PharmaSUG 2021 - Paper QT-007

How to Write a SAS® Open Code Macro to Achieve Faster Updates

Frederick Cieri, CSG (Clinical Solutions Group) an IQVIA business

ABSTRACT

This paper demonstrates how to write a SAS® macro using mostly open code statements to make a macro easier to debug, run, and understand. A macro is not an open code item, but open code techniques can be used to simplify macro code. Instead of having to run a whole macro to get results, an open code macro can be submitted by running just one data step or procedure at a time which saves time in running code and making updates. Examples are given to convert nested macros with '%if %then' and '%do' loops code to macros using primarily open code statements. This topic would benefit any macro programmer from beginner to advanced programmer.

INTRODUCTION

In my career as a macro developer in the pharmaceutical industry, a macro written in a heavily nested macro style with many '%if %then' and '%do' loops statements can be very difficult to understand because the whole macro must be submitted repeatably to comprehend and modify the code. A '%return' statement can be used to stop a macro at a specific line in a macro, but all the code prior to the specific line will have to be submitted every time for any update. Running all or part of a large macro with thousands of lines code or with large data sets can be time consuming. Using a small data set to test a macro can reduce processing time, but some test cases must be run with larger data sets. Contrast this to an open code program where each data step and procedure can be run separately. A macro can be made to mimic an open code program by making local parameters global and converting nested macro code to open code. To properly run any open code section of a macro, the permanent and temporary data sets must not be deleted after a macro run and must be uniquely named to assure traceability. After a macro is fully developed then the temporary data sets can be deleted at the end of the macro. A series of examples will be displayed to convert a nested style macro to an open code style macro. Given the ability to run practically any section of an open code macro interactively, this method should make macro modifications easier and faster to perform.

CONVERTING A NESTED MACRO

A series of examples will be displayed to convert a nested style macro to an open code style macro. The examples of the conversion are listed below with hyperlinks to the code:

TABLE OF CONTENTS FOR NESTING CONVERSION EXAMPLES

1 CONVERT THE LOCAL MACRO VARIABLES TO GLOBAL

After the code and macro call in the example below are submitted, each data step and procedure can be highlighted and submitted independently like an open code program because the local parameters in the macro are now global. The last highlighted data step in the macro used to

create the global macro parameters from the local macro parameters can be placed in the beginning, end, or any location in the code. To get the global parameters from the SAS help table of macro values, the use of system macro variable 'sysmacroname' for macro name makes the code dynamic because now the name of the macro does not need to be hard coded. The newly created global macro variables are placed in the '__z_macro_parms_global' data set to easily view the macro parameters, but a 'data _null_' could be used instead which creates no output data set. The print out of the view SASHELP.VMACRO in '1.1 View Macro Variables' shows the local macro parameters from the macro have been made global. To avoid scope problems leading to over-writing macro variables, the data step to convert local to global parameters should be removed prior to the final version of the macro. View the code below:

```
%************************************
Example:(1) Make the local macro parameters global during development
to easily use the parameters in an open code environment using dynamic
code.

Macro shows the contents of a data set and prints out selected
variables.

  parameters
   indata = input data set
   invars = input variables separated by a space
****************************************;
%macro test_1_macro(indata=, invars= );

  %** code ***;
  proc contents data = &indata;
   title "data = &indata";
  run;


  proc print data = &indata;
    title "data = &indata";
    var &invars;
  run;


  %************************************
    Make local parameters global. The sysmacroname macro variable is an
    automatic macro that contains the macro name.
    Remove code after development.
  ************************************;
  data __z_macro_parms_global;
          set sashelp.vmacro;
         where scope = "&sysmacroname";
         call symputx(name,symget(name),'G');
  run;


 %*** View macro variables     ***;
 proc print data = sashelp.vmacro;
  where name in ('INDATA' 'INVARS');
  title 'data = sashelp.vmacro';
 run;

%mend test_1_macro;
```

```
%test_1_macro(indata=sashelp.cars, invars=model MPG_city MPG_Highway
msrp );
```

1.1 VIEW MACRO VARIABLES

```
data = sashelp.vmacro
```

Obs	scope	name	offset	value
1	TEST_1_MACRO	INDATA	0	sashelp.cars
2	TEST_1_MACRO	INVARS	0	model MPG_city MPG_Highway msrp
3	GLOBAL	INDATA	0	sashelp.cars
4	GLOBAL	INVARS	0	model MPG_city MPG_Highway msrp

2 USE SAS® BASE CODE WITH MACRO PARAMETERS INSTEAD OF MACRO CODE

This example gives a data set with a series of numeric variables that are summarized and later printed. The macro code style is given first followed by an open code macro which converts the nested macro code by replacing the '%if %then' and '%do %end' statements with data step statements where key comments are highlighted. After viewing many macros, many '%if %then' and '%do %end' statements are used but are not needed because open code statements could easily be used instead giving access to an open code macro. This open code example and all the subsequent examples use the code from the 'Convert the local macro variables to globaL' example to convert the local macro parameters to global. After the open code macro and macro call in the example below are submitted for the first time, the macro can now be submitted by each data step or procedure section. View the code below:

```
%*************************************
 Example:(2) In data steps, use SAS base code with macro parameters where
  possible instead of macro code.
  Use 'if then' statements versus '%if %then' statement.
  Use Arrays without '%do loop'.

 Macro summarizes series of numeric variables that are later printed.
 parameters
   indata   = input data set
   num_vars = number of variables in series of x1-x*
   outdata  = output data set
   debug    = Show test prints? Value- yes or no.
*************************************;


%*** Data set series of numeric x variables      ***;
data test;
  x1=5;
  x2=3;
  x3=-1;
  output;
run;
```

2.1.1 %*** Macro code style ***;

```
%macro test_2_macro(indata=  ,num_vars= ,outdata= ,debug=);
```

```
  Data &outdata;
    set &indata;
    array ar_x(*)  %do i=1 %to &num_vars;
                       x&i
                   %end;;

    sum_var = 0;

    %do i=1 %to &num_vars;
      if  ar_x(&i) > 0 then  sum_var = sum_var + ar_x(&i);
    %end;

    %if %qupcase(&debug) = YES %then put sum_var = ;;
  run;


  proc print data = &outdata;
      title "data = &outdata";
  run;

%mend test_2_macro;


%test_2_macro(indata=test  ,num_vars=3 ,outdata=new ,debug=Yes);
```

