More About SAS Macros

(Than You Thought Possible)

Donald P. Gallogly

DCBS IMD

Topics

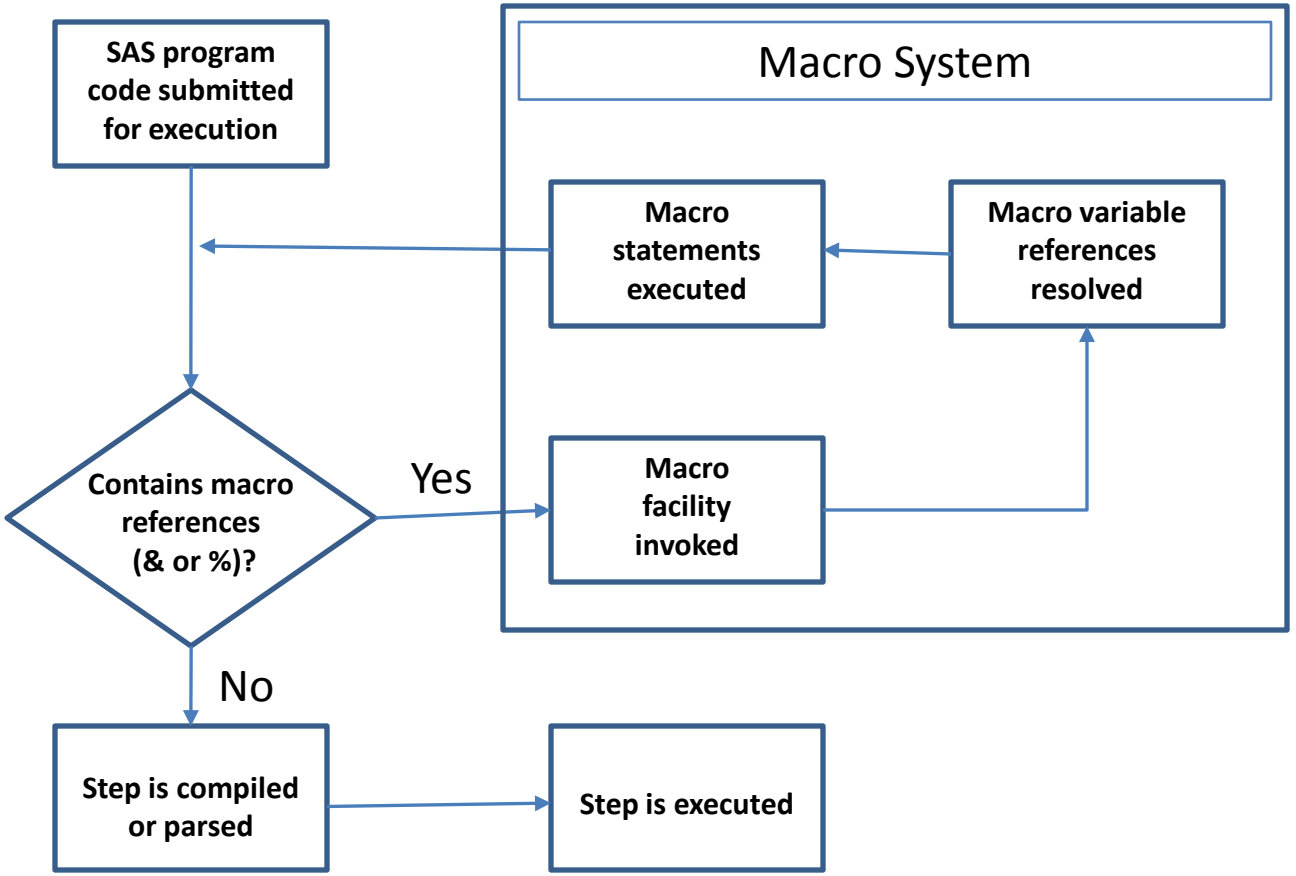
- The SAS Macros System
- Macro Variables
- Writing Macros

The SAS Macros System

The SAS Macros System

- SAS macros and macro variables are input to the SAS Macro System.
- SAS Macro System is a SAS-code generator.
- Output from the system is SAS code.

The SAS Macros System



The SAS Macros System

```
data &data (drop=lower_bound upper_bound
                %if &byvar = dropme %then dropme;);
```

The SAS Macros System

data basket (drop=lower_bound upper_bound);

or

data basket (drop=lower_bound upper_bound dropme);

MPRINT and MFILE

- MPRINT and MFILE are system options that allow you to see the SAS code that the macro system generates.
- These are very helpful when debugging macros.
  - MPRINT causes resolved SAS code to be written to the log.
  - MFILE causes SAS code to be written to an external file.

