Paper CT05

P_MACRO, Parameters Extraction from Macros to SAS Dataset

Eswara Satyanarayana Gunisetti, Cytel Inc. UK, London, United Kingdom

ABSTRACT

Macro parameters are variables whose values initialized when we invoke the macro. Parameters give macros flexibility, and including PARMBUFF option gives more flexibility by allowing a variable number of parameters.

In this paper we learn how to extract macro parameters, along with their default values from a group of macros in a folder, to a SAS dataset, to easily view all parameters in one location. With the P_MACRO resultant dataset, it is easy to automate the macro call by including all defined macro parameters.

We will cover the importance of standard programming practices to ease the extraction of macro parameters, and approaches to follow if nested macros are present or if no parameters are defined. We will discuss how we retain the macro parameter order as it was defined within the macro while saving to a dataset. We will also discuss use cases to see how this could help our programmers.

INTRODUCTION

With the use of P_MACRO, we extract macro parameters from a group of macro program files and store them to a SAS dataset. Along with parameters, macro program also extracts default values if defined already, retains the order of parameter in the way it was defined. This way it will be easy for a programmer to reference them at one place within SAS.

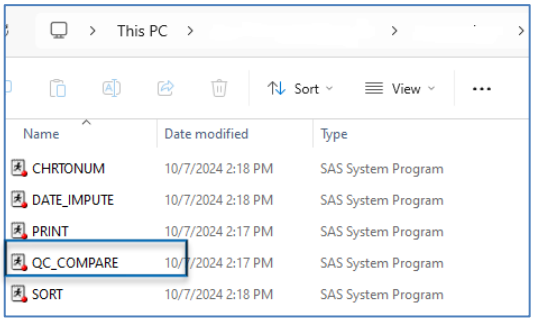
Using stored information in the resultant dataset we produce an automated macro call with a complete list of parameters for each individual macro.

In the below sections I discuss about what the macro does, why it is needed, how to program and limitations.

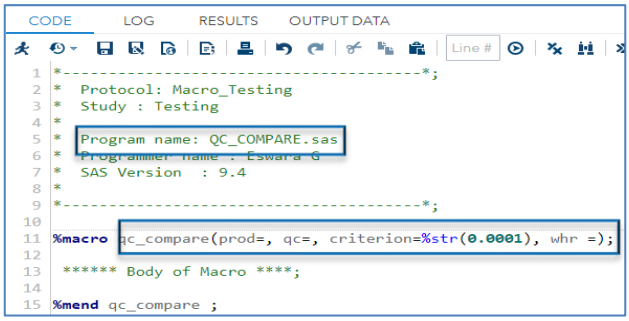
WHAT DOES P_MACRO DO?

1. Extracts parameters from group of macro programs to SAS dataset
2. Extracts Default values along with parameters if defined already
3. Retains order of parameters
4. Prioritizes main macro information over nested macros
5. Provides a common text for the macros programs without parameters
6. Generates automated macro call

SAS Macro program files in a folder



SAS Program named QC_COMPARE



SAS DATASET named P_MACRO

Table: WORK.P_MACRO | View: Column labels | Filter: (none)

Total rows: 4 Total columns: 4

	Macro Name	Macro Parameter Name	Macro Parameter Default Value	Parameters Order as defined in Macro
1	QC_COMPARE	prod		1
2	QC_COMPARE	qc		2
3	QC_COMPARE	criterion	%str(0.001)	3
4	QC_COMPARE	whr		4

SAS Macro program files in a folder: Represents the group of macro programs in a folder. Macro will consider all the macro programs in a folder and extract the parameters from each file and save those details to a SAS dataset.

SAS Program named QC_COMPARE: Represents QC_COMPARE program which is one of the files in the folder. Highlighted are the section from where macro will extract the details.

SAS DATASET named P_MACRO: Represents the resultant dataset from the macro. From each macro program file, macro will extract macro name, macro parameters, their default values if defined already and the order of them as defined and save the details to a SAS dataset.

WHY IS NEEDED?