2.1.2 %*** Open code style ***;

```
%macro test_2_open(indata=  ,num_vars= ,outdata= ,debug=);

  data &outdata;
    Set &indata;

    %** Use an array series instead of a macro loop **;
    array ar_x(*)  x1-x&num_vars  ;

    sum_var = 0;

    %** Use an array loop instead of a macro loop **;
    do i=1 to dim(ar_x);
     if  ar_x(i) > 0 then  sum_var = sum_var + ar_x(i);
    end;

    %** Use a data statement with parameter to remove the '%if %then' **;
    if upcase("&debug") = 'YES' then put sum_var=;
  run;


  proc print data = &outdata;
      title "data = &outdata";
  run;


  %**************************************
```

```
    Make local parameters global. The sysmacroname macro variable is an
automatic macro that contains the macro name.
    Remove code after development.
  ****************************************;
  data __z_macro_parms_global;
        set sashelp.vmacro;
        where scope = "&sysmacroname";
        call symputx(name,symget(name),'G');
  run;


%mend test_2_open;


%test_2_open(indata=test  ,num_vars=3, outdata=new ,debug=Yes);
```

3 USE '%SYSFUNC' MACRO FUNCTION WITH 'IFC' FUNCTION INSTEAD OF '%IF %THEN %ELSE' AND '%DO %END'

This example uses the same macro and nested macro code to open code transformation techniques from the 'Use SAS® base code with macro parameters instead of macro code' example that summarized a series of numeric variables, but now a parameter is added to determine the length of a new character variable. The macro code style is given first followed by an open code macro which converts the nested macro code by replacing a '%if %then' statement with the highlighted '%sysfunc' macro function and 'ifc' data statement used together to create a macro function to select programming text. After submitting the open code macro and macro call for the first time, the macro can now be submitted by each data step or procedure section. View the code below:

```
%****************************************
 Example:(3) Use '%sysfunc' macro function with 'ifc' data step
function for conditional selection to use instead
 of '%if %then %else' and '%do %end'.

 Macro summarizes series of numeric variables that are later printed,
 and a character variable is created with length based on a parameter.

   indata     = input data set
   num_vars   = number of variables in series of x1-x*
   length_out = length of output character variable.  Enter blank for
                length of 100 or integer > 0.
   outdata    = output data set
   debug      = Show test prints? Value- yes or no.
****************************************;


%*** Data set series of numeric x variables      ***;
data test;
  x1=5;
  x2=3;
  x3=-1;
  output;
run;
```

3.1.1 %*** Macro code style ***;

```
%macro test_3_macro(indata= ,num_vars= ,length_out= ,outdata= ,debug=);

  %*** Length of aedecod_adj has default of 100 when no value is
  entered ***;
  Data &outdata;
   length aedecod_adj
      %if       %nrbquote(&length_out) = %str( )  %then $100;
      %else                                              $&length_out;
    ;
    set &indata;
    array ar_x(*)  %do i=1 %to &num_vars;
                       x&i
                   %end;;

    sum_var = 0;

    %do i=1 %to &num_vars;
      if  ar_x(&i) > 0 then  sum_var = sum_var + ar_x(&i);
    %end;

    %if %qupcase(&debug) = YES %then put sum_var = ;;

    aedecod_adj = 'new value';
  run;


  proc print data = &outdata;
      title "data = &outdata";
  run;


  proc contents data = &outdata;
      title "data = &outdata";
  run;

%mend test_3_macro;

options mprint;
%test_3_macro(indata=test ,num_vars=3 ,length_out= ,outdata=new
,debug=Yes);
%test_3_macro(indata=test ,num_vars=3 ,length_out=115 ,outdata=new
,debug=Yes);
```

3.1.2 %*** Open code style ***;

```
%macro test_3_open(indata=  ,num_vars= ,length_out= ,outdata= ,debug=);


  %*** Length of aedecod_adj has default of 100 when no value is
entered ***;
  data &outdata;
```

```
    %** The %sysfunc with ifc replace %if to achieve open code. **;
    length aedecod_adj
           $%sysfunc(ifc( &length_out = %str( ),100,&length_out) );

    Set &indata;

    %** Use an array series instead of a macro loop **;
    array ar_x(*)  x1-x&num_vars  ;

    sum_var = 0;

    %** Use an array loop instead of a macro loop **;
    do i=1 to dim(ar_x);
     if  ar_x(i) > 0 then  sum_var = sum_var + ar_x(i);
    end;

    %** Use a data statement with parameter to remove '%if %then' **;
    if upcase("&debug") = 'YES' then put sum_var=;

    aedecod_adj = 'new value';
  run;


  proc print data = &outdata;
      title "data = &outdata";
  run;


  proc contents data = &outdata;
      title "data = &outdata";
  run;


  %*************************************
    Make local parameters global. The sysmacroname macro variable is an
automatic macro that contains the macro name.
    Remove code after development.
  *************************************;
  data __z_macro_parms_global;
         set sashelp.vmacro;
         where scope = "&sysmacroname";
         call symputx(name,symget(name),'G');
  run;


%mend test_3_open;


options mprint;
%test_3_open(indata=test ,num_vars=3 ,length_out=    ,outdata=new
,debug=Yes);
%test_3_open(indata=test ,num_vars=3 ,length_out=115 ,outdata=new
,debug=Yes);
```

4 USE 'DATA _NULL_' TO REPLACE '%IF %THEN' FOR AN INTEGRITY CHECK

Based on the macro from the ‘Use ‘%sysfunc’ macro function with ‘ifc’ function instead of ‘%if %then %else’ and '%do %end' example, this example shows how an integrity check for the proper values of the ‘debug’ parameter can be written with a ‘data _null_’ to replace ‘%if %then’ statement. The macro code style is given first followed by an open code macro with the comments prior to the ‘data _null_’ highlighted. From my experience, most macro integrity checks use nested macro code because macro code is usually used within macros, but open code could be used instead giving the added benefit of an interactive submission. After submitting the open code macro and macro call for the first time, the macro can now be submitted by each data step or procedure section. View the code below:

```
%*************************************
 Example:(4) Use 'data _null_' statement to replace '%if %then' statements
used for an integrity check.

Macro summarizes series of numeric variables that are later printed,
and a character variable is created with length based on a parameter.

   indata     = input data set
   num_vars   = number of variables in series of x1-x*
   length_out = length of output character variable.  Enter blank for
                length of 100 or integer > 0.
   outdata    = output data set
   debug      = Show test prints? Value- yes or no.
*************************************;

%*** Data set series of numeric x variables      ***;
data test;
  x1=5;
  x2=3;
  x3=-1;
  output;
run;
```

