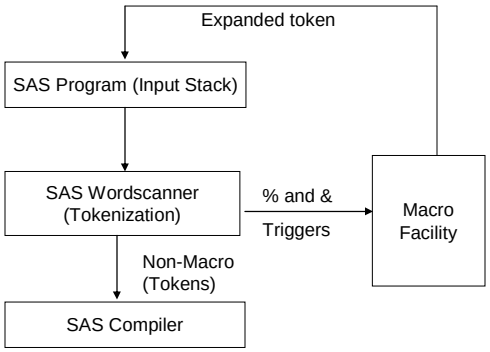
Intermediate and Advanced SAS® Macros



SYSTEMS SEMINAR CONSULTANTS, INC.

Steven First and Tim Broeckert
2997 Yarmouth Greenway Drive, Madison, WI 53711
Phone: (608) 278-9964, Web: www.sys-seminar.com

Intermediate and Advanced SAS® Macros

This course was written by Systems Seminar Consultants, Inc. SSC specializes SAS software and offers SAS:

- Training Services
- Consulting Services
- Help Desk Plans
- Newsletter subscriptions to *The Missing Semicolon™.*

Objectives

After completing this class students will:

- understand how the macro system fits with the rest of SAS software
- Understand the various macro interfaces
- Be better SAS programmers.

SAS Macro Overview

Macros construct input for the SAS compiler.

Functions of the SAS macro processor:

- pass symbolic values between SAS statements and steps
- establish default symbolic values
- conditionally execute SAS steps
- invoke very long, complex code in a quick, short way.

Notes:

- The MACRO PROCESSOR is the SAS system module that processes macros.
- The MACRO LANGUAGE is how you communicate with the processor.

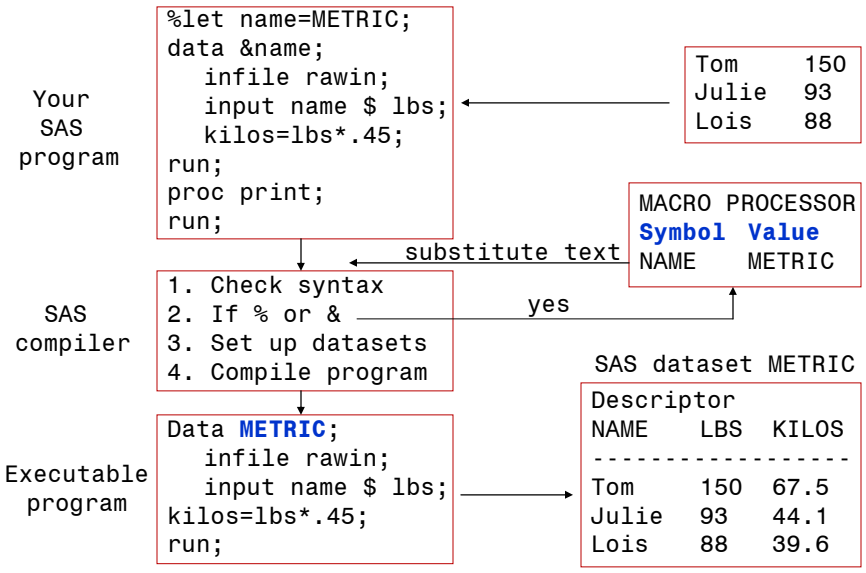
The SAS Macro Language

A second SAS programming language for string manipulation.

Characteristics:

- 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 Processor Flow Example



Automatic Macro Variables

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
```


Automatic Macro Variables (continued)

```
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
```

Example:

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

Resolves to:

```
FOOTNOTE "THIS REPORT WAS RUN ON FRIDAY, 17MAR06";
```

Displaying Macro Variables

%PUT displays macro variables to the log at compile time.

Syntax:

```
%PUT text macrovariables ;
%PUT _all_ | _user_;
```

Example:

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

Partial SAS Log:

```
***** 17MAR06 *****
```


Displaying All Macro Variables

%PUT can display all current macro variables.

```
%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 17MAR06
AUTOMATIC SYSDAY Friday
```

Partial SAS LOG Continued

```
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
```


The “Black Hole” of Macros

Other errors that can cause SAS to ignore statements that are submitted.

Examples:

- Missing or wrong spelling on %MEND statement.
- Mismatched quotes, comments.

Again, submit the following statement until SAS gives the message below.

