SAS MACRO TUTORIAL
S. Morton and Phil Winters, SAS Institute

MACRO DOCUMENTATION

SAS users have a new facility available in SAS82:

The SAS macro language.

*****************************************************************************

DOCUMENTED IN:


TECHNICAL REPORT P-127: "ENHANCEMENTS TO THE SAS MACRO FACILITY"

MACRO FACILITY AND OTHER IPPS

The family of Institute Program Products (IPPs):

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>macro</td>
<td>BASE SAS</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>mainstream</td>
</tr>
<tr>
<td>step</td>
<td>PROCedures</td>
</tr>
<tr>
<td>compiler</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>graphics</td>
<td>SAS/GRAPH</td>
</tr>
<tr>
<td>PROCedures</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ETS</td>
<td>SAS/ETS</td>
</tr>
<tr>
<td>PROCedures</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>FSP</td>
<td>SAS/FSP</td>
</tr>
<tr>
<td>PROCedures</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>SAS/OR</td>
</tr>
<tr>
<td>PROCedures</td>
<td></td>
</tr>
</tbody>
</table>
```

The macro facility is part of the base SAS product.
APPLICATIONS OF MACRO

The Macro language is intended to allow a user to:

• extract system information known to the SAS system
• perform conditional execution of SAS DATA and PROC steps
• generate data-dependent SAS statements (SAS code) and SAS steps
• generate repetitive SAS code
• develop user-friendly systems
• develop line-prompted and full-screen menu systems
• communicate information between SAS steps
• develop production SAS systems.

FUNCTIONAL VIEW OF IPPS

How the SAS macro language is related to the Institute Program Products:

```
  SAS SOURCE STATEMENTS
    ↓
  MACRO LANGUAGE
    ↓
  RAW DATA
    ↓
  SAS SUPERVISOR
    ↓
  DATA STEP
    ↓
  FORMAT LIBRARY
    ↓
  PROCEDURE LIBRARIES
```

358
MAJOR USES OF MACRO

In most cases, the major uses of the macro language are to:

- conditionally execute PROC or DATA steps
- repeatedly execute a group of statements
- transfer information across SAS steps
- package SAS tasks into systems.

WHAT IS THE MACRO LANGUAGE?

- The SAS macro language is a programming language for string manipulation.

- Strings are sequences of characters. Everything is a character string to the macro facility.

- The strings will typically be SAS code or pieces of SAS code.

WHAT IS THE MACRO LANGUAGE?

The macro language has three basic constructs:

1. Symbolic macro variables
   SAS naming rules apply

2. Macro statements
   start with a percent sign (%) and a macro keyword and end with a semicolon (;)

3. Model text
   usually SAS statements or parts of SAS statements
WHAT IS THE MACRO LANGUAGE?

Macro variables
• store string values
• manipulate string values
• act as parameters.

Macro statements
• manipulate variables
• are used for assignment and substitution
• are used for branching and conditional execution.

NOTE: macro variables are not SAS DATA step variables.

SIMPLE VIEW OF MACRO WORKINGS

When SAS macro statements are used in a SAS program, they are first processed by the macro "preprocessor."

1. The macro preprocessor resolves the macro statements and macro variables and produces standard SAS code.

2. This generated code becomes part of the code that makes up your SAS program.

3. The resulting SAS program is then passed to the SAS compiler for execution.

NOTE: The macro processor doesn't preprocess the whole SAS job but it is more tightly coupled to SAS.
ADDITION OF THE MACRO FACILITY

When necessary the supervisor transfers control to the macro facility.

---+
1 1 1 MACRO
SUPERVISOR

MACRO
FACILITY

---+
SAS CODE
SAS CODE

COMPILE
EXECUTE

The macro facility emits SAS code that is then returned to the supervisor.

ROLE OF THE SAS WORDSCANNER

TWO tokens trigger the wordscanner to notify the macro facility to look at the INPUT STREAM:

% &

If either of these tokens is followed by a non-blank, the macro facility takes control of the wordscanner.
MACRO VARIABLE RULES

- Each macro variable has a NAME. SAS naming rules apply (1 to 8 characters beginning with a letter or underscore).

- A macro variable is REFERENCED (called) anywhere in a SAS job by placing an ampersand (&) immediately before the macro variable name.

- A macro variable reference tells SAS to replace the reference with the value of the macro variable.

- An optional period (.) delimits a macro variable reference.

NOTE: The length of a macro (symbolic) variable value range is zero to the maximum defined for your installation. Maximum length ranges from 1024 to 26672 characters. (See your computing center staff for the maximum length at your installation.)

TYPES OF MACRO VARIABLES

Macro variables are defined:
- by the SAS supervisor
- by the USER.

