

SAS® Macros – Bits and Pieces

Lawrence Heaton-Wright, Quintiles



Introduction

- A look through some useful statements, functions, options and variables that can be used as part of a programmers toolkit
- Statements
 - › %GOTO and %LABEL, %RETURN
- Functions
 - › %UPCASE, %EVAL and %SYSEVALF, %SYSFUNC
- Options
 - › Debugging and log printing options, MINOPERATOR
- Data step/PROC SQL interactions
 - › CALL SYMPUTX, CALL EXECUTE, PROC SQL INTO CLAUSE
- Closing Remarks

%GOTO and %LABEL

- Statements are paired together in macro code
- It does exactly what it says
 - › It goes to (jumps) to the point in the program defined by the label
 - › Note that the label is defined in the code as %LABEL but referenced in the %GOTO without the %
- If the GOTO condition is reached the macro processor does not execute any code between the %GOTO and the %LABEL
- You can achieve the same result using a series of %IF ... %THEN statements
 - › It can be hard keeping track of these
 - › %GOTO is easier to understand
 - › To some level it's self-documenting
- Used primarily for the early exit from a macro based on error-trapping
 - › You can customise the error message for the user – which they may ignore!

%GOTO and %LABEL

%MACRO macroname;

<macro code section 1>

%IF exit condition 1 %THEN %GOTO exit;

<macro code section 2>

%IF exit condition 2 %THEN %GOTO exit;

<macro code section 3>

%IF exit condition 3 %THEN %GOTO exit;

<macro code section 4>

%exit:

<exit macro code>

%MEND macroname;

%GOTO and %LABEL

%MACRO macroname;

<macro code section 1>

%IF exit condition 1 %THEN %GOTO exit;

<macro code section 2>

%IF exit condition 2 %THEN %GOTO exit;

<macro code section 3>

%IF exit condition 3 %THEN %GOTO exit;

<macro code section 4>

%exit:

<exit macro code>

%MEND macroname;

%RETURN

- Causes the normal termination of the currently executing macro
- Used when %GOTO/%LABEL combination cannot be not used
 - › Trying to exit out of a sub-macro to a label in the parent macro
- Very useful option when you just want to stop the macro executing and we're not interested in performing any clean up functionality

%MACRO *return_macro*;

<macro code>

%IF exit condition %THEN %RETURN;

<more macro code>

%MEND *return_macro*;

- If the exit condition is met then the RETURN statement terminates the macro

%UPCASE

- Converts macro variable values to uppercase
- This is incredibly useful when you're checking the values of macro variables in an IF statement (such as data checking).
 - › For instance I might have a macro variable which I intend to only be Y or N
 - › The user could enter Y, y, N or n
 - › Using %UPCASE can immediately reduce the combinations to check in half

`%LET ynVar=%UPCASE(&ynVar);`

%UPCASE

- Combining this with the IN operator ensures that only the intended values are entered in the macro variable:

```
%IF &ynVar IN (N Y) %THEN %DO; <more macro code> %END;
```

- If the variable was not upper-cased then the SAS code would be:

```
%IF &ynVar IN (N n Y y) %THEN %DO; <more macro code> %END;
```

- If there are more options or longer text strings then it could get quite congested within the IN statement.
- You can also use this function as part of a statement without changing the macro variable:

```
%IF %UPCASE(&SysProcessName) NE DMS PROCESS %THEN %DO;
```

- This means that I am typing less text in my macro and giving the program less chance to fall over through spelling mistakes or typos.

%EVAL and %SYSEVALF

- All macro variables are defined as character variables
- To perform some form of mathematical operation on macro variables these 2 functions allow us to evaluate macro variables as numeric values rather than character values.
- %EVAL allows you to evaluate integer expressions as numeric values.

