Paper SM02

Enhancing SAS Macro Governance with %_mac_usage

Guido Wendland, Cytel Inc., Germany

ABSTRACT

Knowing which macros are used in a program or study can be useful for various reasons:

- To populate the program header
- To identify the need for program updates
- To increase consistency across programs
- To identify dependencies between macros
- To identify potential issues or estimate the impact of macro updates
- Etc.

The macro %_mac_usage provides a detailed Excel report of all macro invocations. This presentation offers insights into the involved techniques to derive and report this information.

INTRODUCTION

Efficient SAS macro management is essential in clinical programming, where expanding code libraries and complex reporting requirements are common. The macro %_mac_usage addresses the challenge of tracking macro usage across multiple folders and programs. It offers insights by classifying macro invocations based on analysis type, source, and frequency.

Standard SAS macros help automate and standardize programming tasks for regulatory submissions and study reporting. As code libraries and teams grow, organizations need better tools to monitor macro usage, reduce redundancies, and maintain consistent standards. The %_mac_usage utility scans specified folders and subfolders, providing structured visibility across studies. Its output supports standardization, audit readiness, and code optimization.
This paper details the purpose, functionality, programming challenges, and reporting capabilities of %_mac_usage.

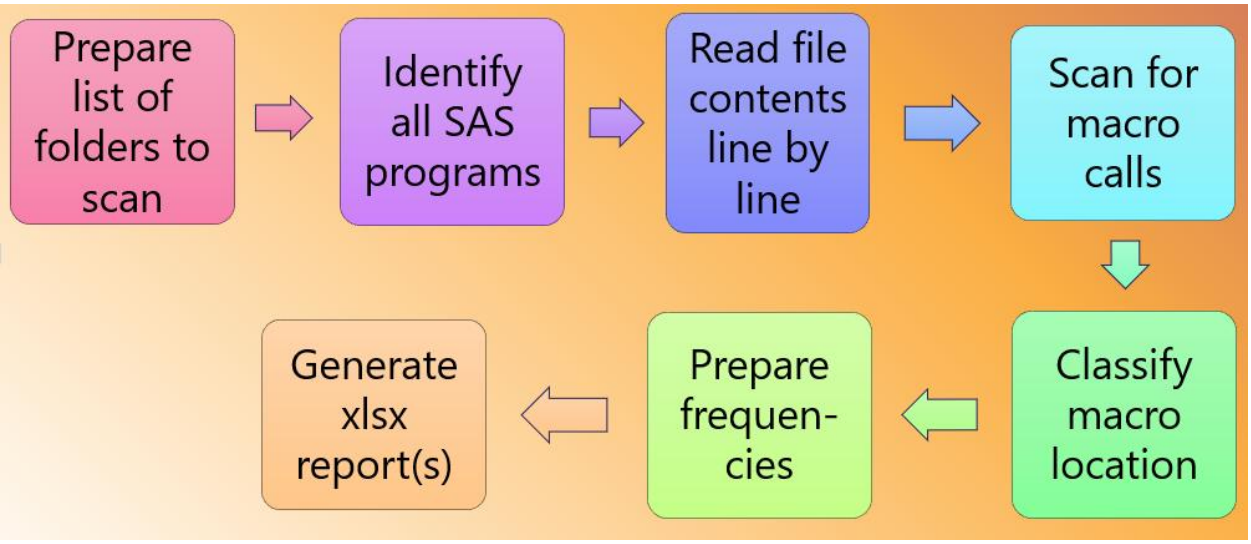
MACRO DESIGN

The main functionality of %_mac_usage relies on two key parameters:

- **ROOT:** Specifies the location(s) of SAS programs to be scanned.
- **OXLSXPATH:** Defines the output path and filename for the Excel report.

Additional parameters refine the scope or output, such as selecting particular programs, macros, or adding a timestamp to the output file.

The following diagram illustrates the various building blocks of the macro:



Parameters should be selected based on the project's needs—broad scans for audits or focused scans for targeted reviews. Adjustments may be required if folder structures or naming conventions change. Although optimized for typical clinical reporting setups, the macro is flexible for various environments.

KEY PROGRAMMING CHALLENGES

In order to prepare the list of folders to scan it is necessary to split multiple root paths which are in this case separated by | and generate a macro variable for each part:

```
%let _i=1;
%do %until (%qscan(%bquote(&ROOT.),&_i.,%str(|)) = %str( )) ;
    %let _root&_i.=%qscan(%bquote(&root.),&_i.,%str(|));
        %put &_i.) Root path is &&_root&_i.;
    %let _i=%eval(&_i.+1);
%end;
```

This is probably one of the most often used techniques in macro programming. The parameter ROOT is transformed into multiple macro variables, e.g. if there are three root paths:

```
%let ROOT=path1|path2|path3;
→
&_root1=path1;
&_root2=path2;
&_root3=path3;
```

It should be noted that the macro variables are represented as **&&_root&_i**, where **&_i**. is the indicator for the ith root folder. SAS processes ampersands (&) in two passes: During the first pass the double ampersand (&&) is resolved to a single ampersand (&) and &_i. is resolved to its numeric value in the loop. Then in the second pass &_root1, &_root2. etc are resolved to the defined paths.

