than SYMGET(one variable) does.
When a macro variable contains additional macro references, RESOLVE will resolve, but SYMGET does not.
If argument is non-existent, RESOLVE returns unresolved reference, SYMGET returns a missing value.
Because it's more flexible, RESOLVE takes slightly more resources.

A RESOLVE Example
Various uses of CALL RESOLVE.

```
%let event=Holiday;
%macro mdate;
```

```
4th of July
%mend mdate;
data test;
length var1-var3 $ 15;
when='%mdate';
var1=resolve('&event');  /* macro variable reference      */
var2=resolve('%mdate');  /* macro invocation              */
var3=resolve(when);      /* DATA step var with macro call */
put '*** ' var1= var2= var3=;
run;

*** var1=Holiday var2=4th of July var3=4th of July
```

The SYMEXIST Function
Test for existence of a macro variable.

Syntax:

SYMEXIST(macro variable);

Notes:

macro variable can be a quoted string
macro variable can also be data step variable containing name of a macro variable
returns 1 if variable exists, otherwise 0.

A SYMEXIST Example
Test two variables for existence.

```
%let a=yes;
data _null_;
 if symexist('a') then
    put '**** a exists';
 else
    put '**** a doesnt exist';
if symexist('b') then
    put '**** b exists';
 else
    put '**** b doesnt exist';
run;
```

Partial SAS log:

```
**** a exists
**** b doesnt exist
```

The SYMLOCAL Function
Test for local scope.

Syntax:

SYMLOCAL(argument);

Notes:

argument can be a quoted string
argument can also be data step variable or expression containing name of a macro variable
returns 1 if variable is local, otherwise 0.

The SYMGLOBL Function
Test for global scope.

Syntax:

SYMGLOBL(argument);

Notes:

argument can be a quoted string
argument can also be data step variable or expression containing name of a macro variable
returns 1 if variable global, otherwise 0.

A SYMGLOBL, SYMLOCAL Example
Test for Scope.

```
%let c=yes;
%macro test;
%let d=yes;
data _null_;
 if symlocal ('c') then  put '**** c a is local';
 if symglobl('c') then   put '**** c is global';
 if symlocal ('d') then  put '**** d is local';
 if symglobl('d') then   put '**** d is global';
run;
%mend test;
%test
```

SAS Log
**** c is global
**** d is local

The SYMDEL Call Routine
Deletes the specified variable in global symbol table.

Syntax:

CALL SYMDEL(macro variables <, NOWARN>);

Notes:

macro variable can be a quoted string
macro variable can also be data step variable containing name of a macro variable.

A CALL SYMDEL Example
Create a variable and then delete it.

```
%let a=yes;
data _null_;
 if symexist('a') then
    put '**** a exists';
 call symdel('a');
 if symexist('a') then
    put '**** a still exists';
 else
    put '**** a doesnt exist';
run;
```

SAS Log:

```
**** a exists
**** a doesnt exist
```

Using DATA Step Functions In Macros

%SYSFUNC and %QSYSFUNC can call most DATA step functions from the macro language.

Syntax:

%SYSFUNC (function(argument(s))<, format>)
%QSYSFUNC (function(argument(s))<, format>)

Notes:

Most DATA step functions can be used except DIF, DIM, HBOUND, IORCMSG, INPUT, LAG, LBOUND, MISSING, PUT, RESOLVE, SYMGET, variable information functions

A %SYSFUNC Example
Print X that exists and give message for Z that doesn't exist.

```
%macro dsexist(dsn);
%if %sysfunc(exist(&dsn)) %then
   %do; proc print data=&dsn;
        title "&dsn";run;
   %end;
%else
   %put *** &dsn doesnt exist;
%mend dsexist;
%dsexist(x)
MPRINT(DSEXIST):   proc print data=x;
MPRINT(DSEXIST):   title "x";
MPRINT(DSEXIST):   run;
%dsexist(z)
*** z doesnt exist
```

A %SYSFUNC Example Using Formatting
Reformat the current date using the worddate format.

```
title "%sysfunc(date(),Worddate.) Final Report";
```

Generates:

```
 title "February 11, 2005 Final Report";
```

PROC SQL Interface

The INTO clause stores SQL select values into macro variables

Syntax:

INTO : mac-var-spec-1 < ..., : macro-variable-specification-n>

Notes

INTO is much like SYMPUT
INTO combines the power of SQL with the macro language.