```
MACRO VARIABLES
(symbolic variables)

USER-DEFINED

SAS DEFINED

AUTOMATIC
```
Automatic variables:

- are created when SAS is invoked
- retain their values for the duration of the SAS job
- reflect information about the SAS job.

Some automatic variables can be changed by the user.

Automatic variables: list

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSBUFFR</td>
<td>holds a string entered in response to a %INPUT statement.</td>
</tr>
<tr>
<td>SYSDATE</td>
<td>date the SAS job executed.</td>
</tr>
<tr>
<td>SYSDAY</td>
<td>day of the week the SAS job began execution.</td>
</tr>
<tr>
<td>SYSDEVIC</td>
<td>name of the current graphics device.</td>
</tr>
<tr>
<td>SYSDSN</td>
<td>name of the most recently created SAS data set in the current SAS job.</td>
</tr>
<tr>
<td>SYSENV</td>
<td>mode of execution - FORE (as in foreground) or BACK (as in background).</td>
</tr>
<tr>
<td>SYSINDEX</td>
<td>number of macros that have started execution in the current SAS job.</td>
</tr>
<tr>
<td>SYSPARM</td>
<td>same as DATA step function SYSPARM().</td>
</tr>
<tr>
<td>SYSRC</td>
<td>last return code set by %TSO %CHS and macros</td>
</tr>
<tr>
<td>SYSSCP</td>
<td>operating system acronym--OS, CMS, or DOS.</td>
</tr>
<tr>
<td>SYSTIME</td>
<td>time the SAS job began execution.</td>
</tr>
<tr>
<td>SYSVER</td>
<td>version of SAS being used.</td>
</tr>
</tbody>
</table>

Automatic variables can be referenced anywhere in a SAS job.

NOTE: * indicates a variable that cannot be changed.
AUTOMATIC VARIABLES: USING

DOUBLE QUOTES:

SAS code:

TITLE "THIS IS THE REPORT FOR &SYSDATE";

SAS compiler sees: the reference is resolved

TITLE "THIS IS THE REPORT FOR JAN84";

SINGLE QUOTES:

SAS code:

TITLE 'THIS IS THE REPORT FOR &SYSDATE';

SAS compiler sees: the reference is NOT resolved

TITLE 'THIS IS THE REPORT FOR &SYSDATE';

%LET SYNTAX

%LET - used to define macro variables and assign values to them.

SYNTAX:

1. %LET macrovariablename = string ;
   ex: %LET MONTH=MAY;
   This defines a macro variable named MONTH.
   It has the value MAY.

2. %LET &macrovariablename = string ;
   ex: %LET &MONTH=THE BIRD OF PARADISE;
   This defines a macro variable named MAY.
   It has the value THE BIRD OF PARADISE.

The difference between these two forms is that a macro variable reference (&month) will be resolved before the %LET statement is executed.
%LET EXAMPLE

Using these statements in a SAS job:

%LET MONTH=MAY;
TITLE "&MONTH";

produces:

TITLE "MAY";

Adding these statements to the same SAS job:

%LET &MONTH=THE BIRD OF PARADISE;
TITLE2 "&MAY";

produces:

TITLE2 "THE BIRD OF PARADISE";

SAS uses the two titles for the top of the printed pages:

MAY
THE BIRD OF PARADISE

%GLOBAL %LOCAL STATEMENTS

Two statements can be used to explicitly define a macro variable's reference environment

%GLOBAL list of variables;

The %GLOBAL statement is used to make macro variables available to all referencing environments of a job. %GLOBAL can be used anywhere in a SAS job.

%GLOBAL STATE;

%LOCAL list of variables;

The %LOCAL statement is used to specify macro variables that are to be available to a single macro. %LOCAL can be used only inside a macro definition. %LOCAL is used to prevent resetting the value of a global variable or to prevent contaminating an index variable with a value from another macro. Useful in recursive calling of a macro.

%MACRO SLOCAL;
%LOCAL I;
%MEND SLOCAL;
%GLOBAL %LOCAL STATEMENTS

A situation occasionally arises that requires a macro variable that is by definition local (that is, a macro parameter) to be globally referenced (its value used by another macro).

A simple solution would seem to be:

```
%MACRO TEST(IN);
  %GLOBAL IN;
  %MEND TEST;

%MACRO DOIT;
  TITLE "THE VARIABLE &IN IS GLOBAL";
  %MEND DOIT;
```

However, calling test %TEST(INVALUE)
yields

```
1501:
ATTEMPT TO %GLOBAL A NAME WHICH EXISTS IN A LOCAL ENVIRONMENT
```

Macro parameters are local.

GLOBAL/LOCAL FEATURES

Global macro variables:

- are defined any time during a SAS job
- are available globally once defined
- retain their values across DATA and PROC steps
- have values that may be changed any time during the SAS
- are a method of communication among macros
- include automatic macro variables.

Local macro variables:

- can only be defined by or in a macro
- are available only inside the macro that defines them
- can retain their values across DATA and PROC steps if the DATA or PROC steps are all contained within the macro
- are primarily used as index or counting variables.
MACRO DEFINITION SYNTAX

The general form of a macro definition is:

```
%MACRO macroname(parameters)/options;
  macrotext
%MEND macroname;
```

- The macro name must be a valid SAS name.
  1 to 8 characters, letters, or _• (CMS: 7 character maximum).
- Macro parameters are macro variables for which values are supplied by the macro call or in the macro itself.
- The STMT option is the only option currently available with the %MACRO statement. This option changes the form of the macro call.
- %MEND is used to end the macro definition. macroname is not required BUT STRONGLY RECOMMENDED.
- MACROTEXT is generally SAS statements or parts of SAS statements along with macro programming statements.

PARAMETER RULES

Parameter definition rules:

- The parameter list is enclosed in parentheses (  ).
- Parameters are separated by commas.
- Parameters may be positional or keyword.
- When the two parameter types are mixed, positional parameters must be defined before keyword parameters.
PARAMETER DEFINITIONS

Positional parameters and keyword parameters define macro variables that can be used with the macro text.

SYNTAX of parameter list:

positional parameters
(SASname,SASname,SASname,...)

keyword parameters
(SASname=string of characters,SASname=string,...)

mixed parameters
(SASname,SASname,SASname=string,...)

PARAMETER DEFINITIONS

Example of positional parameters

%MACRO CLINT(GOOD,BAD,UGLY);
  DATA &GOOD;
  SET &BAD;
  BY &UGLY;
  IF FIRST.MOVIE;
%END CLINT;

Example of keyword parameters

%MACRO CLINT(GOOD=BEST,BAD=WESTERN,UGLY=MOVIE);
  DATA &GOOD;
  SET &BAD;
  BY &UGLY;
  IF FIRST.MOVIE;
%END CLINT;
STATEMENT OPTION OF A MACRO

When the STMT option is specified, the macro call:

• may NOT require a leading %
• DOES require a semicolon
• cannot have ( or ) for parameter specifications- no commas
• REQUIRES OPTIONS IMPLMAC.

NOTE: the STMT option is documented in TECHNICAL REPORT P-127.

RESERVED WORD TABLE CONTENTS

These are the reserved words currently in the list

| ABORT | ABEND |
| ACT   | ACTIVATE |
| BY    | CLEAR |
| CLOSE | CMS |
| COPY  | DISPLAY |
| DO    | DEACT |
| DEL   | DELETE |
| EVAL  | FILE |
| ELSE  | END |
| EDIT  | GLOBAL |
| GO    | GOTO |
| INDEX | INPUT |
| INFILE | IF |
| INC   | INCLUDE |
| LET   | LOCAL |
| LIST  | LSTM |
| MACRO | MEND |
| NRQUOTE | NRSTR |
| OPEN  | ON |
| PUT   | PAUSE |
| QUOTE | RESOLVE |
| RETURN | RUN |
| SUBSTR | SCAN |
| STOP  | SAVE |
| STR   | TSO |
| THEN  | TO |
| UNQUOTE | UNSTR |
| UPCASE | UNTIL |

