Club des utilisateurs SAS de Québec

Demystifying the SAS® Macro Facility – by Example

Harry Droogendyk
Stratia Consulting

Marje Fecht
Prowerk Consulting

Introduction

The SAS macro facility enables you to apply a wealth of useful, uncomplicated, real-world solutions to enhance your coding pleasure, reduce coding effort, and minimize error.

Hopefully, this presentation will remove some of the mystery of macros and provide you with tips and tricks you can take away and implement immediately.

 2

Introduction

This presentation focuses on using the macro facility to

- reduce code repetition
- increase control over program execution
- minimize manual intervention
- create modular code.

Until you understand the “inner workings” of the macro facility, it *can* be mysterious. We’ll share some common *Gotchas* to help you avoid programming and macro traps.

 3

GOTCHA - #1 Tedious Repetition and Maintenance

Changes required each month for new datasets:

```
title "Sales 200511";
proc print data=sales_200511 noobs; run;

title "Sales 200512";
proc print data=sales_200512 noobs; run;

title "Sales 200601";
proc print data=sales_200601 noobs; run;

title "Sales 200602";
proc print data=sales_200602 noobs; run;
```

 4

GOTCHA - #2 – What is wrong with IF?

```
data gotcha;
  set DailyTxns;
  if txnSize > 50000 then do;
    %let alert = NOTE: BIG Transactions;
  end;
run;
proc means data=gotcha mean min max;
  var TxnSize;
  title "Average Txns  &alert";
run;
```

```
Average Txns  NOTE: BIG Transactions

        Mean          Minimum          Maximum
-------------------------------------------
    23941.62        20000.00         49685.48
-------------------------------------------
```

 5

Macro Variables – What *IS* the timing?!

Compiler

Word Scanner

Execution Stack

```
data gotcha;
set DailyTxns;
if txnSize>50000 then do;
```

```
data gotcha;
  set DailyTxns;
  if txnSize>50000 then do;
   %let alert = NOTE: BIG Transactions;
  end;
run;
proc means data=gotcha mean min max;
  var TxnSize;
  title "Average Txns  &alert";
run;
```

Macro Processor

Symbol Table

 6

Macro Variables – What *IS* the timing?!

Compiler | Word Scanner | Macro Processor

Execution Stack

```
data gotcha;
set DailyTxns;
if txnSize>50000 then do;
```

Symbol Table

```
alert
 NOTE: BIG
   Transactions
```

```
data gotcha;
  set DailyTxns;
  if txnSize>50000 then do;
   %let alert = NOTE: BIG Transactions;
  end;
run;
proc means data=gotcha mean min max;
  var TxnSize;
  title "Average Txns  &alert";
run;
```

 7

Macro Variables – What *IS* the timing?!

Compiler | Word Scanner | Macro Processor

Execution Stack

```
data gotcha;
set DailyTxns;
if txnSize>50000 then do;
end;
run;
```

Symbol Table

```
alert
 NOTE: BIG
   Transactions
```

```
data gotcha;
  set DailyTxns;
  if txnSize>50000 then do;
   %let alert = NOTE: BIG Transactions;
  end;
run;
proc means data=gotcha mean min max;
  var TxnSize;
  title "Average Txns  &alert";
run;
```

 8

Macro Variables – What *IS* the timing?!

Compiler | Word Scanner | Macro Processor

Execution Stack

```
proc means data=gotcha
  mean min max;
  var TxnSize;
  title "Average Txns
```

Symbol Table

```
alert
 NOTE: BIG
   Transactions
```

```
data gotcha;
  set DailyTxns;
  if txnSize>50000 then do;
   %let alert = NOTE: BIG Transactions;
  end;
run;
proc means data=gotcha mean min max;
  var TxnSize;
  title "Average Txns  &alert";
run;
```

 9

Macro Variables – What *IS* the timing?!

- When macro code is encountered by the data step compiler it is processed immediately
  - i.e. macro code runs before the data step code.
- Mixing macro and data step code can generate unexpected results!