The same technique is also applied to split the list of macro locations.

Now that we have defined all folders to scan through, we need to identify all SAS programs contained in them and their subfolder. The easiest way to implement it is via the pipe command which is a speciality in SAS that runs external commands and captures the output as a data stream:

```
filename sasfiles&_i. pipe "dir /s /b &&_root&_i.\*.sas";

data sasprograms&_i;
    infile sasfiles&_i. truncover;
    input fullpath $300.;
    fname=scan(fullpath, -1, "/\");
    …
run;
```

The statement **"dir /s /b &&_root&_i.*.sas"** is a Windows command that lists all *.sas files under the root folder path &&_root&_i. The option /s ensures that subdirectories are considered and the option /b ensures that the bare format without any additional information like header, file sizes etc is stored.
In the following data step the SAS dataset **sasprograms&_i**. is generated which uses the previously defined filename in an infile statement. The input statement parses each line to the variable fullpath, allowing file paths up to 300 characters.

However, in many organizations the pipe command is unavailable due to security policies or system configurations. You can check this by submitting `%put %sysfunc(getoption(xcmd));`. If the result is NOXCMD, commands like pipe are not permitted. In that case you would use alternative methods like the data step functions dopen/dread and scan through the folders and subfolders iteratively.

Now assume that you have generated this list for all root folders and combined all in one dataset called sasprograms. In the next step we need to extract the SAS code of each program line by line. This can be implemented with the code below:

```
data sasprograms_01;
    length fullpath $300 progcode $500;
    set sasprograms;
    infile dummy filevar=fullpath end=eof truncover lrecl=500;
    input progcode $char500.;
    if not eof then output;
run;
```

The **infile** statement configures how the code will be read from the files listed in variable **fullpath**:
The **filevar=fullpath** statement tells SAS to dynamically open the SAS program files in the variable fullpath for each record in dataset **sasprograms**. The input statement then reads each line from the opened SAS program file into the variable **progcode**.

Now we have the complete code in one dataset and can extract macro invocations from the code:

```
data macro_calls;
  set _sasprograms_01;
   length macro_call $50;
   pos = index(progcode, '%');
   line = progcode;
   do while (pos > 0 and pos < length(line));
      rest = substr(line, pos + 1);
      macro_call = scan(rest, 1, ' ();,=%.''"');
      if not (lowcase(macro_call) in
         ('if','then','do',...)) and not missing(macro_call) then do;
         output;
      end;
      if index(substr(line, pos + 1), '%') > 0 then
         pos = pos + index(substr(line, pos + 1), '%');
      else
      pos = 0;
   end;
run;
```

The substring after each occurrence of "%" is read into variable **rest** and subsequently cut off after the first word boundary in variable **macro_call** that contains the name of the macro. Macro names that are not contained in an exclusion list (containing elements of the macro language) are outputted. The code also ensures that multiple macro invocations in the same line are captured.

Subsequently we have the list of all macro invocations per program and can classify them by location using similar techniques as shown above.

REPORTS

Results are presented in xlsx reports using filters for columns allowing the user to select programs, macros or specific macro locations