These keywords MAY NOT be used as macro names.
CHANGING OLD Macros

Changing SAS79 macros to new SAS82 macros.

If you have existing programs with old SAS macros, you can improve efficiency by changing to the new macro syntax.

<table>
<thead>
<tr>
<th>SAS79 macro syntax</th>
<th>SAS82 macro syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACRO _VARS</td>
<td>%MACRO VARS;</td>
</tr>
<tr>
<td>STATE RANK70 RANK79 POP79 DENSITY MIGRATE;</td>
<td>STATE RANK70 RANK79 POP79 DENSITY MIGRATE;</td>
</tr>
<tr>
<td>%</td>
<td>%END VARS;</td>
</tr>
<tr>
<td>MACRO _PROCESS</td>
<td>%MACRO PROCESS;</td>
</tr>
<tr>
<td>DATA;</td>
<td>DATA;</td>
</tr>
<tr>
<td>INFILE STATESIN;</td>
<td>INFILE STATESIN;</td>
</tr>
<tr>
<td>INPUT _VARS</td>
<td>INPUT %VARS</td>
</tr>
<tr>
<td>%</td>
<td>%END PROCESS;</td>
</tr>
<tr>
<td>_PROCESS</td>
<td>%PROCESS</td>
</tr>
</tbody>
</table>

STRING FUNCTIONS: LIST

The following macro functions are currently available.

%INDEX(argument1,argument2) finds the first occurrence of argument2 in the string argument1

%LENGTH(argument) finds the length of an argument

%SCAN(argument,n,delimiter) scans for "word" in an argument using a delimiter

%SUBSTR(argument,start,stop) substrings a character string from an argument beginning with start for how many.

%UPCASE(argument) translates lowercase characters to uppercase.

Special string handling function:

%EVAL(expression) evaluates arithmetic and logical expressions.
%SCAN-SPECIAL DELIMITERS

%SCAN uses these default delimiters:

blank . < ( + | & ! $ ) ; ~ - / , %

There are two special cases:

Blank delimiter only

%LET NAME = SAMUEL T. GOMPERS ;
%LET FIRST = %SCAN(&NAME,1,);
%LET LAST = %SCAN(&NAME,3,%STR( ));

Comma delimiter only

%LET NAME = RONALD R. RAYGUN, JR. ;
%LET TITLE = %SCAN(&NAME,2,%STR(,));
WRONG:
%LET TITLE=%SCAN(&NAME,2,.)

%EVAL FUNCTION

Use the %EVAL function to perform computations.

• The SAS macro facility is a string (character) handling language. It does not compute.
• The %EVAL function is used to allow the macro language to do integer arithmetic.
• Macro issues an error message if you attempt to perform a calculation on non-integer values.
• If you do %LET X=%EVAL(1/3), the result is truncated.

The %EVAL function has the form:

%EVAL(expression or argument)

The statements:
%LET X=2;
%LET Y=%EVAL(6X+1);
Assign:
6X a value of 2
6Y a value of 3.

<table>
<thead>
<tr>
<th>The statements:</th>
</tr>
</thead>
<tbody>
<tr>
<td>%LET X=2;</td>
</tr>
<tr>
<td>%LET Y=6X+1;</td>
</tr>
<tr>
<td>Assign:</td>
</tr>
<tr>
<td>6X a value of 2</td>
</tr>
<tr>
<td>6Y a value of 3.</td>
</tr>
</tbody>
</table>
QUOTING FUNCTIONS: LIST

The following quoting functions are currently available:

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>%STR()</td>
<td>EXPRESS STRINGS CONTAINING SPECIAL CHARACTERS</td>
</tr>
<tr>
<td>%NRSTR()</td>
<td></td>
</tr>
<tr>
<td>%QUOTE()</td>
<td></td>
</tr>
<tr>
<td>%UNQUOTE()</td>
<td></td>
</tr>
</tbody>
</table>

These functions allow you to use characters that have a special meaning to the macro language without invoking their special meaning.

%STR FUNCTION

%STR hides the meaning of: (+ - & semicolons ) / = from the macro facility.

%STR is a compile time function of the wordscanner and should be used with literal values.

A semicolon (;) ends a macro program statement.