- ✓ As a programmer we work on many programs every day, sometimes we may take over other programmers code to do an adhoc request or fix a comment on their absence. In the programs, we include macro calls of the macros specific to study or company level validated standard macros which were there to reduce the amount of time we spent on programming and increase the quality.
- ✓ Usually, these macro programs saved in a central location and reference them in setup files, auto execution files to use when needed.
- ✓ In SAS macros, we have flexibility that we can exclude some of the parameters while invocation and still the macro executes fine if there is no dependency.

Below is an example where we can limit the parameters and still macro executes fine.

```
%macro sort(DSN =%str(sashelp.cars), OUT =, OPT = , WHR =, VAR = );
    proc sort data = &dsn out = &out. &opt.;
         &whr.;
        by &var.;
    run;
%mend;
```

```
CALL 1: %sort(DSN =%str(sashelp.cars), OUT =CLASS, OPT =NODUPRECS, WHR =%str(where ORIGIN="Asia"), VAR = MAKE MODEL TYPE);
CALL 2: %sort(OUT =CLASS, WHR =%str(where ORIGIN="Asia"), VAR = MAKE MODEL TYPE);
CALL 3: %sort(OUT =CLASS, VAR = MAKE MODEL TYPE);
```

- ✓ When we do not include the complete list of parameters in a call, it will be difficult for the programmers to decide on adding parameters when an update is needed.
- ✓ To do the update programmer should already be aware of the complete list of parameters or use some of debugging options to check out for the list of parameters used, or manually open the macro code and check.
- ✓ We do not have an automated facility within SAS where we can check complete list of macro parameters defined for a group of macros.
- ✓ P_MACRO has been programmed to help programmers to refer to the complete list of parameters defined within a macro program **being in SAS environment itself**. With this, it will be easy to get hold of the complete list of parameters defined along with their default values and position/order.

HOW TO PROGRAM?

1. Read macro program files to SAS
2. Extract macro name
3. Determine macro start & End points
4. Handle Nested macros
5. Handle macro programs with no parameters
6. Retain macro parameters position
7. Bring out Default values
8. Generate Macro call

Step 1: Read macro programs tiles to SAS

✓ INFILE statement with the support of some SAS options would help us to read SAS programs into SAS.

Below is the piece of code for your reference.

```
filename m_param "/&path./macro$/*.sas";
data m_param;
    infile m_param lrecl = 32760 truncover;
    input description $32760.;
run;
```

'*.sas'* in the above program step makes sure only SAS programs are read from the folder.

Step 2: Extract macro name

✓ This is the important step in P_MACRO programming, it guides us to keep right macro information in case of multiple macro calls within the same macro program.
✓ Good programming practices (GPP) or Standard Programming practices help to do program in identical way across different skill sets and project/study levels.
✓ For automation it is important to have a standard approach. Without standard approach none of the programs will look identical and extracting similar information from more than one file will be extremely difficult.

Below screenshot is an example of a programming header. Here PROGRAM NAME is the key text used to provide the name.

```
1  *------------------------------------------------------------------------*;
2  *  Protocol: Macro_Testing
3  *  Study: Testing
4  *
5  *  Program name: QC_COMPARE.sas
6  *  Programmer Name: Eswara G
7  *
8  *  SAS Version: 9.4
9  *------------------------------------------------------------------------*;
```

Following the same approach in all the macro programs eases the extraction process. If it is not followed and simply given a random text as 'Name', 'Macro', or no header portion at all, would make it difficult to identify.

*Make sure the program name and macro name same.

Step 3: Determine macro Start & End points

✓ By syntax macro statement must start with '%macro' keyword followed by a macro name, parameters defined within '()' and statement ends with '**;**'. I have highlighted these sections in the screenshot below for easy reference.

CODE LOG RESULTS

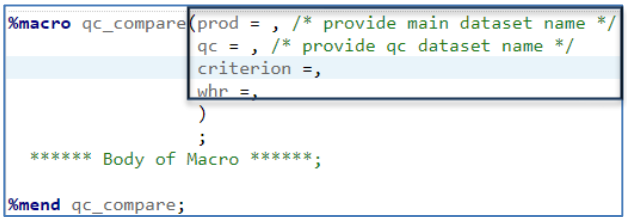
```
1  *----------------------------------------*;
2  *  Protocol: Macro_Testing
3  *  Study : Testing
4  *
5  *  Program name: QC_COMPARE.sas
6  *  Programmer name : Eswara G
7  *  SAS Version  : 9.4
8  *
9  *----------------------------------------*;
10
11 %macro qc_compare(prod=, qc=, criterion=, whr =);
12
13  ****** Body of Macro ****;
14
15 %mend qc_compare ;
16
```

Use SAS functions to identify these key text to locate the macro start position, parameters list and their end position.

