MWSUG 2017 - Paper BB15

Building Intelligent Macros: Driving a Variable Parameter System with Metadata

Arthur L. Carpenter, California Occidental Consultants, Anchorage, Alaska

ABSTRACT

When faced with generating a series of reports, graphs, and charts; we will often use the macro language to simplify the process. Commonly we will write a series of generalized macros, each with the capability of creating a variety of outputs that depend the macro parameter inputs. For large projects, potentially with hundreds of outputs, controlling the macro calls can itself become difficult.

The use of control files (metadata) to organize and process when a single macro is to be executed multiple times was discussed by Rosenbloom and Carpenter (2015). But those techniques only partially help us when multiple macros, each with its own set of parameters, are to be called. This paper discusses a technique that allows you to control the order of macro calls along with each macro's parameter set, while using a metadata control file.

KEYWORDS

Metadata, macro language, CALL EXECUTE, %NRSTR, %TSLIT

INTRODUCTION

When a single macro is to be called multiple times with varying parameters, a metadata control file such as the one discussed in detail in Rosenbloom and Carpenter (2015) can be used. Typically this metadata file will contain one row per macro call and one column per parameter. Depending on the ultimate objective of the programmer, the metadata control file can be used to generate a series of macro variable lists or a series of CALL EXECUTES.

Driving a Single Macro (rectangular control file)

When a single macro is to be called multiple times, the control file will tend to be rectangular. Consider the %PRINTIT macro, which has three parameters (DSN, VARLIST, and TITLE). If we wished to call this macro multiple times with different parameter values we could create a control file with one observation for each time the macro is to be called.

```
%macro printit(dsn=,varlist=,title=);
title1 "&title";
proc print data=&dsn;
   var &varlist;
   run;
%mend printit;
```

The data in this control file (WORK.SINGLE) has one variable for each parameter and one row for each time the macro is to be called. For a simple execution such as this one, a DATA _NULL_ step with a CALL EXECUTE routine can be used to generate the macro calls.

Control file for %PRINTIT

Obs	dsn	varlist	title
1	sashelp.class	name sex height weight	class data
2	summry	_all_	Summarized

```
data _null_;
 set single;
 call execute(catt('%printit(dsn=',dsn,',varlist=',varlist,',title=',title,')'));
 run;
```

For the two observations in the control file, the CALL EXECUTE routine will generate two macro calls, and these will be executed immediately after the DATA step has completed its execution.

```
%printit(dsn=sashelp.class,varlist=name sex height weight,title=class)

%printit(dsn=summry,varlist=_all_,title=Summarized)
```

Multiple Macros

Although the rectangular control file approach works great for multiple calls to a single macro, it is not very efficient when multiple macros are to be called. And it especially does not work well when the different macros have very different parameter structures. In this situation there may still be the need for a metadata control file, however a different approach must be taken.

The basic example used here deals with the generation of an Annual Report. The report is composed of a series of tables, charts, and graphs. Generalized macros have been written, which when executed create one or more of the components for the final report. Because the macros have been generalized, each macro can actually create any number of similar tables by varying the macro's parameters. This allows us to generate hundreds of different tables with only a few macros.

GENERALIZED MACROS

For the purposes of this paper the use of three simple macros will demonstrated. The beauty of the technique that is to be described is that it is completely expandable to any number of macros, each with any number of parameters.

```
%macro printit(dsn=,varlist=,title=);
title1 "&title";
proc print data=&dsn;
   var &varlist;
   run;
%mend printit;

%macro chart(dsn=, gvar=, yvar=, type= ,title=);
title1 "&title";
proc gchart data=&dsn;
   vbar &yvar/group=&gvar type=&type;
   run;
   quit;
%mend chart;

%macro summarize(dsn=,classlist=, var=);
proc summary data=&dsn;
   class &classlist;
   var &var;
   output out=summry mean= n= stderr=/autoname;
   run;
%mend summarize;
```

Figure 1

The %PRINTIT macro shown here and above generates a simple data listing. It has three parameters, which are used to name the data set, the variables to print, and a title.

The %CHART macro uses the GCHART procedure to generate a vertical bar chart. Parameters are used to name the data set of interest, the vertical variable, a grouping variable, the type of the chart to create, and a title.