%STR allows you to use semicolons as string characters. %STR does not affect the percent sign (%) or the ampersand (&) references.

DATETIME CONSTANTS AND MACRO

%UNQUOTE is also needed to make datetime constants because of a wordscanner bug.

START=%UNQUOTE(%QUOTE(%'&SYSDATE%'D));

372
PROGRAM STATEMENTS: LIST

Many SAS DATA step statements also exist as macro program statements but begin with a percent sign (%).

%CHS
%COMMENT
%DO %END
iterative %DO %END
%DO %UNTIL %END
%DO %WHILE %END
%GOTO label;/%label:
%GLOBAL
%IF - %THEN %ELSE
%INPUT
%LET
%LOCAL
%MTRACO
%MEND
%PUT
%TSO

%PUT A BLANK LINE

%STR is the answer.

9?
%LET L2=%STR( );
10?
%INC 8;
FIRST LINE
THIRD LINE
%PUT SYNTAX

%PUT is used to write a line to the SAS log:

%PUT string;

STRING is a list of characters (including macrovariables) which does not contain a semicolon.
It is very useful for showing values of macro variables as a debugging aid.

%PUT Hello there gringo;

%LET WHO=Napoleon Flamingo;
%PUT Hello there gringo &WHO;

%PUT ignores leading and trailing blanks.
If you want to line up messages, use %STR.

NOTE: %PUT strings are truncated to current log file length.

%INPUT SYNTAX

%INPUT is used to input a string from the terminal:

%INPUT variable_list;

Variable_list is a list of macro variables separated by blanks.
%INPUT is documented in TECHNICAL REPORT P-127

%PUT and %INPUT can be used to make interactive systems

%PUT writes a line to the terminal.
%INPUT reads a line from the terminal.
TALKING MACRO

%MACRO PHREO;
%PUT HI, MY NAME IS PHREO. I AM A TALKING MACRO.;
%PUT WHAT IS YOUR NAME?;
%INPUT NAME;
%PUT WELL, &NAME THIS HAS BEEN FUN BUT I HAVE TO GO.;
%MEND PHREO;

%PUT %INPUT MENU

%MACRO FSCALC/STMT;
%------------------------------------;
% THIS MACRO LOOPS TO DO CONDITIONAL ;
% EXECUTION OF DATA AND PROC ;
% STEPS UNTIL THE USER WANTS TO STOP ;
% ------------------------------------;
%LET DONE = 0;
%DO %WHILE((&DONE = 0));
CLEAR;RUN;
%PUT PROC FSCALC: SPREAD SHEET APPLICATIONS;
%PUT %STRT( ) Please Select a demo;
%PUT %STRT( ) 0 Enter 0 to quit;
%PUT %STRT( ) 1 New Product Analysis;
%PUT %STRT( ) 2 Mortgage Schedule;
%INPUT CMDIN;
%IF &CMDIN = 1 &THEN &DO;
PROC FSCALC SCREEN=SAVE.DEM01;
RUN;
%END;
%ELSE_IF &CMDIN = 2 &THEN &DO;
PROC FSCALC SCREEN=SAVE.DEM02;
RUN;
%END;
%ELSE_IF &CMDIN = 0 &THEN &DO;
%LET DONE=1;
%PUT %STRT( );
%PUT Thank you for your interest in PROC FSCALC;
%END;
%ELSE %DO;
%PUT %STRT( );
%PUT Command &CMDIN not recognized. Please reenter;
%PUT %STRT( );
%END;
%END %DO %WHILE LOOP ---;
%MEND FSCALC;
MACRO HELP FRAMEWORK

%macro helper(which);
%*************** framework of a general helper macro ;
% for a SAS MACRO ;
% it can be called whenever necessary by ;
% %helper(A) ;
% which selection to do ;
%%%%%%%%%%%%%%%%%%%%%%%%%%;
%if &which eq %then %do;
% put %nrstr( );
% put %nrstr( general help information );
% put %nrstr( );
% put %nrstr( );
%end;
%else
%if %upcase(&which) eq A %then %do;
% put %nrstr( );
% put %nrstr( );
% put %nrstr( detailed help information );
% put %nrstr( );
%end;
%else
%if %upcase(&which) eq B %then %do;
% put %nrstr( );
% put %nrstr( );
% put %nrstr( );
% put %nrstr( );
%end;
%else %put no help available;
%mend helper;

CREATE MULTIPLE DATA SETS

Create a variable number of subsets with a single SET statement.

In the following example, a SAS data set named YEARLY contains a variable called MONTH with values 1 to 12.