```
*'; *"; *); */; %mend; run;
```

The Resulting Log:

```
ERROR: No matching %MACRO statement for this %MEND
       statement.
```

You can now fix the program and resume.

Practice Exercises

Exercise 1:

- A program that will build several datasets used later in this course is provided on your system. There should be about six datasets.
- Start SAS for this workshop WS107.
- Check the SAS log to insure that the program ran correctly, use the LIBNAME command, file icon, or submit:

```
proc contents data=work._all_;
 run;
```

  to verify that the datasets were built correctly.
- Explore SAS if you would like to.

Practice Exercises

Exercise 2:

- Determine the values of the following system macro variables using your current computer system.

```
&SYSDAY          Current day of the week
&SYSTIME         Current time
&SYSSCP          Operating system being used
&SYSVER          Current SAS version number
&SYSDATE         Current date, in DATE7. Format
```

Exercise 3:

- Using system macro variables run a PROC CONTENTS and a PROC PRINT on the LAST SAS dataset that was created. Include its name in a title.

Exercise 2 Solution

```
%PUT **** SYSDAY  = &SYSDAY;
%PUT **** SYSTIME = &SYSTIME;
%PUT **** SYSSCP  = &SYSSCP;
%PUT **** SYSVER  = &SYSVER;
%PUT **** SYSDATE = &SYSDATE;
```

Log:

```
%PUT **** SYSDAY  = &SYSDAY;
 **** SYSDAY  = Friday
2    %PUT **** SYSTIME = &SYSTIME;
**** SYSTIME = 13:42
3    %PUT **** SYSSCP  = &SYSSCP;
**** SYSSCP  = WIN
4    %PUT **** SYSVER  = &SYSVER;
**** SYSVER  = 6.12
5    %PUT **** SYSDATE = &SYSDATE;
**** SYSDATE = 17MAR06
```


Exercise 3 Solutions

```
proc contents data=&syslast;
 title "contents of &syslast";
run;
```

The Generated Code:

```
proc contents data=WORK.COUNTYDT;
 title "contents of WORK.COUNTYDT";
run;
```

```
contents of WORK.COUNTYDT
   CONTENTS PROCEDURE
Data Set Name:WORK.COUNTYDT               Observations:        6
Member Type:  DATA                        Variables:           2
Engine:       V612                        Indexes:             0
Created:      13:51 Friday, March 17, 2006 Observation Length:  23
Last Modified:13:51 Friday, March 17, 2006 Deleted Observations:0
Protection:                               Compressed:          NO
Data Set Type:                            Sorted:              NO
Label:
```

Macro Variables

- 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;
```


A Problem Needing More than a Macro Variable

The following dataset contains county data.

Data Set COUNTYDT

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

Macro Application (continued)

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?

Answer: A combination of data steps, proc steps, macro statements, and interfaces between them.

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

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.

DATA Step Interfaces

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

Note:

The functions have corresponding macro functions in macro language.

Retrieving Macro Variable Values

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

Syntax:

DATAstepvariable=**SYMGET**(*argument*);

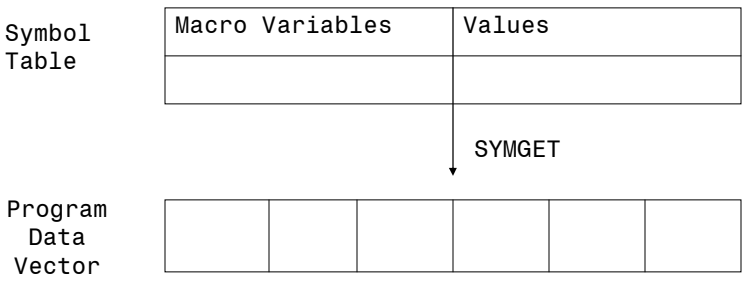
***argument* can be:**

- a character literal (Ex. 'MDEPT')
- a DATA step character variable (Ex. DEPT)
- a character expression (Ex. 'MDEPT' !! '01')

Retrieving Macro Variable Values (continued)

***DATAstepvariable* can be:**

- predefined as character or numeric
- a 200 byte character variable if not predefined
- truncated if the value returned exceeds 200 characters (prior to v7)
- numeric if used in a numeric expression



Another SYMGET Example

A valid DATA step variable containing a macro variable name can be used.

Start → *End*

```
%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;
```

Symbol Table

NAME	VALUE
LANGUAGE	SAS
COURSE	MACRO
ENV	INTERACTIVE

Program Data Vector

WORDS	NAME

Another SYMGET Example (continued)

The generated output:

COURSE DATASET

OBS	WORDS	NAME
1	ENV	INTERACTIVE
2	LANGUAGE	SAS
3	COURSE	MACRO

Storing Macro Variable Values

SYMPUT copies values of DATA step variables into macro variables.