`%IF %EVAL(&x) LT 15 %THEN %PUT INTEGER LESS THAN 15;`

`%ELSE %IF %EVAL(&x) GE 15 %THEN %PUT INTEGER 15 OR ABOVE;`

%EVAL and %SYSEVALF

- %SYSEVALF allows you to evaluate non-integer expressions.

```
%LET x=15; %LET Y=24.2;
```

```
%LET z=%EVAL(&x+&y);
```

In this example the SAS LOG file results in the following message:

ERROR: A character operand was found in the %EVAL function or %IF condition where a numeric operand is required. The condition was: 15+24.2

However, if we use SYSEVALF like:

```
%LET z=%SYSEVALF(&x+&y);
```

We get the following:

```
x=15 y=24.2 z=39.2
```

%SYSFUNC

- This macro function allows you to utilise data step functions within macro code
 - › There are some functions you are not allowed to use
- One of the main uses I've found for %SYSFUNC is to utilise functions such as OPEN, CLOSE, ATTRN, ATTRC to perform tasks such as identifying the number of records in a data set or whether a variable exists within a data set.
- To obtain the number of observations in a data set, this code is extremely efficient as no extra data sets or procedures are used.

```
%LET dsid=%SYSFUNC(OPEN(WORK.DATASET));
```

```
%LET nobs=%SYSFUNC(ATTRN(&dsid.,NOBS));
```

```
%LET rc=%SYSFUNC(CLOSE(&dsid.));
```

- Another use for %SYSFUNC is to enable the use of the GETOPTION function to retrieve the setting of a SAS system option into a macro variable
 - › This is especially useful if you want to turn off certain options (SOURCE, SOURCE2, MPRINT, MLOGIC, etc.) during the course of your macro but then to reset them once your macro has completed execution

Debugging and log printing options

- There are many options to aid the programmer in de-bugging macros
- When developing and de-bugging programs it is worth having some (or all) of these options turned on
- Once into production it is worthwhile turning these options off as they generate tremendous amounts of data in the LOG file

Debugging and log printing options

- MPRINT displays the SAS statements that are generated by the macro execution.
- MPRINTNEST prints nesting information to be displayed in the MPRINT log output.
 - › This is useful when macros are called from within macros, etc. so that you can trace what macro is currently executing and also what macro (if any) it was invoked from.
- MLOGIC shows the trace of the macros execution. This generates a LOT of output but can be really useful in identifying logical issues within a macro.
- MLOGICNEST does the same thing as MPRINTNEST but for MLOGIC.
- SYMBOLGEN displays the results of resolving macro variables.

Debugging and log printing options

- SOURCE/SOURCE2 – these options are not macro LOG options.
- They apply to data step/procedure code (SOURCE2 is from included files).
- If these are turned off then the conventional LOG messages showing DATA step/PROC resolution if turned off.
- This can be very useful when it comes to macro suites where you want to control messages printed to the LOG.
- Top tip: use a keyboard macro file to add these options or turn them off.

MINOPERATOR

- This can be used either in an OPTIONS statement or as an option to the %MACRO statement
 - › OPTIONS MINOPERATOR;
 - › %MACRO macroname / MINOPERATOR;
- This option allows the use of the macro IN operator in an expression (similar to the DATA step IN operator).
- Prior to version 9 if we wanted to check whether a macro variable contained several values then the %IF statement would need to be constructed like:

```
%IF &Var. EQ A OR &Var. EQ B OR &Var. EQ C %THEN %DO;
```
- With the MINOPERATOR option this code can be changed to

```
%IF &Var. IN (A B C) %THEN %DO;
```
- This code is much easier to understand, it's more elegant and maintain such code as it's familiar to the well-known DATA step code.

CALL SYMPUTX

- The CALL SYMPUT routine has been used for years to create macro variables within a DATA step.
- In version 9 CALL SYMPUTX was introduced which introduced many upgrades.
 - › Macro variable can be defined as local or global in scope.
 - › If we need to create global macro variables within a sub-macro we now don't need to use a %GLOBAL statement.
- Numeric to character conversion is automatically applied resulting in less code (no need for COMPRESS and PUT functions).
- The resulting character variable automatically has leading and trailing spaces removed.
- This routine was introduced in v9 and is much, much better than plain old CALL SYMPUT.