4.1.1 %*** Macro code style ***;

```
%macro test_4_macro(indata= ,num_vars= ,length_out= ,outdata= ,debug=);

  %local exit_macro;

  %let exit_macro = 0;

  %***  Integrity check. Check debug parameter.    ***;
  %if %superq(debug) = %str( ) %then
  %do;
      %put %str(ER)ROR: The parameter &debug must not be blank.;
      %let exit_macro =1;
  %end;
  %else %if %qlowcase(&debug) ne yes and %qlowcase(&debug) ne no %then
  %do;
      %put %str(ER)ROR: The parameter debug must equal 'yes' or 'no'.;
      %let exit_macro =1;
  %end;


  %***  Abort     ***;
```

```
  %if &exit_macro = 1 %then %goto exit;


  %*** Length of aedecod_adj has default of 100 when no value is entered
***;

  %** The rest of the code is the same as the previous macro code style
example  **;

  %exit:

%mend test_4_macro;

options mprint;
%test_4_macro(indata=test ,num_vars=3 ,length_out= ,outdata=new
,debug=Yes);
%test_4_macro(indata=test ,num_vars=3 ,length_out=115 ,outdata=new
,debug=bad value);
```

4.1.2 %*** Open code style ***;

```
%macro test_4_open(indata=  ,num_vars= ,length_out= ,outdata= ,debug=);

  %local exit_macro;

  %let exit_macro = 0;

  %***  Integrity check. Check debug parameter. 'Symget' function
  retrieves macro values with no quoting issues.   ***;
  data _null_;
     debug = symget('debug');

     if debug = ' '  then
     do;
        call symputx('exit_macro','1','L');
        put 'ER' "ROR: The parameter debug must not be blank.";
     end;

     if  lowcase(debug) not in ('yes' 'no') then
     do;
        call symputx('exit_macro','1','L');
        put 'ER' "ROR: The parameter debug must equal 'yes' or 'no'.";
     end;

   run;


%***  Abort     ***;
%if &exit_macro = 1 %then %goto exit;


%*** Length of aedecod_adj has default of 100 when no value is entered
***;
```

```
%** The rest of the code is the same as the previous open code style
example **;


  %************************************
    Make local parameters global. The sysmacroname macro variable is an
automatic macro that contains the macro name.
    Remove code after development.
  ************************************;
  data __z_macro_parms_global;
         set sashelp.vmacro;
         where scope = "&sysmacroname";
         call symputx(name,symget(name),'G');
  run;


  %exit:

%mend test_4_open;


options mprint;
%test_4_open(indata=test ,num_vars=3 ,length_out=    ,outdata=new
,debug=Yes);
%test_4_open(indata=test ,num_vars=3 ,length_out=115 ,outdata=new
,debug=bad value);
```

5 USE 'DATA _NULL_' TO DERIVE A MACRO VARIABLE TO REPLACE '%IF %THEN' AND '%DO LOOP'

This example for a macro to determine the printed format for a variable uses a 'data _null_' to create a macro variable to replace a series of macro variables created with '%if %then' and '%do %then' statements. Creating one macro variable instead of a series of macro variable makes the code more readable. The macro code style is given first followed by an open code macro where key comments are highlighted. After submitting the open code macro and macro call for the first time, the macro can now be submitted by each data step or procedure section. View the code below:

```
%************************************
 Example:(5)
  Use Data _null_ to derive macro variables to replace '%if %then', and
'%do loop'.

Macro limits the length of character variables based upon a boundary
parameter and prints numeric variables with the
associated format or if no format is associated then the numeric format
parameter input will be used.

indata               = input data set
char_format_boundary = Enter integer > 0 where character variable length
in the input data set above this parameter value will have the print out
limited to the parameter value.
num_format           = enter numeric format to use for print out when
numeric variable has no format associated in the input data set
```

```
****************************************;

%*** Data set series of numeric x variables      ***;
data test5;

  format x_date date9.;
  x_date = date();

  x1=5;
  x2=3;
  x3=-1;

  name = 'charlie';
  text = repeat('a',200);
  output;
run;
```

5.1.1 %*** Macro code style ***;

```
%macro test_5_macro(indata= ,char_format_boundary= ,num_format= );

  %local i data_id var_num var_name var_length var_fmt;

  %let data_id = %sysfunc(open(&indata));
  %let var_num = %sysfunc(attrn(&data_id,nvars));

  %***  Loop to derive macro variables name with format   ***;
  %do i=1 %to &var_num;
     %let var_name   = %sysfunc(varname(&data_id,&i));
     %let var_type   = %sysfunc(vartype(&data_id,&i));
     %let var_length = %sysfunc(varlength(&data_id,&i));
     %let var_fmt    = %sysfunc(varformat(&data_id,&i));

     %put &var_name &var_type &var_length &var_fmt;

     %local var_&i;

     %if  &var_type = C %then
     %do;
         %if &var_length > &char_format_boundary %then
            %let var_&i = &var_name $&char_format_boundary..;
         %else  %let var_&i = &var_name $&var_length..;
     %end;
     %else %if  &var_type = N %then
     %do;
         %if %nrbquote(&var_fmt) = %str( ) %then
             %let var_&i = &var_name &num_format..;
         %else %let var_&i = &var_name &var_fmt.;
     %end;
  %end;


  %let close   = %sysfunc(close(&data_id));
```

```
  proc print data = &indata;
      format
         %do i =1 %to &var_num;
            &&var_&i
         %end;
      ;
      title "data = &indata";
  run;


%mend test_5_macro;

options mprint;

%*** Macro code     ***;
%test_5_macro(indata=test5 ,char_format_boundary=15 ,num_format=best );
```

5.1.2 %*** Open code style ***;

```
%macro test_5_open(indata= ,char_format_boundary= ,num_format= );

 %*** Data step to derive macro variable to hold variable names with
formats   ***;
 data _null_;
   %** A large length in needed to allow for many variables ***;
   length var_format_statement $10000;

   data_id = open("&indata");
   var_num = attrn(data_id,'nvars');

   %*** Loop to concatenate the variable name with format     ***;
   do i=1 to var_num;

     var_name   = varname(data_id,i);
     var_type   = vartype(data_id,i);
     var_length = varlength(data_id,i);
     var_fmt    = varformat(data_id,i);

     put var_name var_type var_length var_fmt;
     var_format_statement = catx(' ',var_format_statement,var_name);

     if  var_type = 'C' then
     do;
         if var_length > &char_format_boundary then
              var_format_statement =
               catx(' ',var_format_statement,"$&char_format_boundary..");
         else var_format_statement =
                catx(' ',var_format_statement,cats('$',var_length,'.'));
     end;
     else if  var_type = 'N' then
     do;
         if var_fmt = ' ' then
           var_format_statement = catx(' ',var_format_statement,"best." );
         else
           var_format_statement = catx(' ',var_format_statement,var_fmt);
```

```
      end;

   end;

   call symputx('var_format_statement',var_format_statement,'L');
   close   =   close(data_id);
  run;


   %*** Use one macro variable for format instead of a series of macro
   variables in a loop ***;
   proc print data = &indata;
       format &var_format_statement;
       title "data = &indata";
   run;


   %*************************************
     Make local parameters global. The sysmacroname macro variable is an
automatic macro that contains the macro name.
     Remove code after development.
   *************************************;
    data __z_macro_parms_global;
          set sashelp.vmacro;
          where scope = "&sysmacroname";
          call symputx(name,symget(name),'G');
   run;


%mend test_5_open;

options mprint;

%*** Open code     ***;
%test_5_open(indata=test5 ,char_format_boundary=15 ,num_format=best );
```