Important points to look for:

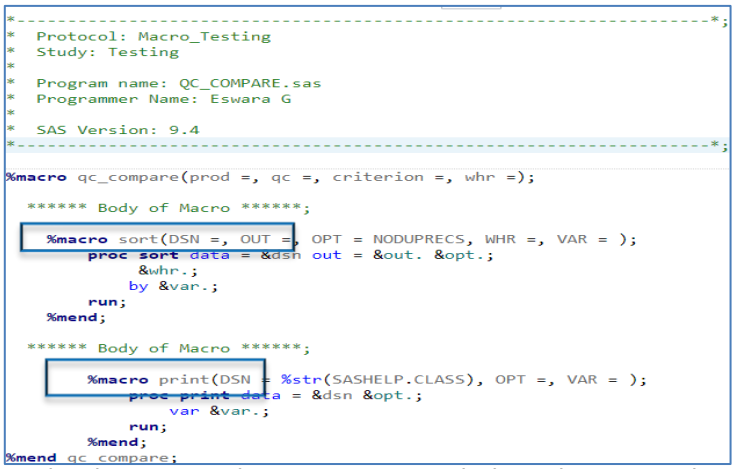
- Identify whether macro call has parameters or no parameters included!
- Parameters provided in a single line or multiple lines.
- Are there any commented text included in between macro parameters?

It is important to write our program in a way that it will take care of all such scenarios and extract only the necessary information.
Below is the screenshot with macro parameters in multiple lines and commented text in between them.



Step 4: Handle Nested macros

- ✓ Nested macros are macros written within macros. When nested macros are in place, it is important to extract the right macro information and exclude all that is not necessary.
- ✓ Macro name that extracted from program header help us to match the macro name that comes from %macro statement and keep only the information related to that.

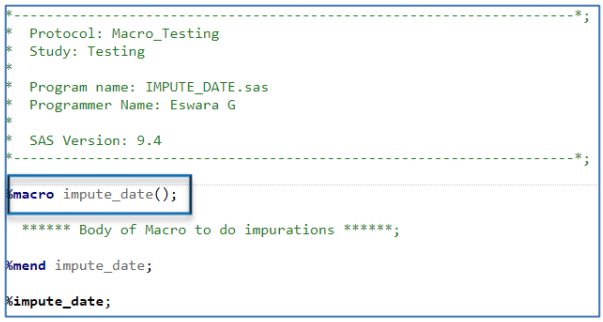


* In the above screenshot, QC_COMPARE is the main macro and SORT, PRINT are the nested macros.

Step 5: Handle macro programs with no parameters

- ✓ There are cases where parameters won't be included while defining the macro with %macro statement. In such cases it is important to identify and provide a common text like '*No parameters defined*' to give programmer an idea of how the macro was defined.

Following screenshot is an example of one such macro.



Step 6: Retain macro parameters position

- ✓ In case of Key word parameters it's not important to retain the position but for Positional parameters it's required to retain their position for invocation.
- ✓ Retaining the position will help us to generate the automated macro call in a similar way as it was defined.
- ✓ Retain statement, simple sequence statements help to retain their position at each macro level.

Step 7: Bring out Default values

- ✓ Saving default values along with macro parameters give programmer an idea of the values that were assigned.
- ✓ This helps to decide whether to include or skip while invocation.

Important points to look for:

i. Macro functions such as %STR, %NRSTR, %BQUOTE, etc.,
ii. Special characters such as '$%;&'
iii. Commented text if included.

* It is important to mask these special characters or remove commented text while extracting default values.

Step 8: Generate Macro call

- ✓ Once all the details about a macro file is saved to a SAS dataset, it is easy to use the information and generate a macro call.
- ✓ A simple retain statement helps to keep all macro parameters together and CALL SYMPUTX stores the value into a macro variable to resolve when and where needed.