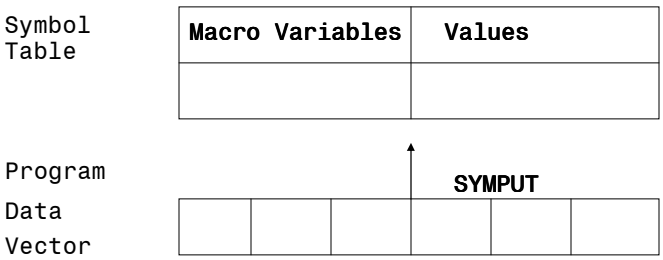
Syntax:

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.

Storing Macro Variable Values (continued)



Notes:

- Values assigned by SYMPUT will not be available to the compiler until the **NEXT** SAS step.

A DATA Step Variable Example

The contents of VAR1 names the macro variables, and the contents of VAR2 becomes the contents of the macro variables.

```
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;
```

Start → End

Program Data Vector

VAR1	VAR2

Symbol Table

Name	Value

A DATA Step Variable Example (continued)

Partial log:

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

Notes:

- The RUN is critical to insure the %PUT statements are compiled in a later step.

SYMPUT With a Numeric Value

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

Start → End

```
%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;
```

Program Data Vector

N	OBSNO	TOTOBS

Symbol Table

NAME	VALUE

CALL SYMPUTX

SYMPUTX will automatically convert and trim numbers.

Syntax:

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

Notes:

- 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.

A SYMPUTX Example

A much simpler call.

Start ⟶ End

```
%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;
```

Program Data Vector

N	OBSNO	TOTOBS

Symbol Table

NAME	VALUE

Practice Exercises

Exercise 4:

- The following program will read the first observation from the COUNTYDT dataset. Type the program and insert a CALL SYMPUT statement that will create a macro variable called MREADING from the data step variable READING. Try to minimize SAS conversion messages and warnings. Run the program until it is correct.

```
data _null_;
 set countydt (obs=1);
                <==== call symput statement here.
 run;
 %put *** mreading=&mreading;
```

Exercise 5:

- Alter the program above to use a CALL SYMPUTX statement. Which do you prefer: SYMPUT or SYMPUTX?

Exercise 4 Solution

```
DATA _NULL_;
  SET CountyDt(OBS=1);
  CALL SYMPUT('MReading',LEFT(PUT(Reading,8.)));
RUN;

%PUT *** mreading=&mreading;
```

LOG:

```
1  DATA _NULL_;
2    SET CountyDt(OBS=1);
3    CALL SYMPUT('MReading',LEFT(PUT(Reading,8.)));
4  RUN;
NOTE: There were 1 observations read from the data set
   WORK.COUNTYDT.
NOTE: DATA statement used:
      real time           0.13 seconds
      cpu time            0.02 seconds
5  %PUT *** mreading=&mreading;
*** mreading=125
```


Exercise 5 Solution

```
DATA _NULL_;
  SET CountyDt(obs=1);
  CALL SYMPUTX('MReading',Reading);
RUN;

%PUT *** mreading=&mreading;
```

LOG:

```
1  DATA _NULL_;
2    SET CountyDt(obs=1);
3    CALL SYMPUTX('MReading',Reading);
4  RUN;

NOTE: There were 1 observations read from the data set
   WORK.COUNTYDT.
NOTE: DATA statement used:
      real time           0.13 seconds
      cpu time            0.02 seconds

5  %PUT *** mreading=&mreading;
*** mreading=125
```

Solving The County Problem

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 macro can then use the macro variables to generate PROC PRINTs.

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
```

CALL EXECUTE

Resolves an argument and executes the resolved value at step boundary.

Syntax:

CALL EXECUTE(*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.
- 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 countydt;
  by countynm;
  if first.countynm then ctyobs=0;
  ctyobs+1;
  if last.countynm then
     call execute('PROC PRINT DATA=COUNTYDT; '          !!
                  'WHERE COUNTYNM="'                    !!
                   countynm                             !!
                  '";'                                  !!
                  'OPTIONS PAGENO=1;'                   !!
                  'TITLE "REPORT FOR COUNTY '           !!
                  countynm                              !!
```

A Call Execute Example (continued)

The rest of the program follows.

```
                '";'                                        !!
                'FOOTNOTE "TOTAL OBSERVATION COUNT WAS ' !!
                put(ctyobs,2.)                              !!
                '";'                                        !!
                'RUN;'
                );
run;
```