CALL EXECUTE

- This brilliant data step call routine allows you to execute macros from inside a data step.
- The macro that is called can have multiple procedures and data steps which makes this an extremely powerful routine.
- One thing to be aware of:
 - › You cannot create macro variables during the execution of the macro called by CALL EXECUTE.
 - › This means that if your macro has %LET or CALL SYMPUT(X) or PROC SQL INTO: macro variable creation statements then the program will fall over due to this restriction.
- Quite often I use a CALL EXECUTE macro call when I need to iterate through several steps utilising different parameters/subjects.

CALL EXECUTE

```
%MACRO ExampleExecute (usubjid=);  
  DATA _selected_data;  
    SET adam.advs (WHERE=(usubjid EQ "&usubjid."));  
  RUN;  
  **** MORE DATA STEPS/PROCEDURES ***;  
%MEND ExampleExecute;
```

```
DATA _NULL_;  
  SET adam.adsl (WHERE=(saffl eq 'Y'));  
  CALL EXECUTE (CATT('%ExampleExecute (usubjid=',usubjid,');'));  
RUN;
```

This builds the macro call for each subject within the input data set
i.e. %ExampleExecute (usubjid='ABC123-0001');
For each subject within the input data set the macro executes for that
subject
Note that the macro executed can have data steps and procedures

CALL EXECUTE

- This brilliant data step call routine allows you to execute macros from inside a data step.
- The macro that is called can have multiple procedures and data steps which makes this an extremely powerful routine.
- One thing to be aware of:
 - › You cannot create macro variables during the execution of the macro called by CALL EXECUTE.
 - › This means that if your macro has %LET or CALL SYMPUT(X) or PROC SQL INTO: macro variable creation statements then the program will fall over due to this restriction.
- Quite often I use a CALL EXECUTE macro call when I need to iterate through several steps utilising different parameters/subjects.
- If macro variables need to be created during the course of the macro then I'd use a different strategy

PROC SQL INTO CLAUSE

- This is an alternative method of creating macro variables to using a DATA step and CALL SYMPUTX.
- Where this method really scores highly is the ability to produce a combined list of values in one macro variable.
- Can be used to identify a list of subject numbers for investigation and to quickly print out the data for these subjects without having to subset the data using merges (or other methods).

PROC SQL INTO CLAUSE

```
PROC SQL NOPRINT;
```

```
  SELECT DISTINCT(QUOTE(usubjid)) INTO: subjects SEPARATED BY ','
```

```
  FROM adsl WHERE trtsdt LT "01APR2014"D;
```

```
QUIT;
```

- This will produce a macro variable which contains a list of subjects that satisfy the WHERE clause:

```
"ABC101001","ABC102004","ABC104001"
```

- The macro variable produced can then be used like:

```
PROC PRINT DATA=dataset WIDTH=MIN HEADING=H;
```

```
  WHERE usubjid IN (&subjects.);
```

```
RUN;
```

PROC SQL INTO CLAUSE

ensures that the selected data is not displayed the output destination

statement ensures that only unique values are selected

ensures that the selected values of the variable are quoted

send the results to the specified macro variable

```
PROC SQL NOPRINT;  
SELECT DISTINCT(QUOTE(usubjid)) INTO: subjects SEPARATED BY ','  
FROM adsl WHERE trtsdt LT "01APR2014"D;  
QUIT;
```

sends all the results into one macro variable separated by the specified character

source data

determine what data to select

Closing Remarks

- There is no specific conclusion to take away from this paper
- However, I hope to have shown how wide-ranging and versatile SAS macro programming is and that there will be enough hints and tips for you to take away from this paper and experiment with and expand your macro programming knowledge.
- There is more in the actual paper

Your comments and questions are valued and encouraged.

Contact the author at:

Lawrence Heaton-Wright

Quintiles Limited

500 Brook Drive,

Green Park,

Reading,

Berkshire.

RG2 6UU

Phone: +44 118 4508320

Email: lawrence.heaton-wright@quintiles.com

Web: www.quintiles.com