Proje	Study	Request	Macro	Macro classification	Progam name	#	Macro location
d419	common	adam	%c_list	mlibrary	trace_lineage	3	root/cdar/common/shared_programs/work_mlibrary/program
xxx	common	raw_adam	%_odsoutput	mlibrary	adam_codelist_issues	1	root/cdar/common/shared_programs/mlibrary/program
xxx	common	raw_adam	%_overlay_v0100	lib01	adam_codelist_issues	11	root/cdar/xxx/common/code_library/raw_adam/macro
xxx	common	raw_adam	%_read_excel	mlibrary	adam_codelist_issues	3	root/cdar/common/shared_programs/mlibrary/program
xxx	common	raw_adam	%adam_codelist_issues	Inner Definition	adam_codelist_issues	1	
xxx	common	raw_adam	%c_list	mlibrary	adam_codelist_issues	6	root/cdar/common/shared_programs/mlibrary/program
xxx	common	raw_adam	%c_varlist	mlibrary	adam_codelist_issues	13	root/cdar/common/shared_programs/mlibrary/program
xxx	common	raw_adam	%_dropmiss	mlibrary	azsol_anno	1	root/cdar/common/shared_programs/mlibrary/program
xxx	common	raw_adam	%_odsoutput	mlibrary	azsol_anno	1	root/cdar/common/shared_programs/mlibrary/program
xxx	common	raw_adam	%_read_excel	mlibrary	azsol_anno	7	root/cdar/common/shared_programs/mlibrary/program
xxx	common	raw_adam	%azsol_anno	Inner Definition	azsol_anno	1	
xxx	common	raw_adam	%_frq	mlibrary	impl_progress	10	root/cdar/common/shared_programs/mlibrary/program
xxx	common	raw_adam	%_impl_progress	Inner Definition	impl_progress	1	
xxx	common	raw_adam	%_odsoutput	mlibrary	impl_progress	1	root/cdar/common/shared_programs/mlibrary/program
xxx	common	raw_adam	%_read_excel	mlibrary	impl_progress	1	root/cdar/common/shared_programs/mlibrary/program
xxx	common	xcr	%_convert	mlibrary	cr02	3	root/cdar/common/shared_programs/mlibrary/program
xxx	common	xcr	%_f_comparelist	mlibrary	cr02	2	root/cdar/common/shared_programs/mlibrary/program
xxx	common	xcr	%_f_maxlength	mlibrary	cr02	1	root/cdar/common/shared_programs/mlibrary/program
xxx	common	xcr	%_frq	mlibrary	cr02	1	root/cdar/common/shared_programs/mlibrary/program
xxx	common	xcr	%_odsoutput	mlibrary	cr02	1	root/cdar/common/shared_programs/mlibrary/program
xxx	common	xcr	%_read_excel_v0103	lib01	cr02	1	root/cdar/xxx/common/code_library/raw_adam/macro
xxx	common	xcr	%c_varlist	mlibrary	cr02	1	root/cdar/common/shared_programs/mlibrary/program
xxx	common	xcr	%cr_overview	Inner Definition	cr02	1	

As illustrated in the first report macro were classified by their source location differentiating between macros found in central locations (e.g. mlibrary) or in the specific project (e.g. lib01) or if the macro is contained in the program itself (Inner Definition).

A second representation lists the macros used (from macro classification “mlibrary”) for selected programs.
In this matrix design the columns display the number of times each of the macros were applied by the respective programs.

proje	study	request	mactype	Program	f_comparelist	c_list	convert	f_maxlength	c_varlist	f_format	assign_width	odsoutput	read_excel	dropmiss	frq	emailgw	mac_usage	chk_projects	funct	email	TOTAL
xxx	common	raw_adam	mlibrary	adam_codelist_issues.sas	.	1	.	.	1	.	.	1	1	.	.	.	.	.	.	.	4
xxx	common	raw_adam	mlibrary	azsol_anno.sas	.	.	.	.	.	.	.	1	1	1	.	.	.	.	.	.	3
xxx	common	raw_adam	mlibrary	impl_progress.sas	.	.	.	.	.	.	.	1	1	.	1	.	.	.	.	.	3
xxx	common	xcr	mlibrary	cr02.sas	1	.	1	1	1	.	.	1	.	.	1	.	.	.	.	.	6

Lastly the frequency of macro usage is summarized by root folder (or request) to provide an overall summary of macro usage.

projec	study	request	assign_width	convert	dropmiss	emailgw	f_comparelist	f_format	f_maxlength	frq	mac_usage	odsoutput	read_excel	c_list	c_varlist	chk_projects	funct	email	TOTAL
xxx	common	raw_adam	1	3	1	5	3	2	3	2	1	9	10	7	5	.	.	.	52
xxx	common	synthetic	2	.	.	.	.	.	.	3	.	3	.	2	2	14	1	.	27
xxx	common	xcr	.	3	.	.	2	1	2	1	.	3	.	3	3	.	.	1	19
			3	6	1	5	5	3	5	6	1	15	10	12	10	14	1	1	98

CONCLUSION

The %_mac_usage macro is a simple yet powerful tool for tracking macro use in SAS programs. Its automated reporting improves standardization, simplifies maintenance, and enhances audit readiness in clinical and regulatory programming. Developed without artificial intelligence, %_mac_usage offers reliable and transparent governance. In the future, AI may support further automation in macro generation and testing.

ACKNOWLEDGMENTS

Chris Spence and Ignacio Parra offered me the opportunity to present to a Cytel audience in preparation for the PHUSE EU Connect.

CONTACT INFORMATION

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

Author Name: Guido Wendland
Company: Cytel Inc.
Email: Guido.Wendland@cytel.com
Website: www.cytel.com