MPRINT and MFILE

```
options mprint;

filename mprint '/raprod/home/coolra/mycode.sas';
options mprint mfile;
```

The SAS Macros System

You can NEVER execute a macro statement from a data step conditional statement.

```
data _null_;
        set mydata;
        if myvar = 1 then %let mymacrovar = TRUE;
run;
```

The SAS Macros System

You can NEVER execute a macro statement from a data step conditional statement.

```
data _null_;
        set mydata;
        if myvar = 1 then
run;
```

The macro system executes the %let statement and leaves nothing behind.

SAS Macro Variables

Properties of SAS Macro Variables

- The value of the macro variable is TEXT.

- The macro processor will replace the macro reference with the value before executing macro statements or compiling SAS code.

Defining SAS Macro Variables

- Keyword `%let` followed by the variable name…
- Followed by an equal sign, some text and a semi-colon.
- Thus,
  - `%let YEAR = 2007;`

Referencing SAS Macro Variables

- To use (reference) a SAS macro variable, use the & followed by the variable name,
    - if &YEAR = 2007 then do;

SAS Macro Variables

- Not data set variables
  - Independent of a SAS data set
  - Contain a single value that remains constant unless explicitly changed
  - Can contain nothing (unlike data set variables)
  - Can contain up to 64k of text (about 12 pages)
  - Can’t contain control characters (‘, “, &, ;)

SAS Macro Variables

- %let nothing =;
- %let y = mx + b;
- %let service_cat = E&M;
- %let division = Workers' Compensation;
- %let debug = *;
- %let state = OR;

SAS Macro Variables

```
%let state = WA;

data _null_;
    set states_data;
    if &state = CA then do;
        …
    end;
run;
```

```
%let state = OR;

data _null_;
    set states_data;
    if &state = CA then do;
        …
    end;
run;
```

SAS Macro Variables

User-Defined Macro Variables

- Defined by the user
- Replaced by their contents at compile-time
- Contain data of special interest to the user

Automatic Macro Variables

- Defined automatically
- Also replaced by contents at compile-time
- Contain data about the state of the system

Useful Automatic Macro Variables

- &sysdate — date that session began (date7.)
- &sysdate9 — date that session began (date9.)
- &syslast — name of the last data set modified (lib.name)
- &sysdsn — name of the last data set modified (lib name)
- &sysuserid — ID of the user who submitted the job
- &sysmacroname — contains the name of the macro that is currently running

symputx

- Data step function to store values into a macro variable.
- call symputx(macro_var, value)

```
data _null_;
   set mydata;
   if myvar = 1 then call symputx(macro_var  ,TRUE);
run;
```

symget

- Data step function that converts macro variable values to character strings that can be assigned to data set variables.
- Use it when macro variable name needs to be resolved during data step execution.

```
data _null_;
      set mydata;
      myvar = &mymacro_var;
run;
```

symget

- Data step function that converts macro variable values to character strings that can be assigned to data set variables.
- Use it when macro variable name needs to be resolved during data step execution.

```
data _null_;
      set mydata;
      if servcat = “surg” then charge = rvu * &cfsurg;
      else if servcat = “EandM” then charge = rvu * &cfEandM;
      else if servcat = “pharm” then charge = rvu * &cfpharm;
      else if servcat = “radio” then charge = rvu * &cfradio;
run;
```

symget

- Data step function that converts macro variable values to character strings that can be assigned to data set variables.
- Used when macro variable name needs to be resolved during data step execution.

```
data _null_;
      set mydata;
      charge = rvu * symget('cf'||servcat);
run;
```

SAS Macros

SAS Macros

- Allow you to write dynamic code.
- Allow you to conditionally execute proc and data steps.
- Allow you to create libraries of useful reusable functions.

SAS Macros by Code Substitution

%Macro *OpenSSXP;*

```
filename tab1 "/raprod/wcd/projects/temp&SYSUSERID..xls";

ods tagsets.ExcelXP options(embedded_titles='yes’ suppress_bylines='yes')
    file=tab1;
```

%Mend OpenSSXP;

Passing Parameters

- **Positional Parameters: must be listed in order.**

  %macro ApplyPharmMult(PharmData, Year);

  %*ApplyPharmMult*(PhDat, 2006);

- **Keyword Parameters: must be referred to by keyword.**

  %macro TrimOutliers(data=, byvar=dropme, datavar=);

  %*TrimOutliers (byvar=service_code, data=basket, datavar=payment);*

Passing Parameters

- **Refer to parameters within the macro as you would any macro variables.**

```
%macro TrimOutliers(data=, byvar=dropme, datavar=);
  %if &byvar eq dropme %then %do;
        data &data;
              set &data;
              dropme = 1;
        run;
  %end;
%mend TrimOutliers;
```