The %SUMARIZE macro calls a PROC SUMMARY step and is used to summarize data prior to the generation of a report. The data set generated by this procedure can in turn be used by the %PRINTIT macro.

The macro calls that generate our annual report be called in the order that we would like our report tables to appear. Clearly with a number of macros being called and a series of often different parameters, the calls themselves are not

```
%printit(dsn=sashelp.class,varlist=name sex height weight,title=class data)
%chart(dsn=sashelp.class,yvar=weight,gvar=sex,type=percent, title=Weight distribution
for gender)
%summarize(dsn=sashelp.class,classlist=sex age,var=height weight)
%printit(dsn=summry,varlist=_all_,title=Summarized class data)
```

Figure 2

easily read. The construction of this type of series of macro calls is prone to error. It is easy to miss specify a parameter, skip a macro call, place calls in the wrong order, and other such coding problems. One way to make the code easier to read and maintain is to place one parameter per line of code.

```
%printit(dsn=sashelp.class,
         varlist=name sex height weight,
         title=class data)
%chart(dsn=sashelp.class,
       yvar=weight,
       gvar=sex,
       type=percent,
       title=Weight distribution for gender)
%summarize(dsn=sashelp.class,
           classlist=sex age,
           var=height weight)
%printit(dsn=summry,
         varlist=_all_,
         title=Summarized class data)
```

Figure 3

Figure 3 shows the same macro calls as were shown in Figure 2, however it is now much easier to visualize the individual parameters in each macro call. It is also much easier to assess the order of the macro calls and whether or not the order of the macro calls is appropriate for our annual report. All this and we have not yet even added any comments!

BUILDING METADATA

Creating a series of calls to generalized macros, such as was done in Figure 3, is often sufficient for most applications, but when you have hundreds of macro calls or if you calling macros that have more than three or four parameters, even coding them as is done in Figure 3 can be problematic. One approach that can be used to simplify the control of large numbers of macro calls is through the use of metadata.

The metadata approach moves all of the critical information provided in the code into data, while leaving the syntax behind. The information in Figure 4 is the same as in Figure 3 except all the code has been removed. In this particular example the metadata is being stored in a XLS file, however it could just as easily been in a CSV file, TEXT file, or any other file form that can be imported into SAS®.

	A	B	C
1	macro	parm	value
2	printit	dsn	sashelp.class
3	printit	varlist	name sex height weight
4	printit	title	class data
5	chart	dsn	sashelp.class
6	chart	yvar	weight
7	chart	gvar	sex
8	chart	type	percent
9	chart	title	Weight Distribution for Gender
10	summarize	dsn	sashelp.class
11	summarize	classlist	sex age
12	summarize	var	height weight
13	printit	dsn	summry
14	printit	varlist	_all_
15	printit	title	Summarized Class Data

Figure 4

One immediate advantage of this approach is that the metadata can be maintained by someone with little or no SAS knowledge. As we will see as we look at how this metadata is used, the technique is highly expandable, and can support drastic changes to the metadata without causing any changes to the code that uses the metadata.

In the approach used here we will read this metadata file into SAS using a PROC IMPORT step. Generally this will be the easiest way to bring the metadata into SAS, but as long as we end up with a SAS data set, it does not matter.

The PROC IMPORT is completely straight forward. In the PROC IMPORT step shown in Figure 5, the XLS file 2_METADATA.XLS is imported and the SAS data set WORK.METADATA is created. This data set has the same information in it as was contained in the XLS file. Depending on your version of SAS and your OS, the DBMS used by PROC IMPORT may be different, such as XLS, XLSX.

```
proc import file="&somepath\2_MetaData.xls"
            out=metadata
            replace
            dbms=excelcs;
   run;
```

Figure 5

Inspection of the data set WORK.METADATA, shown in Figure 6 shows that it contains the same information as is shown in Figures 2, 3, and 4.

We now have the ability to take advantage of the metadata in a DATA step. We can build lists of macro variables, as was demonstrated in Rosenbloom and Carpenter (2015), or we can build the macro calls directly and execute them through the use of the CALL EXECUTE routine. Because in this example we are only interested in executing a series of macro calls, the CALL EXECUTE approach will be used.