6 USE 'CALL EXECUTE' FUNCTION TO REPLACE '%IF %THEN','%DO LOOP'

This example based on the previous ‘Use ‘Data _null_’ to derive a macro variable to replace '%if %then' and '%do loop' example for a macro to determine the printed format for a variable uses a ‘call execute’ statement with a data set of code to submit to replace a series of macro variables created with ‘%if %then’ and ‘%do %then’ statements. The ‘call execute’ statement is a versatile tool in submitting code. The 'data _null_' in the previous example has a maximum 32,000 length limitation in the concatenated character variable to hold the variables. This could be overcome in the ‘data _null_’ with multiple macro variables, but the 'call execute' function provides an alternative with no length limitation because each variable is on a separate row in the data set holding the code to execute. Please read the documentation for the proper usage of 'call execute' because the function does have a limitation when calling macros that create macro variables which can be overcome with the proper coding techniques. In this example, the call execute only submits SAS® statements and no macro is called. After submitting the open code macro and macro call for the first time, the macro can now be submitted by each data step or procedure section. View the code below where key comments are highlighted:

```
%*************************************
 Example:(6)
```

```
  Repeat of previous example (limiting character variable and writing
numeric variable with a format).
  Use 'call execute' function to replace '%if %then','%do' loop.  The
'data _null_' in the previous example has a limitation in
  the length of the concatenated character variable to hold the variables.
This could be overcome in the data _null_ with
  multiple macro variables, but the 'call execute' function provides an
alternative with no length limitation.  Please read the
  documentation for 'call execute' because the function does have a
limitation when calling macros that create macro variables with
  'call symput' function.  In this example, the call execute only submits
SAS statements and no macro is called.

Macro limits the length of character variables based upon a boundary
parameter and prints numeric variables with the
associated format or if no format is associated then the numeric format
parameter input will be used.

indata               = input data set
char_format_boundary = Enter integer > 0 where character variable length
                       in the input data set above parameter value will
                       have the printout limited to the parameter value.
num_format           = enter numeric format to use for printing out when
                       numeric variable has no format
                       associated in the input data set

****************************************;


%*** Data set series of numeric x variables      ***;
data test6;

  format x_date date9.;
  x_date = date();

  x1 = 5;
  x2 = 3;
  x3 = -1;

  name = 'charlie';
  text = repeat('a',200);
  output;
run;


%*** Open code style    ***;
%macro test_6_open(indata= ,char_format_boundary= ,num_format= );

 %*** Create code to execute in a call execute in a future step    ***;
 data exe_code;
   length code $300;

   data_id = open("&indata");
   var_num = attrn(data_id,'nvars');

   %*** loop through to put one variable with format on each line of the
   data set     ***;
```

```
   do i=1 to var_num;
     var_name   = varname(data_id,i);
     var_type   = vartype(data_id,i);
     var_length = varlength(data_id,i);
     var_fmt    = varformat(data_id,i);

     put var_name var_type var_length var_fmt;

     code = var_name;

     if  var_type = 'C' then
     do;
         if var_length > &char_format_boundary then
              code = catx(' ',code,"$&char_format_boundary..");
         else code = catx(' ',code,cats('$',var_length,'.'));
     end;
     else if  var_type = 'N' then
     do;
         if var_fmt = ' ' then code = catx(' ',code,"best." );
         else                  code = catx(' ',code,var_fmt);
      end;

    output;
  end;

   close   =   close(data_id);
 run;


%*** Execute code for proc print with derived format statement. ***;
 data _null_;
    set exe_code end = last;

    if _n_ = 1 then
    do;
       call execute('proc print data = &indata;');
       call execute(  'format');
    end;

    call execute(code);

    if last then
    do;
       call execute(  ';');
       call execute('title "data = &indata";');
       call execute('run;');
    end;

  run;


  %*************************************
    Make local parameters global. The sysmacroname macro variable is an
 automatic macro that contains the macro name.
    Remove code after development.
  *************************************;
  data __z_macro_parms_global;
```

```
          set sashelp.vmacro;
          where scope = "&sysmacroname";
          call symputx(name,symget(name),'G');
  run;

%mend test_6_open;

options mprint;

%*** Open code     ***;
%test_6_open(indata=test6 ,char_format_boundary=15 ,num_format=best );
```

7 USE 'DATA _NULL_' TO CREATE REUSABLE MACRO VARIABLES WITHIN A '%DO' LOOP

This example to print character variables and summarize numeric variables uses a ‘data _null_’ to create reusable macro variables with a ‘%do’ loop to create a hybrid nested and open code style of macro. The nested macro code style is given first followed by the hybrid macro where key comments and code are highlighted. This style is especially useful in making it possible to submit bits of code from large macro ‘%do’ loops. To process part of the ‘%do’ loop interactively, first, submit the macro and macro call. Second, create a global value for the loop counter macro variable 'i' (%let i=1;) from the possible values of 1 to the number of data sets entered in the ‘indata’ macro parameter. Third, submit the ‘data _null_’ immediately after the “%do i’ loop statement to create a value for the reusable macro variable for the ‘%do i’ loop. Lastly, submit each data step of procedure in the loop interactively. To understand the flow of the hybrid macro, the print out in section ‘7.1.3 View conversion of parameters to data’ following the hybrid code macro shows how the macro input data parameter was converted to a data set. View the code and print out below:

```
%*************************************
 Example:(7)
  Use 'data _null_' to create reusable macro variables within a '%do' loop
to increase the use of open code.
  With this technique, each sequence of the loop can be executed
interactively once the macro loop variable is made global.

 Macro reads in each data set and prints the character variables and
summarizes the numeric variables with proc means.

indata   = input data sets separated by a space. ie. sdtm.dm sdtm.ae
obs      = number of observations to print
debug    = View test prints. Yes/NO

*************************************;
```