```
%MACRO SUBSETS(HOWMANY, ROOTNAME=MONTH);
  %LET SETS = &ROOTNAME.1;
  %DO C = 2 %TO &HOWMANY;
    %LET SETS=&SETS &ROOTNAME&C;
  %END;
  DATA &SETS;
  SET YEARLY;
  %LET ELSE;
  %DO I = 1 %TO &HOWMANY;
    %ELSE IF MONTH = &I THEN OUTPUT &ROOTNAME&I;
    %LET ELSE=ELSE;
  %END;
  RUN;
%MEND SUBSETS;
```

The macro call,

```
%SUBSETS(4, ROOTNAME MONTH)
```

emits this code to SAS:

```
DATA MONTH1 MONTH2 MONTH3 MONTH4;
SET YEARLY;
 IF MONTH = 1 THEN OUTPUT MONTH1;
 ELSE IF MONTH = 2 THEN OUTPUT MONTH2;
 ELSE IF MONTH = 3 THEN OUTPUT MONTH3;
 ELSE IF MONTH = 4 THEN OUTPUT MONTH4;
RUN;
```

DATA STEP REVIEW

SYMGET function
used to assign the value of a macro variable to a SAS DATA step variable.

SYMPUT function
used in a SAS DATA step to create a macro variable whose value is that of a SAS DATA step variable.

These functions operate at DATA step execution time.
SYMPUT, SYMGET OVERVIEW

These two new DATA step functions can read and write from the symbol table to the program data vector (pdv) during execution of a DATA step.

<table>
<thead>
<tr>
<th>SAS DATA STEP</th>
<th>SYMBOL TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM</td>
<td>write</td>
</tr>
<tr>
<td>DATA</td>
<td>SYMPUT</td>
</tr>
<tr>
<td>VECTOR</td>
<td>read</td>
</tr>
<tr>
<td></td>
<td>SYMGET</td>
</tr>
</tbody>
</table>

SYMPUT SYNTAX

SYMPUT creates macro variables from DATA step variables.

CALL SYMPUT(argument1,argument2);

Argument1 names the macro variable. Argument2 assigns the macro variable value.
SYMPUT AND FORMATING EXAMPLE

This example illustrates using a DATA step to determine the beginning and ending date in a set of data and then using SYMPUT to create a formatted macro variable.

DATA DATES;
  RETAIN START STOP;
  INPUT DATE MDDYY6. SALES;
  IF _N_=1 THEN DO;
    START=DATE;
    STOP=DATE;
  END;
  IF START > DATE THEN START=DATE;
  IF STOP < DATE THEN STOP=DATE;
  RETURN;
EOF: CALL SYMPUT('START',PUT(START,DATE7.));
  CALL SYMPUT('STOP',PUT(STOP,DATE7.));
STOP;
CARDS;
010183 1000.00
011583 500.00
020183 1000.00
022383 10000
041083 0
040283 255.00
053083 255.00
RUN;

PROC PRINT;
  TITLE "SALES DATA FROM &START TO &STOP";
RUN;
SYMPUT AND FORMATING EXAMPLE

SALES DATA FROM 01JAN83 TO 30MAY83

<table>
<thead>
<tr>
<th>OBS</th>
<th>START</th>
<th>STOP</th>
<th>DATE</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8401</td>
<td>8401</td>
<td>8401</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>8415</td>
<td>8401</td>
<td>8415</td>
<td>500</td>
</tr>
<tr>
<td>3</td>
<td>8432</td>
<td>8401</td>
<td>8432</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>8454</td>
<td>8401</td>
<td>8454</td>
<td>10000</td>
</tr>
<tr>
<td>5</td>
<td>8500</td>
<td>8401</td>
<td>8500</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>8500</td>
<td>8401</td>
<td>8492</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>8550</td>
<td>8401</td>
<td>8550</td>
<td>255</td>
</tr>
</tbody>
</table>

NUMBER-OBSERVATIONS IN DATASET