VIEWTABLE: Work.Metadata

	macro	parm	value
1	printit	dsn	sashelp.class
2	printit	varlist	name sex height weight
3	printit	title	class data
4	chart	dsn	sashelp.class
5	chart	yvar	weight
6	chart	gvar	sex
7	chart	type	percent
8	chart	title	Weight Distribution for Gender
9	summarize	dsn	sashelp.class
10	summarize	classlist	sex age
11	summarize	var	height weight
12	printit	dsn	summry
13	printit	varlist	_all_
14	printit	title	Summarized Class Data

Figure 6

BUILDING AND EXECUTING THE MACRO CALLS

Except for the actual SAS syntax the metadata contains all the information that we need to construct the macro calls. Our approach will be to read the data and use the power of the DATA step to create the macro call, including its parameters. The macro calls will then be submitted for execution.

For the first macro call, a call to the %PRINTIT macro, we need to construct the code shown in Figure 7. Because this code will be submitted for execution using CALL EXECUTE, the macro call itself must be masked during the execution of the DATA step itself. This allows the CALL EXECUTE routine to add the macro call to a buffer for execution after the termination of the DATA step.

```
%printit(dsn=sashelp.class,
         varlist=name sex height weight,
         title=class data)
```

Figure 7

We can mask the macro call through the use of single quotes. Effectively we need to construct an argument for CALL EXECUTE that both contains and masks the macro call. The single quotes prevents the DATA step from seeing the %PRINTIT as a macro call, and the macro call is constructed and passed out of the DATA step, where it is executed after the DATA step completes its own execution.

```
data _null_;
call execute('%printit(dsn=sashelp.class,
                       varlist=name sex height weight,
                       title=class data)');

run;
```

Figure 8

The DATA step in Figure 9 will read the metadata and utilize the information that it contains to build the masked macro call.

```
data _null_;
   length string $500; ❶
   retain string;
   set metadata;
   by macro notsorted; ❷
   if first.macro then do; ❸
      string = cats('%',macro,'(',parm,'=',value);
   end;
   else string = cats(string,',',parm,'=',value); ❹
   if last.macro then do; ❺
      string = cats(string,')');
      put string=; ❻
      call execute(string); ❼
   end;
   run;
```

Figure 9

❶ A character string variable is defined that will hold the macro call. In this particular instance the string length has been limited to 500 characters. This may be too short for some applications, but is sufficient for these examples.

❷ The BY statement enables the use of FIRST. and LAST. processing. The NOTSORTED keyword is needed because although the incoming data is grouped by macro name, it is not sorted by macro name. Note this logic fails if there are two successive calls to the same macro (see Extension 2 below for a coding solution).

❸ This is the first observation for this macro call. The call itself will be held by the variable STRING, so the macro name, its % sign, and the value of first parameter are added to the variable first.

❹ This is not the first observation for this macro, add the parameter and its value to the growing list.

❺ This is the last observation for this macro, add the last two closing parentheses.

❻ The value of STRING is written to the SAS Log so that you can see what it contains. This is of course optional and is only shown here to demonstrate the value held by the variable STRING.

❼ The value of the variable STRING, which contains the masked macro call, is passed to the buffer for execution after the DATA step terminates.

A portion of the SAS Log (Figure 10) shows the results of the PUT statement ❻, which was included in the DATA step so that we could visualize the generated code.

```
string=%printit(dsn=sashelp.class,varlist=name sex height weight,title=class data)
string=%chart(dsn=sashelp.class,yvar=weight,gvar=sex,type=percent,title=Weight Distribution for Gender)
string=%summarize(dsn=sashelp.class,classlist=sex age,var=height weight)
string=%printit(dsn=summry,varlist=_all_,title=Summarized Class Data)
```

Figure 10

The SAS Log also shows us the code that is actually executed after being written to the buffer by CALL EXECUTE. The first %PRINTIT macro call is shown in Figure 11.

```
NOTE: CALL EXECUTE generated line.
1   + %printit(dsn=sashelp.class,varlist=name sex height weight,title=class data)
```