 10

Decisions – Creating Macro Variables

- Who processes the %let ?
- When is the %let processed ?
- Macro processor deals with %let
  - doesn't recognize as related to the data step
  - the %let never makes it to the *data step* compiler
- Timing demands a different approach such as a SYMPUT to create the macro variable

 11

SAS Macro Facility

The SAS Macro facility includes:

- Macro variables
  - Automatic (system defined)
  - User defined
- Macro functions
- Macro definition / creation
- Macro programming statements
  - conditional, iterative processing
  - environment control.

 12

Macro Variables

- Macro variables
  - contain text
  - are typically used for text substitution.
- Automatic (system-defined) macro variables:
  - &sysdate and &sysdate9
  - &sysjobid
  - &syserr
- User-defined macro variables:
  - %let mode = PROD;
  - DATA step: call symputx('nobs' , N);
  - PROC SQL: select into :mvar …

 13

Macro Variables – Resolving / Viewing

- To view the contents of Macro variables
  - %put Mode is &mode;
  - %put _user_;
  - %put _automatic_;
- Macro variables are "typically" resolved *prior to* SAS code compilation.
  - Title "Report run on &RunDate";
  - %let type = temp;
  - SET lib.mydata &type.data;

 14

Macro Variables – Not just a "word"

Macro variables contain text - any text

```
%let code = %str(data a; i = 7; run;);
options symbolgen;
&code;

====== LOG ========
SYMBOLGEN:  Macro variable CODE resolves to
 data a; i = 7; run;
NOTE: The data set WORK.A has 1 observations and 1
   variables.
```

 15

Removing the "meaning" of text

Why was %str( ) used?

```
%let code = %str(data a;i = 7;run;);
```

What is the value of &code below?

```
%let code = data a;i = 7;run;;
```

 16

Why use Macro Variables

- Easily Maintained Code
  - Change the value of a macro variable once and the new value substitutes wherever &mode appears
    - don't *care* where those values are in program
- "Table-Driven" Code
  - monthly proc prints from Gotcha # 1
  - use data to dynamically define macro variables
  - macro variables act as parameters to program
  - program adapts / reacts to the data
- Controlling Program Execution
  - should this line of code / step execute ?

 17

Macro Variables – Custom Date Values

Custom date values are useful for:

- Report titles *Reporting as of March 27, 2006*
- "Versioned" file names
  - SAS log *CpmgnA_20060327_1329.log*
  - Reports *CmpgnA_20060327.xls*
- Subset criteria – especially with SQL pass-thru

In the "old days", assigning custom date values to macro variables was ONLY accomplished via a data step. Now there are more elegant solutions.

 18

Macro Variables – Custom Date Values
Classic Approach

```
data _null_;
  call symput('d',put(today(),yymmddN8.));
run;
title "Report as of &d";
```

Report as of 20060327

 19

Macro Variables – Custom Date Values
Streamlined Approach

In one line, using %sysfunc!

```
%let Date = %sysfunc(today(),yymmddN8.);
title "Report As of &Date";
%let log = CpmgnA_&Date..log;
```

Or, without creating an intermediate variable:

```
title "As of %sysfunc(today(),yymmddN8.)";
```

Watch spaces!

```
%let num = %sysfunc(today(),8.);
%put **&num**; **  16887**
```

 20

Macro Variables – Controlling Execution

Consider using macro variables to control

- storage locations for logs, data, and output
- sampling during testing
- effectively halting program (or single step) execution when errors detected.

```
%let run = run;
.. step ..
%if &syserr %then %let run = %str(run cancel);
data _null_;
    etc…
&run;
```

 21

Switching between TEST and PROD

To enable sampling in TEST, set sample to null or %str();

To avoid sampling in PROD, set sample to an asterisk

```
%let mode = PROD;
%let sample = *; /*specify an * to avoid sampling*/
%let yyyymm = 200510;
libname cc "H:\sugi31\&mode\&yyyymm\Cmpgn";
data cc;
  set cc.campaigns;
  &sample if ranuni(1) * 10 > 9.9;
  where LOB = '610';
run;
```

 22

Macro Variables – Recap

- Macro variables contain text
- Macro variables are useful for substituting values into SAS code
- Use of macro variables enables efficiency
  - macro variable date stamp creates unique names
  - dynamic titles
  - promoting from TEST to PROD is a snap!

 23

Macro Variables – creating an "IN" list

Dynamic "in" lists help make low-maintenance programs.