An INTO Example

A Dataset With Names and Ages

Class Dataset

Obs	Name	Age
1	Steve	44
2	Tom	43
3	Jen	22

An INTO Example

First select a known number of Names and Ages into macro variables.

```
proc sql noprint;
  select  distinct name,
          age
          into:mname1-:mname3,
              :mage1-:mage3
          from class;
quit;

%put *** mname1=&mname1 mname2=&mname2 mname3=&mname3;
%put *** mage1=&mage1 mage2=&mage2 mage3=&mage3;
```

Partial Log:

```
*** mname1=Jen mname2=Steve mname3=Tom
*** mage1=22 mage2=43 mage3=44
```

An INTO Example
If you don't know how many Names exist, let SQL count them.

```
proc sql noprint;
  select  left(put(count(distinct name),3.))
           into:mtotct from class;
  select  distinct name,
          age
          into:mname1-:mname&mtotct,
              :mage1-:mage&mtotct from class;
quit;
%put *** mtotct=&mtotct;
%put *** mname1=&mname1 mname2=&mname2 mname3=&mname3;
%put *** mage1=&mage1 mage2=&mage2 mage3=&mage3;
Partial Log:
*** mtotct=3
*** mname1=Jen mname2=Steve mname3=Tom
*** mage1=22 mage2=43 mage3=44
```

An INTO Example
SQL can store all values in a single macro variable and separate, quote.

```
proc sql noprint;
    select  distinct quote(name),
          age
          into:mnames separated by ', ',
              :mages separated by ', ' from class;
quit;

%put *** mnames=&mnames;
%put *** mages=&mages;

Partial Log:

*** mnames="Jen     ", "Steve   ", "Tom     "
```

```
*** mages=22, 43, 44
```

Interfaces to SAS Component Language

Similar interfaces to DATA step.

Interface	Description
SYMGET function	returns macro variable at execution
SYMGETNfunction	returns numeric macro variable
CALL SYMPUT routines	assigns SCL values to macro variables
CALL SYMPUTN routines	assigns numeric values to macro variables

An SCL Example
Create a macro variable with SCL and reference it within the SUBMIT block with &.

```
MAIN:
      . . .
    CALL SYMPUT('MYR',PUT(YR,4.));
    SUBMIT CONTINUE;
     DATA SELECTS;
       SET BIRTHS.CERTS;
      IF &MYR=YEAR(INFBDATE);
    ENDSUBMIT;                    /*EXECUTE CODE, CONTINUE*/
      . . .
    PROC PRINT DATA=TOP10;
     BY INFSEX;
     VAR INFFIRST;
     TITLE "10 MOST POPULAR NAMES IN YEAR &MYR";RUN;
    ENDSUBMIT;
    RETURN;
```

Interfaces to SAS/Connect

Create and retrieve macro values to and from remote servers.

Syntax:

%SYSLPUT macro-variable=<value</REMOTE=remote-session-id>>;
%SYSRPUT local-macro-variable=remote-macro-variable;

Notes;
%SYSLPUT creates a macro variable on a remote server.
%SYSRPUT retrieves a macro variable on a remote server.

A SAS/CONNECT Example
Assign contents of local macro variable &ldsn to a remote macro variable.

```
%syslput rdsn=&ldsn;
 rsubmit;
  proc print data=&rdsn;
  title "Dataset &rdsn":
 endrsubmit;
```

Notes:
%PUT with system variables can show you system and values.

A SAS/CONNECT Example
Assign contents of remote macro variable &sysinfo to local macro variable.

```
rsubmit;              /* executes on the remote host. */
proc download data=remote.data1 out=local.data1;
run;
   /* RETCODE is on local, SYSINFO is on remote host. */
%sysrput retcode=&sysinfo;
endrsubmit;
%macro testit;         /* executes on the local host. */
 %if &retcode = 0 %then
   %do;
    further processing on local host
   %end;
%mend testit;
%testit
```

Interfaces to Operating System
%SYSEXEC executes operating system commands.

Syntax:

%SYSEXEC(system-command);

Notes:
Any return code returned is assigned to macro variable SYSRC.

A %SYSEXEC Example
Run a Clist on Z/OS or a batch file in Windows.

```
%macro testlib;
%if %upcase(&sysscp)=OS %then
     %sysexec ex 'my.clist(utility) ';
%else %if %upcase(&sysscp)=WIN %then
         %sysexec utility.bat;
      %else %put NO UTILITIES AVAILABLE ON &sysscp..;
%mend testlib;

%testlib
```

Retrieving Environment Variables
%SYSGET returns variables from your operating system.

Syntax:

%SYSGET(environment-variable-name);

Example:
Get the logon id in UNIX

```
%let name=%sysget(USER);
```

```
%put Userid running is &name;
```

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 authors at:

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

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.