Figure 11a

Depending on your version of SAS the macro may be expanded in the SAS Log.

```
NOTE: CALL EXECUTE generated line.
1   + title1 "class data";
1   +                     proc print data=sashelp.class;    var name sex height weight;    run;
```

Figure 12b

EXTENSION 1: MACRO CALLS WITHOUT PARAMETERS

The DATA step in Figure 9 assumes that each macro has at least one parameter. While this will generally be true, it will not always be true. Fortunately the coding changes are simple and fairly straightforward. We merely need to detect those macro calls without parameters, so that we can do some special handling.

```
data _null_;
   length string $500;
   retain string;
   set metadata;
   by macro notsorted;
   if first.macro then do;
      if parm=' ' then do; ❶
         string = cats('%',macro); ❷
      end;
      else do;
         string = cats('%',macro,'(',parm,'=',value);
      end;
   end;
   else string = cats(string,',',parm,'=',value);
   if last.macro then do;
      if parm ne' ' then do; ❸
         string = cats(string,')');
      end;
      put string=;
      call execute(string);
   end;
   run;
```

Figure 13

This is actually simple if we assume that the PARM variable will only be missing for macros without parameters. As this seems to be a reasonable assumption we can add the necessary logic to the DATA step in Figure 9.

❶ The assumption is that the PARM variable will be missing for macros without parameters.

❷ Only the % sign and macro name need to be concatenated.

❸ A single parenthesis is needed to close macro calls with parameters.

EXTENSION 2: CONTROLLING CALL ORDER

The DATA steps in Figures 9 and 12 will both fail if the same macro is called successively. This is an artifact of the use of the macro name in the BY statement. As a result there is no way to differentiate between two different calls to the same macro when one directly follows the other. A common way to solve this problem is to add a numbering system of some kind to the metadata. For reporting systems very often a table number will be included anyway and we can take advantage of the this number to differentiate the macro calls.

	A	B	C	D
1	table	macro	parm	value
2	1.1.1	chart	dsn	sashelp.class
3	1.1.1	chart	yvar	weight
4	1.1.1	chart	gvar	sex
5	1.1.1	chart	type	percent
6	1.1.1	chart	title	Table: Weight Distribution fo
7	1.1.2	summarize	dsn	sashelp.class
8	1.1.2	summarize	classlist	sex age
9	1.1.2	summarize	var	height weight
10	1.1.2	printit	dsn	summry
11	1.1.2	printit	varlist	_all_
12	1.1.2	printit	title	Table: Summarized Class Dat
13	1.1.3	printit	dsn	sashelp.class
14	1.1.3	printit	varlist	name sex height weight
15	1.1.3	printit	title	Table: class data

Figure 14

Adding a table number to the metadata gives us a way to not only augment our titles, but to solve the problem of successive macro calls to the same macro. In Figure 13 the TABLE column has been added and the order of the macro calls has been changed. Notice that now that there are two successive calls to %PRINTIT, but that each has its own distinct table number (1.1.2 and 1.1.3).

In the titles 'Table:' has been added. In the DATA step that processes the metadata, the table number will be added to the title using this code and the TRANWRD function.

```
data _null_;
   length string $500 value $100; ❶
   retain string;
   set metadata;
   by table macro notsorted; ❷

   * Insert Table number if requested; ❸
   value = tranwrd(value,'Table:',catx(' ','Table:',table));

                    . . . Code not shown . . .
```

Figure 15

❶ The length of VALUE has been increased to accommodate the table number that is inserted at ❸.

❷ The variable TABLE has been added to the BY statement. This ensures that successive macro calls of the same macro can be distinguished.

❸ The TRANWRD function is used to add the table number wherever the 'Table:' is detected.

The remainder of the DATA step remains unchanged from Figure 12.

EXTENSION 3: WORKING WITH LISTS WITHIN A PARAMETER

In the previous examples, the list of values in the variable list (PARAM=VARLIST) for the %PRINTIT macro (line 14 in Figure 13) were on a single line. When the user does not have full control over how the metadata control table is created, it may not be possible to place all the items in a list on a single line. When this can happen our data step needs to be flexible enough to accommodate this form of the metadata.