In the example below a statement style macro is used to test an empty SAS data set and return a message or the number of observations in the data set.

The IMPLMAC and DQUOTE option must be turned on to use the macro in statement style form.

To use the macro enter:

NOBS datasetname;

OPTIONS NONPRINT NOMACROGEN NOSYMBOLGEN NOMLOGIC DQUOTE IMPLMAC;

%MACRO NOBS(dsn)/STMT;
%GLOBAL NOBS;
DATA _NULL_;
  I=1;
  IF _N_=1 AND NOBS=0 THEN DO;
    PUT "THE DATA SET &DSN IS EMPTY. PROCESSING WILL CONTINUE."
    PUT 'ERROR STATUS HAS NOT BEEN SET.';
    CALL SYMPUT('NOBS', '0');
    STOP;
  END;
  ELSE DO;
    SET &DSN POINT=I NOBS=NOBS;
    PUT "THE DATA SET &DSN CONTAINS " NOBS " OBSERVATIONS."
    CALL SYMPUT('NOBS', LEFT(PUT(NOBS, BEST.)));
    STOP;
  END;
RUN;
%MEND NOBS;
%NOBS
SYMGET EXAMPLES

SYMGET returns the value of a macro variable identified by the argument.

SYMGET(argument);
X = SYMGET(argument);

The argument (name of the macro variable) can have two forms:

1. the actual name of the macro variable enclosed in single quotes without an ampersand (&)
   X = SYMGET('STATE');

2. the name of a DATA step character variable whose value is the name of a macro variable.
   X = SYMGET(PLACE); where PLACE is a variable in the PDV with the value STATE.

MACRO SYSTEM OPTIONS: FUNCTION

<table>
<thead>
<tr>
<th>SAS SYSTEM OPTIONS</th>
<th>MACRO EFFECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACRO/NOMACRO</td>
<td>permit macro execution</td>
</tr>
<tr>
<td>MACROGEN/NOMACROGEN</td>
<td>print macro expansion</td>
</tr>
<tr>
<td>MCOMPILE/NONCOMPILE</td>
<td>load macro compiler by default</td>
</tr>
<tr>
<td>MERROR/NOMERROR</td>
<td>unmatched macro names</td>
</tr>
<tr>
<td>MLEAVE=</td>
<td>leave k unallocated memory</td>
</tr>
<tr>
<td>MLOGIC/NOMLOGIC</td>
<td>trace execution</td>
</tr>
<tr>
<td>MSIZE=</td>
<td>k of macro memory</td>
</tr>
<tr>
<td>MSymsize=</td>
<td>size of macro symbol table</td>
</tr>
<tr>
<td>MWORK=</td>
<td>macro work area size</td>
</tr>
<tr>
<td>SYMBOLGEN/NOSYMBOLGEN</td>
<td>print expansion of macro variable(s)</td>
</tr>
<tr>
<td>SERROR/NOSERROR</td>
<td>unknown macro variable</td>
</tr>
<tr>
<td>SYSParm='</td>
<td>pass text to &amp;SYSParm</td>
</tr>
</tbody>
</table>

NOTE: see the SAS USER'S GUIDE: BASICS, 1982 EDITION for complete documentation of system options and the OPTIONS statement.
WARNING MESSAGE OPTIONS

MERROR/NOMERROR

MERROR/NOMERROR causes a warning message to be issued for a name that looks like a macro call when no macro of that name has been defined. It is a good debugging aid for locating misspelled macro keywords and names.

SERROR/NOSERROR

SERROR/NOSERROR causes a warning message to be issued for names that look like macro variable references for which no symbolic variable can be found. This debugging tool helps find misspelled symbolic names or references to a symbolic variable before the variable is defined.

LOG INFORMATION OPTIONS

NOMACROGEN/MACROGEN