7.1.1 %*** Macro code style ***;

```
%macro test_7_macro(indata= ,obs= ,debug=  );

  %local i data_parse_i data_id var_num j var_name ;
  %let i=1;

  %*** Get character and numeric variables into separate macro variables
   ***;
  %do %while(%scan(&indata,&i,%str( )) ne %str( ) );
  %*** Start of %do while loop data parse    ***;
```

```
  %local data_parse_&i var_char_&i var_num_&i;


  %let data_parse_&i  = %scan(&indata,&i,%str( ));
  %let data_id        = %sysfunc(open(&&data_parse_&i));
  %let var_num        = %sysfunc(attrn(&data_id,nvars));

  %local var_char_&i var_num_&i;

  %*** loop to find the variable type and output the macro variables
  holding character or numeric variables    ***;
  %do j=1 %to &var_num;
  %*** Start loop metadata      ***;
     %let var_name   = %sysfunc(varname(&data_id,&j));
     %let var_type   = %sysfunc(vartype(&data_id,&j));

     %if &var_type = C %then %let var_char_&i = &&var_char_&i &var_name;
     %else %if  &var_type = N %then
           %let var_num_&i  = &&var_num_&i  &var_name;

     %if %qupcase(&debug) = YES %then
     %do;
        %put ... ;
        %put data_parse_&i=&&data_parse_&i var_name =&var_name
             var_type=&var_type ;
        %put var_char_&i = &&var_char_&i ;
        %put var_num_&i  = &&var_num_&i ;
     %end;

  %*** End loop metadata      ***;
  %end;

  %let close   = %sysfunc(close(&data_id));
  %let i = %eval(&i +1);

%*** End of %do while loop data parse     ***;
%end;


%*** Print character variables. Summarize numeric variables     ***;
%let i=1;

%do %while(%scan(&indata,&i,%str( )) ne %str( ) );
%*** Start print/summary loop   ***;

  %if %nrbquote(&&var_char_&i) ne %str( ) %then
  %do;
      proc print data = &&data_parse_&i(obs = &obs);
       var &&var_char_&i;
       title "Print Character variables. data=&&data_parse_&i obs=&obs";
      run;
  %end;

  %if %nrbquote(&&var_num_&i) ne %str( ) %then
  %do;
      proc means data = &&data_parse_&i;
```

```
        var &&var_num_&i;
        title "Summarize Numeric variables. data = &&data_parse_&i ";
       run;
    %end;

     %let i = %eval(&i +1);

  %*** End of %do while loop data parse     ***;
  %end;

%mend test_7_macro;

options mprint;

%*** Macro code     ***;
%test_7_macro(indata=sashelp.cars sashelp.demographics  sashelp.comet
,obs=5 ,debug=yes);
```

7.1.2 %*** Hybrid macro and open code style ***;

```
%macro test_7_open(indata= ,obs= ,debug=  );

  %*** Puts indata parameter into a data set with one row per data set.
  Find number of data sets.    ***;
  data parms ;
    %*** Derive local debug to comment out code    ***;
    Call symputx('debug_local',
                 ifc(lowcase(symget('debug'))='yes','','*'),'L');

    length indata indata_i $%length(&indata) ;
    indata = symget('indata');
    count_indata  = countw(indata,' ');

    %*** number of data sets     ***;
    call symputx('count_indata',put(count_indata,best.),'L');

    do i=1 to count_indata;
      indata_i = scan(indata,i,' ');
      output;
    end;
  run;


  %*** Convert parameters to data. ***;
  proc print data = parms;
     title 'data = parms. ';
  run;


  %** &debug_local equal blank or * and replaces need for '%if' to see
  test print. **;
  %&debug_local.put count_indata=&count_indata;


  %local i indata_i var_char;
```

```
%*** Loop for print characters or summarize numeric variables.   ***;
%*** To process interactively, create a global value for macro ***;
%*** variable 'i' from 1 to the number of data sets in data parms. ***;
%*** Then submit the highlighted 'data null' immediately after the '%do
i' loop ***;
%do i=1 %to &count_indata;
%*** Start of %do while loop print or summarize   ***;

  %*** Create local reusable macro variables to process by reading ***;
  %*** one observation of data set 'parms' per loop.   ***;
  data _null_;
    set parms(firstobs = &i obs = &i);
    call symputx('indata_i',indata_i,'L');
  run;

  %&debug_local.put i=&i indata_i=&indata_i;


  %*** Get metadata    ***;
  proc contents data = &indata_i out = cont_out_&i noprint;
  run;


  %*** Find Character variables     ***;
  %let var_char=;

  proc sql noprint;
    select name into: var_char
    separated by ' '
    from
      cont_out_&i
    where
      type = 2;
    ;
   quit;

   %&debug_local.put i=&i indata_i=&indata_i var_char=&var_char;


  %*** Find Numeric variables     ***;
  %let var_num=;

  proc sql noprint;
    select name into: var_num
    separated by ' '
    from
      cont_out_&i
    where
      type = 1;
    ;
   quit;

  %&debug_local.put i=&i indata_i=&indata_i var_num=&var_num;


  %*** In the next two data steps, replace '%if' with data statement
```

```
    'if' to ****;
    %*** decide to run the procedure if variables are present.  ***;
    %*** The 'symget' function reads a macro variable into a data step
    with no   ***;
    %*** macro resolution issues.  ***;

    %*** Character variables to print. Use 'if' instead of '%if'.  ***;
    data _null_;
      if symget('var_char') ne ' ' then
      do;
         call execute("proc print data = &indata_i(obs=&obs);");
         call execute(catx(' ','var',symget('var_char'),';'));
         call execute(
         'title "Print Character variables. data = &indata_i obs=&obs";');
         call execute('run;');
      end;
    run;


    %*** Numeric variables to summarize. Use 'if' instead of '%if'. ***;
    data _null_;
      if symget('var_num') ne ' ' then
      do;
         call execute("proc means data = &indata_i;");
         call execute(catx(' ','var',symget('var_num'),';'));
         call execute(
         ' title "Summarize Numeric variables. data = &indata_i";');
         call execute('run;');
      end;
    run;

  %*** End of %do while loop print or summarize     ***;
  %end;


  %***************************************
    Make local parameters global. The sysmacroname macro variable is an
automatic macro that contains the macro name.
    Remove code after development.
  ***************************************;
  data __z_macro_parms_global;
         set sashelp.vmacro;
         where scope = "&sysmacroname";
         call symputx(name,symget(name),'G');
  run;

%mend test_7_open;



%*** Open code     ***;
options mprint;
%test_7_open(indata=sashelp.cars sashelp.demographics  sashelp.comet
,obs=5 ,debug=yes);
```

7.1.3 View conversion of parameters to data

```
data =  parms
```

Obs	indata			indata_i	count_ indata	i
1	sashelp.cars	sashelp.demographics	sashelp.comet	sashelp.cars	3	1
2	sashelp.cars	sashelp.demographics	sashelp.comet	sashelp.demographics	3	2
3	sashelp.cars	sashelp.demographics	sashelp.comet	sashelp.comet	3	3