A portion of some metadata that is in this form is shown in Figure 15. Rather than appearing on a single line the list of variables used with the VARLIST parameters appears with one variable per line. This causes the parameter name (VARLIST) to repeat (lines 16-19 in Figure 15). One obvious advantage of this form is that the list of items can be very long.

15	1.1.3	printit	dsn	sashelp.class
16	1.1.3	printit	varlist	name
17	1.1.3	printit	varlist	sex
18	1.1.3	printit	varlist	height
19	1.1.3	printit	varlist	weight
20	1.1.3	printit	title	Table: class data

Figure 16

The changes to the DATA step require us to detect lists that span multiple observations. This is accomplished by adding the variable PARM to the BY statement, which gives us the ability to use FIRST. and LAST. processing on the PARM variable.

```
data _null_;
   length string $500 value $100;
   retain string;
   set metadata;
   by table macro parm notsorted; ❶

   value = tranwrd(value,'Table:',catx(' ','Table:',table));

   if first.macro then do;
      string = cats('%',macro); ❷
      if parm ne ' ' then string = cats(string,'(',parm,'=',value);
   end;
   else if first.parm then string = cats(string,',',parm,'=',value); ❸
   else  string = catx(' ',string,value); ❹
   if last.macro then do;
      if parm ne ' ' then string = cats(string,')');
      put string=;
      call execute(string);
   end;
   run;
```

❶ The variable PARM is added to the BY statement.

❷ The logic to detect macro calls without parameters has been simplified.

❸ If this is the first occurrence of this parameter, add the parameter name and equal sign as well as its value to the macro call.

❹ If it is not the first occurrence of the parameter then just add the parameter value separate by a space.

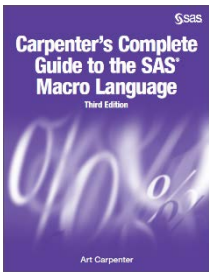
Figure 17

CONCLUSION

Applications for the use of metadata control files to manage the execution of a series of macro calls are many and varied. Most commonly the metadata is of a form where each observation of the metadata forms a distinct macro call. While this form of metadata may be the most commonly encountered, it is by no means the only metadata form. Through the power of the DATA step we can make use of metadata that may come to us in a variety of forms.

In this paper the discussion centers on metadata in a vertical format. This type of control file tends to use multiple observations to describe each macro call and may require the use of DATA step logic to build each macro call. But because we are working in the DATA step, this approach can be highly flexible.

REFERENCES



Carpenter, Art, 2016, *Carpenter's Complete Guide to the SAS® Macro Language, Third Edition*, SAS Institute Inc, Cary, NC. http://support.sas.com/publishing/authors/carpenter.html

Fehd, Ronald and Art Carpenter, 2007, "List Processing Basics: Creating and Using Lists of Macro Variables" by Ronald Fehd and Art Carpenter which was presented at the 2007 SAS Global Forum (Paper 113-2007). The discussion of the paper looks at different approaches used in the automation of programs by using various kinds of macro variable lists. This paper appears in proceedings of a number of conferences, including: SASGF(2007), WUSS (2008), MWSUG (2009), SESUG (2009). http://www.caloxy.com/papers/72Lists.pdf

Rosenbloom, Mary F. O. and Carpenter, Arthur L., 2015, "Are You a Control Freak? Control Your Programs – Don't Let Them Control You!", presented at the SAS Global Forum 2015 Conference in Dallas, Texas (paper 2220-2015). http://support.sas.com/resources/papers/proceedings15/2220-2015.pdf

ABOUT THE AUTHOR

Art Carpenter is a SAS Certified Advanced Professional Programmer and his publications list includes; five books and numerous papers and posters presented at SAS Global Forum, SUGI, PharmaSUG, WUSS, and other regional conferences. Art has been using SAS since 1977 and has served in various leadership positions in local, regional, and international user groups.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Arthur L. Carpenter
California Occidental Consultants
10606 Ketch Circle
Anchorage, AK 99515

(907) 865-9167
art@caloxy.com
www.caloxy.com

View Art Carpenter's paper presentations page at:
http://www.sascommunity.org/wiki/Presentations:ArtCarpenter_Papers_and_Presentations

TRADEMARK REFRENCES