- `where dept in (&DeptList);`
- `where me_dt in (&MonthList);`

The data step can create the list, but SQL takes less "thinking".

 24

Macro Variables – creating an "IN" list

SQL Solution:

```
proc sql noprint;
 select distinct quote(trim(region))
     into :reglist separated by ","
     from sashelp.shoes
   where returns > 10000;
quit;
%put &reglist;
```

```
====== Partial Log ========
"Africa","Canada","Central
  America/Caribbean","Middle East","Pacific","United
  States","Western Europe"
```

Tip: Use translate() to change double quotes to single

 25

Macro Variables – using an "IN" list

```
proc sql;
  select region
       , sum(returns) as tot_returns
    from sashelp.shoes
   where region in ( &reglist )
   group by region;
quit;
```

```
====== Partial Output ========
Region                      tot_returns
---------------------------------------
Africa                            74087
Canada                           129394
```

 26

Macro Variables – creating an "IN" list

The code works to produce the IN list needed for *this* application, but . . .
what if you want to re-use the code for
- other variables and different data sources ?
- both numeric and character values ?

A macro provides flexibility to:
- specify input parameters
  - specify data set and variable names
- allow decision making
- create reusable code.

 27

Creating a macro

- Starts with **%macro** *macro_name*;
- Ends with **%mend** *macro_name*;
- Virtually anything you want inside!

```
%macro sug_macro;
     %put SAS rocks !!;
%mend sug_macro;
%sug_macro;
====== LOG =======
SAS rocks !!
```

 28

Why create a macro?

- Macro expressions for looping and decisioning can *only* take place inside a macro
  - loops %do / %end
  - decisioning %if / %then / %else
- Modular code
  - macro "functions"
- Parameter-driven code
- Job security ;-)

 29

Creating an "IN" list – Macro-ized

Change previous example into a macro

```
%macro inlist;
%global reglist;
  proc sql noprint;
     select distinct quote(trim(region))
       into :reglist separated by ','
       from sashelp.shoes;
  quit;
%mend inlist;
%inlist;
```

Is this better? No, but *NOW* let's add some value!

 30

Creating an "IN" list – Macro Parameters

Parameters add flexibility - *more* useful for *more* applications.

```
%macro inlist(ds=,fld=,mvar=inlist);
  %global &mvar; *make var available outside macro;
  proc sql noprint;
      select distinct quote(trim(&fld))
        into :&mvar separated by ','
        from &ds;
  quit;
%mend inlist;
%inlist(ds=sashelp.shoes,mvar=reglist,fld=region);
```

Specification of parameters allows for "dynamic" code.

Creating an "IN" list – Macro Parameters

While we've added the ability to specify parameters to "drive" the code, we're still somewhat limited.

- What if we want a list of numeric values? Should they be quoted?
- How can the macro "decide" if quotes are required?
- We're going to add another macro parameter… but we're not *always* going to use it. More mystery. ☺

Let's look at macro "decisioning" which will help us deal with numeric vs. character values.

Creating an "IN" list – Decisioning

```
%macro inlist(ds=,fld=,type=C,mvar=inlist);
   %global &mvar;
   proc sql noprint;
        %if &type = C %then %do;
             select distinct quote(trim(&fld))
        %end; %else %do;
             select distinct &fld
        %end;
        into :&mvar separated by ',' from &ds;
   quit;
%mend inlist;
%inlist(ds=sashelp.shoes,mvar=reglist,fld=region);
```

Where's type= ? It defaults to value in macro ***definition !***

Creating a SEQUENCE ("array") of Macro Variables

In the previous example, all values were stored in a single macro variable. However, a sequence of macro variables can be handy for iterative processing.

For example, you may need to produce monthly data for a "dynamically determined" number of months.

- `%let Month1 = 198001;`
- `%let Month2 = 198002;`

Think of "Gotcha1", PROC PRINTS of multiple "month" datasets

Creating a sequence ("array") of macro variables is easy!