Conditional Macro Statements

- **Conditional statements make decisions based on current conditions.**

```
%macro TrimOutliers(data=, byvar=dropme, datavar=);
   %if &byvar eq dropme %then %do;
            data &data;
                     set &data;
                     dropme = 1;
            run;
   %end;
%mend TrimOutliers;
```

%Do blocks

- **%Do blocks allow you to repeat sections of code repeatedly.**

```
data thresholds (keep=&byvar upper_bound lower_bound);
    set percentiles;
    %do i=90 %to 99;
      if P%eval(&i+1)/P&i gt 1.5 then upper_bound = P&i*1.2;
      else
    %end;
    upper_bound = P&i;
run;
```

%Do blocks

- **%Do blocks allow you to repeat sections of code repeatedly.**

```
data thresholds (keep=servcat upper_bound lower_bound);
    set percentiles;
    if P91/P90 gt 1.5 then upper_bound = P90*1.2;
    else if P92/P91 gt 1.5 then upper_bound = P91*1.2;
    else if P93/P92 gt 1.5 then upper_bound = P92*1.2;
    else if P94/P93 gt 1.5 then upper_bound = P93*1.2;
    else if P95/P94 gt 1.5 then upper_bound = P94*1.2;
    else if P96/P95 gt 1.5 then upper_bound = P95*1.2;
    else if P97/P96 gt 1.5 then upper_bound = P96*1.2;
    else if P98/P97 gt 1.5 then upper_bound = P97*1.2;
    else if P99/P98 gt 1.5 then upper_bound = P98*1.2;
    else if P100/P99 gt 1.5 then upper_bound = P99*1.2;
    else upper_bound = P100;
run;
```

Writing to the log

- **%Put writes to the log from the macro processor just as Put does at run-time.**
  - **%put &sysdsn displays the name of last modified dataset.**

Writing to the log

- **%Put writes to the log from the macro processor just as Put does at run-time.**
  - **%put &sysdsn displays the name of last modified dataset.**
  - **%put ERROR: File not found.**
  - **%put WARNING: Data out of range.**
  - **%put NOTE: Have a nice day.**

Text functions

• %index(arg1, arg2);	Returns the position of arg2 in arg1.
• %length(arg);	Returns the length of arg.
• %scan & %qscan(arg, n);	Returns the nth word in arg1.
• %substr & %qsubstr(arg, pos, <len>);	Returns a string of length len, starting at position pos, from arg.
• %upcase & %qupcase(arg);	Returns arg in ALL CAPS.

Use the Q(uoted) functions when arg might include quotes, ampersands, percent signs, etc.

Mathematical Functions

- %eval and %sysevalf do arithmetic using macro variables and return the result.
  - %eval only does integer arithmetic.
  - %sysevalf only does floating point arithmetic.

Mathematical Functions

```
%macro line;
    %let m = 0.5;
    %let b = 2;
    %do x = 0 to 100;
        %let y = &m * &x + &b;
        %put y = &y;
    %end;
%mend line;
```

Mathematical Functions

$y = 0.5 * 0 + 2$

$y = 0.5 * 1 + 2$

$y = 0.5 * 2 + 2$

$y = 0.5 * 3 + 2$

$y = 0.5 * 4 + 2$

$y = 0.5 * 5 + 2$

$y = 0.5 * 6 + 2$

$y = 0.5 * 7 + 2$

…

$y = 0.5 * 100 + 2$

Mathematical Functions

```
%macro line;
    %let m = 0.5;
    %let b = 2;
    %do x = 0 to 100;
        %let y = %sysevalf(&m*&x + &b);
        %put y = &y;
    %end;
%mend line;
```

Mathematical Functions

y = 2.0

y = 2.5

y = 3.0

y = 3.5

y = 4.0

y = 4.5

y = 5.0

y = 5.5

…

y = 52.0

Using data step functions and CALL routines

- %syscall and %sysfunc can be used to call routines and functions that are normally only available in the data step.

Using CALL Routines

- %syscall executes CALL routines.

```
%let seed = 9876;
%let random_num = ;
%syscall ranuni(seed, random_num);
%put Your pseudorandom number is &random_num;
```

Your pseudorandom number is 0.9876783463462

Using data step functions

- %sysfunc executes data step functions.

```
%if %sysfunc(abs(payment)) > %sysfunc(abs(charge)) %then %do;

%put Today is &sysfunc(date(), date7.);
```

Thank you for coming

- Q & A Time