A Call Execute Example (continued)

The resulting SAS Log:

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

Call Execute Using a Macro

First, define a macro to do most of the work.

```
%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;
```


Call Execute Using a Macro

Now, call execute with a macro call. The macro executes immediately, the generated code will run after this step ends.

```
data pass2;
 set 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=COUNTYDT;
      WHERE COUNTYNM="ASHLAND";
      OPTIONS PAGENO=1;
      TITLE "REPORT FOR COUNTY ASHLAND";
      FOOTNOTE "TOTAL OBSERVATION COUNT WAS 2";
      RUN;

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


Practice Exercises

Exercise 6:

- The following data step will read all rows of the COUNTYDT dataset. Type it in and code a CALL execute statement that will submit a separate PROC PRINT step for each value of COUNTYNM. Include a title and a run statement for each report.
- Note: Be careful of mismatched quotes.

```
data _null_;
 set countydt;
 by countynm;
 if last.countynm then
   do;
        ←====  code call execute here
   end;
run;
```

Exercise 6 Solution

```
DATA _NULL_;
  SET CountyDt;
  BY CountyNm;
  IF LAST.CountyNm THEN
    DO;
      CALL EXECUTE(
                  "PROC PRINT DATA=CountyDt;"    !!
                  "WHERE CountyNm ='"            !!
                   CountyNm                      !!
                  "';"                           !!
                  "TITLE 'Report for county "    !!
                   CountyNm                      !!
                  "';RUN;"
                  );
    END;
RUN;
```


Exercise 6 Log (Partial)

```
NOTE: There were 6 observations read from the data set
   WORK.COUNTYDT.
NOTE: DATA statement used:
      real time           0.04 seconds
      cpu time            0.02 seconds
NOTE: CALL EXECUTE generated line.

1   + PROC PRINT DATA=CountyDt;WHERE CountyNm ='ASHLAND
   ';TITLE 'Report for county ASHLAND        ';RUN;
NOTE: There were 2 observations read from the data set
   WORK.COUNTYDT.
      WHERE CountyNm='ASHLAND        ';
NOTE: PROCEDURE PRINT used:
      real time           0.40 seconds
      cpu time            0.07 seconds

2   + PROC PRINT DATA=CountyDt;WHERE CountyNm ='BAYFIELD
   ';TITLE 'Report for county BAYFIELD       ';RUN;
NOTE: There were 3 observations read from the data set
   WORK.COUNTYDT.
      WHERE CountyNm='BAYFIELD       ';
NOTE: PROCEDURE PRINT used:
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Exercise 6 Output

```
Report for county ASHLAND

Obs    COUNTYNM    READING

 1     ASHLAND       125
 2     ASHLAND       611


Report for county BAYFIELD

Obs    COUNTYNM    READING

 3     BAYFIELD      101
 4     BAYFIELD      101
 5     BAYFIELD      222


Report for county BAYFIELD

Obs    COUNTYNM    READING

 6    WASHINGTON     101
```


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
```

A PROC SQL Solution To The County Problem

PROC SQL can create 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    */
```


The Macro Is Unchanged

```
%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          */
```

Note: The output is exactly the same as the earlier COUNTYMC invocation.

Practice Exercises

Exercise 7:

- The COUNTYDT dataset contains 3 unique values in the COUNTYNM field.
- Use PROC SQL to place the unique values of COUNTYN into separate macro variables. Display those values with %PUT _USER_;

Exercise 8:

- Use PROC SQL to place all values of COUNTYNM into a single macro variable. Wrap quotes around the values and separate them with commas.

Exercise 7 Solution

```
PROC SQL NOPRINT;
  SELECT LEFT(PUT(COUNT(DISTINCT CountyNm),3.))
    INTO :MTotCt
    FROM CountyDt;
  SELECT DISTINCT CountyNm
    INTO :MCountyNm1 - :MCountyNm&MTotCt
    FROM CountyDt;
QUIT;
%PUT _USER_;
```

LOG:

```
11  %PUT _USER_;
GLOBAL SQLOBS 3
GLOBAL SQLOOPS 34
GLOBAL MTOTCT 3
GLOBAL MCOUNTYNM1 ASHLAND
GLOBAL MCOUNTYNM2 BAYFIELD
GLOBAL MCOUNTYNM3 WASHINGTON
GLOBAL SQLXOBS 0
GLOBAL SQLRC 0
```