8 USE 'VVALUE' FUNCTION TO GET VALUES WHEN VARIABLE COULD BE CHARACTER OR NUMERIC

This example to take an input character or numeric date variable and output an iso8601(yyyy-mm-dd) character date variable uses the 'vvalue' function when the input variable could be character or numeric which avoids a SAS® error of using numeric syntax on a character variable and vice versa. The 'vvalue' function which returns the formatted character value of the original numeric or character variable is helpful when the type of the input variable is unknown. The 'vtype' function used on the original variable returns the variable type which replaces the '%if %then' statement. The macro code style is given first followed by the open code macro where key comments are highlighted. After submitting the open code macro and macro call for the first time, the macro can now be submitted by each data step or procedure section. View the code below:

```
%*************************************
 Example:(8)
  Use 'vvalue' SAS function to get values when variable could be character
or numeric.

  This macro takes an input character date variable or numeric date
variable and outputs iso8601(yyyy-mm-dd) character date variable

indata   = input data sets separated by a space. ie. sdtm.dm sdtm.ae
invar    = input character or numeric date variable
outdata  = output data set name
outvar   = name of output character variable to be created in
iso8601(yyyy-mm-dd) form

*************************************;
```

8.1.1 %*** Macro code style ***;

```
%macro test_8_macro(indata=,invar=,outdata=,outvar= );

  %local data_id var_num var_type;

  %*** Get variable type     ***;
  %let data_id  = %sysfunc(open(&indata));
  %let var_num  = %sysfunc(varnum(&data_id,&invar));
  %let var_type = %sysfunc(vartype(&data_id,&var_num));
  %let data_close = %sysfunc(close(&data_id));

  data &outdata;
    set &indata;

    %** Convert based on variable type macro variable **;
    %if       &var_type = C %then iso_num  = input(&invar,anydtdte.);
```

```
    %else %if &var_type = N %then iso_num  = &invar;
    ;

    &outvar = put(iso_num,e8601da.);

  run;

  proc print data = &outdata;
    title "data = &outdata";
  run;

%mend test_8_macro;


data test8;
  date_char = '01Jan2020';
  date_num =  '02Jan2020'd;
run;

options mprint;
%test_8_macro(indata=test8,invar=date_char,outdata=out8_char,outvar=iso_da
te );
%test_8_macro(indata=test8,invar=date_num,outdata=out8_num,outvar=iso_date
);
```

8.1.2 %*** Open code style ***;

```
%macro test_8_open(indata=,invar=,outdata=,outvar= );


  data &outdata;
    set &indata;

    %*** Copy the value into a new variable to remove any formatted value
    because vvalue ***;
    %*** function returns a formatted character value.     ***;
    var_copy= &invar;

    %*** Original value is now an unformatted character value ***;
    value_char = vvalue(var_copy);

    %*** Use the original variable with 'vtype' function to figure out the
    variable type ***;
    %*** If character date variable then convert to numeric date.   ***;
    %*** If numeric date variable then convert alpha numeric value to
     numeric value. ***;
    if  vtype(var_copy) = 'C' then
       iso_num   = input(value_char,?? anydtdte.);
    else if vtype(var_copy) = 'N' then
       iso_num   = input(value_char,?? best.);

    %** Convert numeric date to character date in yyyy-mm-dd format **;
    &outvar = put(iso_num,e8601da.);
  run;
```

```
  proc print data = &outdata;
    title "data = &outdata";
  run ;


    proc contents data=&outdata ;
  run;


  %*************************************
    Make local parameters global. The sysmacroname macro variable is an
automatic macro that contains the macro name.
    Remove code after development.
  *************************************;
  data __z_macro_parms_global;
         set sashelp.vmacro;
         where scope = "&sysmacroname";
         call symputx(name,symget(name),'G');
  run;

%mend test_8_open;


data test8;
  date_char = '01Jan2020';
  format date_num date9.;
  date_num =  '02Jan2020'd;
run;

options mprint;
%test_8_open(indata=test8,invar=date_char,outdata=out8_char,outvar=iso_dat
e );
%test_8_open(indata=test8,invar=date_num,outdata=out8_num,outvar=iso_date
);
```

9 WRITE MINI MACROS TO HANDLE DYNAMIC CODE SITUATIONS.

This example writes a mini macro to handle dynamic code situations to create a hybrid macro and open code style macro. This macro is a repeat of the macro used in the 'Use 'Data _null_' to derive a macro variable to replace '%if %then' and '%do loop' example (limiting character variable and writing numeric variable with a format), but now the macro code derivation for variable followed by format is placed into a mini-macro. To submit interactively, first, highlight and submit the 'var_names' mini-macro separately and the mini-macro becomes global. Second, submit the whole hybrid macro and macro call. The macro can now be submitted by each data step or procedure section. View the code below where key comments are highlighted:

```
  %*************************************
   Example:(9)
    Write mini macros to handle dynamic code situations. This macro is a
  repeat of example
     (limiting character variable and writing numeric variable with a
  format), but now the macro
    code derivation for variable followed by format is placed into a
  mini-macro. Highlight and submit the
    var_names mimi-macro separately and the mini-macro becomes global.
```

```
Macro limits the length of character variables based upon a boundary
parameter and prints numeric variables with the
associated format or if no format is associated then the numeric format
parameter input will be used.

indata               = input data set
char_format_boundary = Enter integer > 0 where character variable
                       length in the input data set above parameter
                       value will have the print out limited to the
                       parameter value.
num_format           = enter numeric format to use for print out when
                       numeric variable has no format
                        associated in the input data set

****************************************;

%*** Macro code **;
%macro test_9_open_hybrid(indata= ,char_format_boundary= ,num_format=
);

    %local var_format;


  %*************************************
   Macro derives a macro variable holding a string of variable names
   followed by a format.
   For each format, the macro limits the length of character variables
   based upon a boundary parameter
   and prints numeric variables with the associated format or if no
   format is associated then the numeric
   format parameter input will be used.

   macrovar = name of existing macro variable with value to be derived.
   System macro variables
    indata, char_format_boundary, and num_format defined in the parent
   macro.
  *************************************;
  %macro var_names(macrovar=);

     %local i data_id var_num var_name var_type var_length var_fmt
     local_var;
     %let data_id = %sysfunc(open(&indata));
     %let var_num = %sysfunc(attrn(&data_id,nvars));
     %let local_var=;

     %do i=1 %to &var_num;
        %let var_name   = %sysfunc(varname(&data_id,&i));
        %let var_type   = %sysfunc(vartype(&data_id,&i));
        %let var_length = %sysfunc(varlength(&data_id,&i));
        %let var_fmt    = %sysfunc(varformat(&data_id,&i));

        %put  i=&i &var_name &var_type &var_length &var_fmt;

        %if  &var_type = C   %then
        %do;
            %if &var_length > &char_format_boundary %then
        %let local_var = &local_var &var_name $&char_format_boundary..;
```

```
            %else
             %let local_var = &local_var &var_name $&var_length..;
        %end;
        %else %if  &var_type = N   %then
        %do;
            %if %nrbquote(&var_fmt) = %str( ) %then
              %let local_var = &local_var &var_name &num_format..;
            %else
              %let local_var = &local_var &var_name &var_fmt.;
        %end;

        %put i=&i local_var= &local_var;
     %end;

     %let &macrovar = &local_var;

     %let close   = %sysfunc(close(&data_id));

  %mend var_names;


  %*** Get macro variables to hold the character and numeric variables
  with adjusted formats      ***;
  %let var_format=;
  %var_names(macrovar=var_format);


  proc print data = &indata;
      format &var_format;
      title "data = &indata";
  run;


  %****************************************
    Make local parameters global. The sysmacroname macro variable is an
    automatic macro that contains the macro name.
    Remove code after development.
  ****************************************;
   data __z_macro_parms_global;
         set sashelp.vmacro;
         where scope = "&sysmacroname";
         call symputx(name,symget(name),'G');
  run;


%mend test_9_open_hybrid;


options mprint;

%*** Data set series of numeric x variables      ***;
data test9;

  format x_date date9.;
  x_date = date();

  x1=5;
```

```
  x2=3;
  x3=-1;

  name = 'charlie';
  text = repeat('a',200);
  output;
run;

%*** Macro code     ***;
%test_9_open_hybrid(indata=test9 ,char_format_boundary=15
,num_format=best );
```