Creating a SEQUENCE / "array" of Macro Variables

```
proc sql noprint;
select distinct put(mdy(month,day,year), yymmN6.)
  into :Mth1 - :Mth999
  from sashelp.retail;
%let NumMths = &sqlobs;
quit;
%put &numMths Months of Dates;
%put Mth 1: &Mth1, Mth &NumMths: &&Mth&NumMths;

====== Partial Log ========
%put &numMths Months of Dates;
58 Months of Dates
%put Mth 1: &Mth1, Mth &NumMths: &&Mth&NumMths;
Mth 1: 198001, Mth 58: 199404
```


Macro variable r-r-resolution

- Discussed word scanner and macro processor earlier
  - as macro triggers are encountered, macro processor kicks in
  - finds an ampersand, searches the symbol table for variable
  - substitutes value.

`%put &&Mth&NumMths;`

- Multiple passes to resolve multiple ampersands
  - `&&Mth&NumMths`
    - `&&Mth` resolves to `&Mth`
    - `&NumMths` resolves to 58
      - result is `&Mth58`
  - `&Mth58` resolves to `199404`

Using a SEQUENCE / "array" of Macro Variables

Remember "Gotcha 1" ?

```
%macro print(lmt);
  %do i = 1 %to &lmt;
     title "Sales &&Mth&i";
     proc print data=sales_&&Mth&i noobs;
     run;
  %end;
%mend print;

%print(&NumMths)
```

 37

Using a SEQUENCE / "array" of Macro Variables

options mprint; reveals the code generated

```
====== Partial Log ========
MLOGIC(PRINT):  Beginning execution.
MLOGIC(PRINT):  Parameter LMT has value 3
MLOGIC(PRINT):  %DO loop beginning; index
  variable I; start value is 1; stop value is 3;
  by value is 1.
MPRINT(PRINT):   title "Sales 198001";
MPRINT(PRINT):   proc print data=sales_198001
  noobs;
MPRINT(PRINT):   run;
      <snip>
MLOGIC(PRINT):  %DO loop index variable I is now
  4; loop will not iterate again.
MLOGIC(PRINT):  Ending execution.
```

 38

Real-World Application of Macro "functions"

- Daily process to merge Master dataset with daily Transaction dataset
  - macro **%do_merge** performs merge.
- When Transaction dataset doesn't exist or is empty we want to skip the merge
  - exist() function
  - number of obs is available
    - nobs= on SET statement
    - sashelp.vtable nobs value
    - attrn() function.

 39

Olde Methodology

Could use a data step to define macro variables:

```
data _null_;
  call symput('exist',exist('transaction'));
run;


data _null_;
  if 0 then set transaction nobs=nobs;
  call symput('nobs',nobs);
  stop;
run;
```

 40

More Recent Methodology

Or use SQL and SAS metadata:

```
%let nobs = 0;   ** Need to predefine NOBS in
         case transaction DS does not exist ;
proc sql;
  select nobs into :nobs
    from sashelp.vtable
   where libname = 'YOURLIB'
     and memname = 'TRANSACTION';
  %let exist = &sqlobs;
quit;

%put EXIST=&exist NOBS=&nobs;
```

 41

Magnificent Macro Methodology

```
  ***** utility macro to provide ds attributes;
%macro attrn(ds,attrib);
  %let dsid = %sysfunc(open(&ds,is));
  %sysfunc(attrn(&dsid,&attrib))
  %let rc = %sysfunc(close(&dsid));
%mend attrn;

%macro check_transaction;
  %if %sysfunc(exist(transaction)) %then %do;
     %if %attrn(transaction,nobs)> 0 %then %do;
          %do_merge
     %end; %else %put No obs in TRANSACTION;
  %end; %else %put TRANSACTION does NOT exist;
%mend check_transaction;
%check_transaction;
```

 42

Macro Miscellanies

- SAS/Connect
  - %sysrput – copy remote macro variables to local
  - %syslput – copy local macro variables to remote
- "cleaning up" macro variables
  - v8 – set value to nothing

```
%let mvar = ;
```

  - v9 – actually delete macro variable

```
%symdel mvar;
```

- determining macro variable existence
  - v9 - %symexist, %symglobl
- SASAUTOs

Conclusion

- SAS macro facility
  - made up of macro variables, macro definitions, macro "functions"
- Reduces code repetition
- Minimizes program maintenance
- Increases control over program execution
- Creates modular, reusable code

Club des utilisateurs SAS de Québec

Thank you

Prowerk™

Marje Fecht
Prowerk Consulting
marje.fecht@prowerk.com