Following screenshot has the piece of code that could support to automate and print the macro call to log file using macro variable named similar to macro name. I have highlighted the main piece of code for easy in below screenshots for easy reference.

```
data autocall;
    length autocall $32000;
    set p_macro;
    by macro_name paramn;
    retain autocall;
    if param_d_val ne "" then m_parameter = strip(m_parameter)||"="||strip(param_d_val);
        else m_parameter = strip(m_parameter)||"=";
    if first.macro_name then autocall = m_parameter;
        else autocall = strip(autocall)||","||strip(m_parameter);
    if last.macro_name then autocall = 'macro('||strip(autocall)||");";
    if upcase(m_parameter) = "NO PARAMETERS DEFINED" then autocall = m_parameter;
    if last.macro_name; * and autocall ne "";
run;

data _null_;
    set autocall;
    by macro_name paramn;
    call symputx(scan(macro_name,1,"."),autocall);
    *put _all_;
run;

%put %bquote(&qc_compare.);
```

In the above screenshot, &QC_COMPARE macro variable holds the macro call details and prints to log when resolved with %PUT statement as shown in below screenshot.

```
93
94          %put %bquote(&qc_compare.);
macro(prod=,qc=,criterion=0.001,whr=);
```

Limitations

1. **Default values provided with %LET statement**
   a) %Let can be used to conditionally check and assign a default value.
   b) In this case since no value is provided initially, no default value will be extracted.

2. **Adding PARMBUFF**
   a) Using **PARMBUFF** option and SYSPBUFF, you can define a macro that accepts a varying number of parameters at each invocation.
   b) SYSPBUFF resolves to the text supplied as parameter values in the invocation.
   c) Providing a common text for all the macro files that use PARMBUFF would help programmer to decide.

```
%macro printz/parmbuff;
   %let num=1;
   %let dsname=%scan(&syspbuff,&num);
   %do %while(&dsname ne);
      proc print data=&dsname;
      run;
      %let num=%eval(&num+1);
      %let dsname=%scan(&syspbuff,&num);
   %end;
%mend printz;
```

```
%printz(purple,red,blue,teal)
```

CONCLUSION

- ✓ In SASHELP library we have a dataset named VCOLUMN which holds detailed information about metadata of libraries, datasets and variables present for that SAS session. This will help programmers in many ways to identify/query some of the important information about the datasets/variables for active session.

- ✓ Like VCOLUMN dataset, the resultant dataset generated through P_MACRO would support programmers to find list of macros within a folder, their entire list of parameters in defined order along with default values at one place within SAS.

SASHELP.VCOLUMN

	libname	memname	name	varnum
1	SASHELP	CLASS	Name	1
2	SASHELP	CLASS	Sex	2
3	SASHELP	CLASS	Age	3
4	SASHELP	CLASS	Height	4
5	SASHELP	CLASS	Weight	5
6	SASHELP	CLASSFIT	Name	1
7	SASHELP	CLASSFIT	Sex	2
8	SASHELP	CLASSFIT	Age	3
9	SASHELP	CLASSFIT	Height	4
10	SASHELP	CLASSFIT	Weight	5
11	SASHELP	CLASSFIT	predict	6

Final Dataset from P_MACRO programming

Table: WORK.P_MACRO | View: Column labels | Filter: (none)

Total rows: 4 Total columns: 4

	Macro Name	Macro Parameter Name	Macro Parameter Default Value	Parameters Order as defined in Macro
1	QC_COMPARE	PROD		1
2	QC_COMPARE	QC		2
3	QC_COMPARE	CRITERION	0.0001	3
4	QC_COMPARE	WHR		4

- ✓ Generating an automated macro call using the resultant dataset would support programmer to have entire parameters list handy and use as needed.

REFERENCES

- Carpenter, A. (2017). Building Intelligent Macros: Using Metadata Functions with the SAS® Macro Language.
- SAS Macro Programming Made Easy, Third Edition, Book by Michele M. Burlew

CONTACT INFORMATION

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

Author Name: Eswara Satyanarayana Gunisetti
Company: Cytel Inc. UK
Address: Aldwych House, 71-91 Aldwych, London WC2B 4HN
Work Phone: 07586991314
Email: eswara.gunisetti@cytel.com
Website: www.cytel.com