10 USE 'RESOLVE' FUNCTION TO CALL MINI MACROS

This example uses the 'resolve' function to call a mini macro to handle dynamic code situations to create a hybrid nested macro and open code style macro. This macro is a repeat of the macro in the 'Use 'data _null_' to create reusable macro variables within a '%do' loop' example (prints the character variables and summarizes the numeric variables with proc means), but now the macro code derivation for the character or numeric variables is placed into a mini-macro and called by the resolve function. Afterwards, a 'data _null_' with 'call execute' is used instead of a macro loop to decide to print character variables or summarize numeric data. To submit interactively, first highlight and submit the 'var_names_type' mini-macro separately and the mini-macro becomes global. Second submit the whole hybrid macro and macro call. The macro can now be submitted by each data step or procedure section. To understand the flow of the hybrid macro, the print out in section '10.1 View variables in the data' following the macro shows how the macro input data parameter was converted to a data set of data set names and variable names. View the printout code below where key comments and code are highlighted:

```
%************************************
 Example:(10)
  Rewrite example (prints the character variables and summarizes the
  numeric variables with proc means)
  using resolve function to bring in the variables by type by calling a
  macro function which contains
  the macro looping.  This technique leads to a more open code style
  program.

 Macro reads in each data set and prints the character variables and
 summarizes the numeric variables with proc means.

indata   = input data sets separated by a space. ie. sdtm.dm sdtm.ae
obs      = number of observations to print
debug    = View test prints. Yes/NO

************************************;


%macro test_10_open_hybrid(indata= ,obs= ,debug=  );

 %local debug_local;

   %************************************
   Macro function which returns the variable names for the variable
   type selected
     indata      = input data
```

```
   type        = variable type.  C or N
   debug_macro = View macro resolution. Enter blank or *.
****************************************;
%macro var_names_type(indata= , type= ,debug_macro=);

   %local i data_id var_num var_name var_type local_var;

   %let data_id   = %sysfunc(open(&indata));
   %let var_num   = %sysfunc(attrn(&data_id,nvars));
   %let local_var =;

   %do i=1 %to &var_num;
      %let var_name   = %sysfunc(varname(&data_id,&i));
      %let var_type   = %sysfunc(vartype(&data_id,&i));

      %if   %qupcase(&var_type) = %qupcase(&type)   %then
       %let local_var = &local_var &var_name;
      %&debug_macro.put indata=&indata type=&type var_name=&var_name
         var_type=&var_type   local_var = &local_var;
   %end;

   %let close   = %sysfunc(close(&data_id));

   &local_var

%mend var_names_type;


%*** Puts data set parameter into a data set with a row for each data
set.   Derive local debug parameter.   ***;
data parms;
  %*** Derive local debug to comment out code    ***;
  call symputx('debug_local'
              ,ifc(lowcase(symget('debug'))='yes','','*'),'L');

  length indata indata_i $%length(&indata);
  indata = symget('indata');
  count_indata = countw(indata,' ');

  do i=1 to count_indata;
    indata_i = scan(indata,i,' ');
    output;
  end;
run;


%*** Cycle through the input data sets to get the variable names of
each variable type     ***;
data var_types;
  set parms;
  length vars_char vars_num $10000;

  vars_char = resolve(catt('%var_names_type(indata=', indata_i
              ,',type=C ,debug_macro=',"&debug_local",')'));
  vars_num  = resolve(catt('%var_names_type(indata=', indata_i
              ,',type=N ,debug_macro=',"&debug_local",')'));
```

```
  run;


  %*** Create call code to decide to call print for characters
  variables and proc means for numeric variables    ***;
  data _null_;
    set var_types;

    %***  If character variables are present then proc print with '%if
    %then' not used.  ***;
    if vars_char ne ' ' then
    do;
       call execute(catx(' ','proc print data ='
                    ,indata_i,"(obs=&obs);"));
       call execute(catx(' ','var',vars_char,';'));
       call execute(catx(' ','title "Print Character variables. data='
                   ,indata_i,"obs=&obs",'";'));
       call execute('run;');
    end;

    %*** If numeric variables are present then proc means with '%if
     %then' not used.  ***;
    if vars_num ne ' ' then
    do;
      call execute(catx(' ','proc means data =',indata_i,';'));
      call execute(catx(' ','var',vars_num,';'));
      call execute(catx(' ','title "Summarize Numeric variables. data='
                          ,indata_i,'";'));
      call execute('run;');
    end;
  run;


  %*************************************
    Make local parameters global. The sysmacroname macro variable is an
automatic macro that contains the macro name.
    Remove code after development.
  **************************************;
  data __z_macro_parms_global;
         set sashelp.vmacro;
         where scope = "&sysmacroname";
         call symputx(name,symget(name),'G');
  run;

%mend test_10_open_hybrid;



%*** Open code     ***;
options mprint;
%test_10_open_hybrid(indata=sashelp.cars sashelp.demographics
sashelp.comet ,obs=5 ,debug=Yes);
```