Exercise 8 Solution

```
PROC SQL NOPRINT;
  SELECT DISTINCT QUOTE(CountyNm)
  INTO :MCountyNm SEPARATED BY ","
  FROM CountyDt;
QUIT;
%PUT MCountyNm=&MCountyNm;

LOG:
1  PROC SQL NOPRINT;
2    SELECT DISTINCT QUOTE(CountyNm)
3    INTO :MCountyNm SEPARATED BY ","
4    FROM CountyDt;
5  QUIT;
NOTE: PROCEDURE SQL used:
      real time           0.01 seconds
      cpu time            0.01 seconds

7  %PUT MCountyNm=&MCountyNm;
MCountyNm="ASHLAND        ","BAYFIELD       ","WASHINGTON     "
```


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
```

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
```

Why were single quotes used?

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 does not exist';
if symexist('b') then
    put '**** b exists';
 else
    put '**** b does not exist';
run;
```

Partial SAS log:

```
**** a exists
**** b does not 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 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";
```


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 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;
```

Practice Exercises

Exercise 9:

- A very common problem is to run a proc against all members of a SAS library.
- The following program produces a SAS dataset containing an observation for each variable within each dataset in the library. A partial print is shown below.

```
PROC CONTENTS DATA=WORK._ALL_ NOPRINT OUT=CONTOUT;
 RUN;
PROC PRINT DATA=CONTOUT;
VAR LIBNAME MEMNAME NAME;
TITLE 'CONTOUT';
RUN;
```


The Resulting Dataset

CONTOUT

OBS	LIBNAME	MEMNAME	NAME
1	WORK	FREQ	DEPT
2	WORK	FREQ	SALARIES
3	WORK	FREQ	SALES
4	WORK	FREQ	YEAR
5	WORK	YEAR1988	EMPLOYDT
6	WORK	YEAR1988	EXPENSES
7	WORK	YEAR1988	NAME
8	WORK	YEAR1988	REGION
9	WORK	YEAR1988	SALES
10	WORK	YEAR1988	STATE

Directions:

- Write a SAS macro that will generate a separate PROC PRINT of all members in the library with an appropriate title.
- Note you will only want to produce one print per member, not one per variable.

Exercise 9 Solution (first step)

```
PROC CONTENTS DATA=WORK._ALL_ NOPRINT OUT=CONTOUT;
 RUN;
 PROC PRINT DATA=CONTOUT;
 VAR LIBNAME MEMNAME NAME;
 TITLE 'CONTOUT';
RUN;
```


Exercise 9 Solution (remainder)

```
DATA ONEMEM;
 SET CONTOUT END=EOF;
  BY MEMNAME;
  IF LAST.MEMNAME;
  KTR+1;
  CALL SYMPUT ('MMEM'||LEFT(PUT(KTR,5.)),MEMNAME);
  IF EOF;
  CALL SYMPUT ('MTOTOBS',LEFT(PUT(KTR,5.)));
RUN;
%MACRO PRTLOOP;
 %DO I = 1 %TO &MTOTOBS;
     PROC PRINT DATA=&&MMEM&I;
        TITLE "&&MMEM&I";
     RUN;
 %END;
%MEND PRTLOOP;
%PRTLOOP
```

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.

- Anyone who would like the macro should please leave a business card.

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     */
/*                                                       */
/*********************************************************/
```

SSCFLAT Macro Partial Source (continued)

```
%MACRO SSCFLAT(MSASDS=,                    /*INPUT SAS DS (REQUIRED       */
   MPREFIX=&SYSPREF..,                     /*OUT PREF, OR DIR OUT         */
   MFLATOUT=&MPREFIX&MMEMNAME..DAT,        /*FLATFILE OUT                 */
   MHEADER=YES,                            /*FIELD NAMES IN FIRST REC     */
   MLIST=YES,                              /*PRINT FLAT FILE IN LOG?      */
   MTRIMNUM=YES,                           /*TRIM NUM TRAIL BLANKS?       */
   MTRACE=NO,                              /*DEBUGGING OPTION             */
   MMISSING=".",                           /*MISSING VALUE CHARACTER      */
   MLRECL=6160,                            /*LARGEST RECORD LENGTH        */
   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;
```

A SSCFLAT Macro Example (continued)

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.

Contact Us

SYSTEMS SEMINAR CONSULTANTS, INC.
SAS® Training, Consulting, & Help Desk Services

Steven J. First
Tim Broeckert
President

(608) 278-9964
FAX (608) 278-0065
sfirst@sys-seminar.com
www.sys-seminar.com

2997 Yarmouth Greenway Drive • Madison, WI 53711