NOMACROGEN was designed for old macros to show the text substitution that was done when an old macro name was used. In SAS82, MACROGEN causes the constant text (the SAS statements before symbolic substitution) that a macro produces to be displayed or printed. It does not display the actual macro definition statements or the macro programming statements. The MPRINT option often provides a better way to see the results of macro execution.

NOMLOGIC/MLOGIC

MLOGIC directs the macro processor to trace its execution. All of the intermediate steps that macro execution performs are displayed on the log. MLOGIC should only be used as an in-depth debugging tool, since the resulting output is bulky and hard to read.

MPRINT/NOMPRINT

MPRINT causes the fully resolved result of a macro execution to be displayed on the SAS log. MPRINT allows you to see what the SAS compiler sees without any extraneous information. The MPRINT option overrides MACROGEN, SYMBOLGEN, and MLOGIC for macro-produced text.

NOSYMBOLGEN/SYMBOLGEN

SYMBOLGEN causes the results of symbolic resolution to be written to the SAS log. SYMBOLGEN should be used primarily as a debugging tool.
DEBUGGING OPTIONS

These SAS system options help in debugging macros:

<table>
<thead>
<tr>
<th>Option</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPRINT</td>
<td>the most useful option—shows the end result of macro processing (the statements as the SAS compiler sees them).</td>
</tr>
<tr>
<td>SYMBOLGEN</td>
<td>shows resolution of macro symbolic variables.</td>
</tr>
<tr>
<td>SOURCE2</td>
<td>displays lines from %INC as they are included, useful if making a macro library.</td>
</tr>
<tr>
<td>NOMPRINT</td>
<td>must be specified for MLOGIC to work.</td>
</tr>
<tr>
<td>MLOGIC</td>
<td>tells the macro executor to show internal resolution of macro variables and calls.</td>
</tr>
<tr>
<td>MACROGEN</td>
<td>specifies whether statements generated by macro execution are printed.</td>
</tr>
</tbody>
</table>

DEBUG MACRO

MACRO DEBUG(x) / STMT;
  %x* Macro from September, 1983
  %x*
  %x* DEBUG LEVELS:
  %x*     No - DEFAULT - NO SHOW;
  %x*     On - MPRINT SYMBOLGEN;
  %x*     All - MLOGIC ETC;
  %x*     ? - HELP;
  %if %x= %then %let x=no;
  %if %upcase(%substr(&x,1,1))=A %then %do;
    OPTIONS MLOGIC NOMPRINT SOURCE SYMBOLGEN MACROGEN SOURCE2;
    %end;
  %else
    %if %upcase(%substr(&x,1,1))=O %then %do;
      OPTIONS NOMLOGIC MPRINT SYMBOLGEN NOMACROGEN SOURCE2 SOURCE;
      %end;
    %else
      %if %upcase(%substr(&x,1,1))=N %then %do;
        OPTIONS NOMLOGIC NOSYMLOGIC NOSOURCE NONOTES NODATE;
        %end;
    %else %do;
      clear; run;
      %put %nrstr(
        %end; DEBUG is a STMT macro to set MACRO options.
      )
      %put %nrstr(
        To use it enter: DEBUG level, where level is:
      )
      %put %nrstr(
        No - Show Nothing (default%)
      )
      %put %nrstr(
        On - MPRINT SYMBOLGEN
      )
      %put %nrstr(
        All - MLOGIC ETC.
      )
      %put %nrstr(
        ? - HELP
      )
      %put %nrstr(
        First character is all you need
      )
      %end DEBUG;