10.1 VIEW VARIABLES IN THE DATA

```
data = var_types
```

```
indata_i              i  vars_char                              vars_num
sashelp.cars          1  Make Model Type Origin DriveTrain      MSRP Invoice EngineSize Cylinders
                                                                Horsepower MPG_City MPG_Highway
                                                                Weight Wheelbase Length
sashelp.demographics  2  NAME ISONAME region                    CONT ID ISO pop popAGR popUrban
totalFR
                                                                AdolescentFPpct AdolescentFPyear
                                                                AdultLiteracypct MaleSchoolpct
                                                                FemaleSchoolpct GNI PopPovertypct
                                                                PopPovertyYear
sashelp.comet         3                                        Dose Rat Sample Length
```

11 USING THE 'DOSUBL' FUNCTION TO BRING IN MACRO VALUES BASED ON MINI MACROS

This example uses the 'dosubl' function to call a mini macro to handle dynamic code situations to create a hybrid nested macro and open code style macro. This macro is a repeat of the macro in the 'Use 'data _null_' to create reusable macro variables within a '%do' loop' example (prints the character variables and summarizes the numeric variables with proc means), but now the macro code derivation for the character or numeric variables is placed into a mini-macro called by the 'dosubl' function. A unique feature of the 'dosubl' function displayed in this example is the function can call a macro containing data steps and procedures and return a macro variable that can be returned to a data step. Please consult SAS® documentation for complete details as the 'dosubl' function can be complicated to implement. Afterwards, a 'data _null_' with a 'call execute' is used instead of a macro loop to decide to print character variables or summarize numeric data. To submit interactively, first highlight and submit the 'var_names_type_ds' mini-macro separately and the mini-macro becomes global. Second submit the whole hybrid macro and macro call. The macro can now be submitted by each data step or procedure section. View the code below where key comments and code are highlighted:

```
%***************************************
 Example:(11)
  Rewrite case (prints the character variables and summarizes the
  numeric variables)
  using the dosubl function to bring in the variables by type by
  calling a macro that uses proc sql and
  a data step.  The dosubl is new in version 9 SAS.  Please read the
  dosubl documentation for a fuller
  understanding of this complex function.  This technique leads to a
  more open code style program.

 Macro reads in each data set and prints the character variables and
 summarizes the numeric variables with proc means.

 indata   = input data sets separated by a space. ie. sdtm.dm sdtm.ae
 obs      = number of observations to print
 debug    = View test prints. Yes/NO
****************************************;

%*** Macro and open code style    ***;
%macro test_11_open(indata= ,obs= ,debug=  );

 %*** Set up     ***;
 %** This macro variable is updated by the 'dosubl' function. **;
 %global var_list;
 %local debug_local;
```

```
%*************************************
   Macro function which returns macro global variable var_list with the
   variable names for the type selected
     indata      = input data
     type        = variable type. C for character  or N for numeric
     debug_macro = View macro resolution. Enter blank or *.
  *************************************;
  %macro var_names_type_ds(indata= , type=  );

    %***  Get metadata    ***;
    proc contents data = &indata out = cont_out noprint;
    run;

    %let sql_var_list=;


    %*** Find variables     ***;
    proc sql noprint;
      select name into: sql_var_list
      separated by ' '
      from
        cont_out
      where
        type = %sysfunc(ifc(%qupcase(&type)=C,2,1));
      ;
     quit;


    %*** Make variable global to allow pass through to the data set***;
     data _null_;
        call symputx('var_list',symget('sql_var_list'),'G');
     run;

  %mend var_names_type_ds;


  %*** Puts data set parameter into a data set with a row for each data
  set.   Derive local debug parameter.   ***;
  data parms ;
    %*** Derive local debug to comment out code    ***;
    call symputx('debug_local'
                ,ifc(lowcase(symget('debug'))='yes','','*'),'L');

    length indata indata_i $%length(&indata) ;
    indata = symget('indata');
    count_indata  = countw(indata,' ');

    do i=1 to count_indata;
      indata_i = scan(indata,i,' ');
      output;
    end;
  run;


  %*************************************
```

```
  Cycle through the input data sets to get the variable names of each
  variable type.
  The dosubl functions returns the value of existing global macro
  variable var_list which is updated by the
  var_names_type_ds macro.
****************************************;
 data var_types;
  set parms;
  length vars_char vars_num $10000;

  rc = dosubl(catt('%var_names_type_ds(indata=', indata_i ,
                   ',type=C)'));
  vars_char = symget('var_list');

  rc = dosubl(catt('%var_names_type_ds(indata=', indata_i ,
                   ',type=N)'));
  vars_num  =  symget('var_list');
run;


%*** Create call code to decide to call print for characters
variables and proc means for numeric variables    ***;
data _null_;
  set var_types;

  %***  If character variables are present then proc print with '%if'
  avoided.   ***;
  if vars_char ne ' ' then
  do;
     call execute(catx(' ','proc print data ='
                   ,indata_i,"(obs=&obs);"));
     call execute(catx(' ','var',vars_char,';'));
     call execute(catx(' ',
'title "Print Character variables. data=',indata_i,"obs=&obs",'";'));
     call execute('run;');
  end;

  %*** If numeric variables are present then proc means with '%if'
  avoided.  ***;
  if vars_num ne ' ' then
  do;
     call execute(catx(' ','proc means data =',indata_i,';'));
     call execute(catx(' ','var',vars_num,';'));
     call execute(catx(' ',
    'title "Summarize Numeric variables. data=',indata_i,'";'));
     call execute('run;');
  end;
run;


%****************************************
  Make local parameters global. The sysmacroname macro variable is an
  automatic macro that contains the macro name.
  Remove code after development.
****************************************;
data __z_macro_parms_global;
      set sashelp.vmacro;
```

```
        where scope = "&sysmacroname";
        call symputx(name,symget(name),'G');
  run;


%mend test_11_open;



%*** Open code     ***;
options mprint;

%test_11_open(indata=sashelp.cars sashelp.electric ,obs=10
,debug=Yes);

  %test_11_open(indata=sashelp.cars sashelp.demographics  sashelp.comet
,obs=5 ,debug=No);
```

CONCLUSION

In conclusion, a series of examples were displayed to convert a nested style macro to an open code style macro to facilitate the development, updates, and reading of macro code. As the department macro library gets larger and more complicated, faster macro updates can lead to much improved productivity. Instead of submitting a whole macro every time an update is needed, just a section of code needs to be highlighted and submitted because the macro parameters are now global and open code statements are used. An open code style macro is no longer a large black box where the whole macro must be submitted to get results. With the open code macro style, there is a cost in setting up and understanding the code because more code or non-tradition coding approaches are used. In my opinion, the primary benefit of faster updates from the open code style far outweighs the costs. Hopefully, this paper will inspire you to try the open code style macro approach and to invent new open code style macro techniques.

CONTACT INFORMATION

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

Frederick Cieri
fecieri@yahoo.com

Keywords:Frederick Cieri

SAS® Macro Updates