PharmaSUG 2026 - Paper AP-222

Finding Macros Called Within A Directory of SAS® Programs

Derek Morgan, Bristol Myers Squibb; Tatiana Kazakova, Parexel

ABSTRACT

As an organization heavily dependent on standard SAS® macros, BMS needed a way to efficiently document the use of those macros in its programs. In the past, this was a time-consuming, manual task requiring significant effort, and was prone to errors. Either the list was compiled after programming was completed, or it was the programmer's job to document macro use in a header for each program. It isn't difficult for one or two programs, but if you have two hundred programs and tight timelines, it becomes onerous. Also, the manual process isn't good at capturing macros called from within other macros. This method is used to find all the macros and nested macros for a given study and produces an RTF table that can be copy-pasted into an ADRG. The methodology described in this paper only uses base SAS, and it only needs read access to files. It does not modify any of the code it inspects. How long it takes is dependent on how much code is to be processed. In benchmark testing, it processed 66,422 lines across 245 files, 177 of which were report-generation programs, in approximately two minutes.

INTRODUCTION

The process is based on three main assumptions: 1) SASAUTOS contains the directories of all macros, from study- to global-level. 2) Macros have the same names as the SAS code files (A requirement for SASAUTOS to work). 3) Macros aren't called by a CALL EXECUTE statement. The user provides the path(s) of the code to search, and the program automatically adds the global library directories, which is important for finding nested macros within the global library. The general process is:

1. Inspect code files for percent ("%") signs immediately followed by a word.
2. Filter the list of %statements in code files by a list of available macros.
3. Process the list of filtered statements to yield one record per program.

Challenges included recursion for subdirectories and multiple percent signs per program line, separating main macros from the macros called within each macro, and intelligent removal of duplicates. The dataset prior to the resulting list can be used to identify the macro code that should be packaged for submission. As with most programming projects, the heavy lifting is in the solution concept and design. However, the SAS code itself isn't complicated, which makes for easier maintenance that doesn't require years of experience.

This code is part of an ongoing effort to automate the process of documenting, assembling, and packaging code for submission. The ultimate endpoint is to provide section 7 of the ADRG for all submissions.

LIMITATIONS

The program is based on BMS programming standards for file location, file naming, and code. For example, the program does not process macros loaded using %INCLUDE. Our programmers are warned that this program won't work in this case, as it is non-standard practice according to our standards.

All our primary programs invoke a standard startup program which performs a variety of set-up tasks, including assigning default LIBNAMEs, FILENAMEs, and setting SASAUTOS. SASAUTOS is built in priority order, with the individual project-level macros coming first in the search order. Per standard, all SAS macros are to be stored in a "/macro" directory. If a macro is stored elsewhere, it won't be found unless that non-standard location is included in the list of directories to search.

If our programmers reassign SASAUTOS using OPTIONS SASAUTOS, the program won't find any of the standard macros. Many of our programmers are under the mistaken impression that adding "SASAUTOS" to the SASAUTOS path will include the current value of SASAUTOS. This will cause problems not just

with this program, but also with using standard macros from the global library. That is a much larger issue, and it's outside the scope of this paper.

DISTINGUISHING MACROS CALLS FROM MACRO STATEMENTS

To make this work, we need to figure out how to identify a macro call. Rather than create a dictionary of all SAS macro statements, we create a dictionary of all possible macro calls. Our standard initialization program builds the SASAUTOS definition. Given our standard, all the macro directories are defined in SASAUTOS, so that is where we'll look. The first step is to turn the list of directories defined in the SASAUTOS option into a simple one record per directory list. The process (shown in Code Snippet 1) is simple enough: parse the SASUTOS string, remove any quotes and parentheses, and create an ordering variable (used in a later process.)

```
DATA _pass1;
LENGTH i 3 sasautodir $ 256 _sautos $ 25602;
_sautos = getoption("SASAUTOS");
i = 1;
sasautodir = '.';
DO UNTIL(CMISS(sasautodir));
   sasautodir = COMPRESS(DEQUOTE(scan(_sautos,i,' ')),"()'");
   IF NOT CMISS(sasautodir) AND UPCASE(sasautodir) NE 'SASAUTOS' THEN
      OUTPUT;
   i + 1;
END;
DROP _sautos;
RUN;
```

Code Snippet 1: Turning SASAUTOS Into a Dataset

In case SASAUTOS has been modified, we remove any duplicates so that each directory is only scanned once. This also gives us a count of how many directories are to be scanned for the next step. A sample of the output after duplicate removal is shown in Table 1.

I	SASAUTODIR
1	"../../../macros"
2	../../../../../library/programs/macros
3	../../../../../../ZZ-lib/programs/macros
4	/cdrdata/library/programs/macros
5	/cdrdata/library/programs/macros/adam

Table 1 : Sample Output from Parsing SASAUTOS

The next step is to search the enumerated directories. Since SASAUTOS is constructed on the principle that the file names match the macro names, we don't need to inspect code to get the macro names. A simple OS directory command gets that list (Code Snippet 2.)

```
%DO i=1 %TO &ndirs;

/* create ls statement to get all .sas filenames in the directory */
    DATA _NULL_;
    LENGTH callstr $ 280 abspath $ 256;
    SET _sasautodirs (FIRSTOBS=&i OBS=&i); ❶
    rc = FILENAME('fref',sasautodir);
    abspath = PATHNAME('fref'); /* need full absolute path for ls */
    rc = FILENAME('fref');
    callstr = QUOTE(CATX(' ','ls',CATS(abspath,'/*.sas')));
    CALL SYMPUTX('pipestr',callstr);
    DROP i;
    RUN;

    /* read filenames - convert to macro call - assumption is that each
       file's name is the macro call */
    DATA _macropgms;
    LENGTH ord 8 macropath $ 264 macrocall $ 32 exclude $ 1;
    ord = INPUT(SYMGET('i'),best.);
    INFILE &pipestr PIPE  PAD MISSOVER;
    INPUT @1 macropath $char264.;
    macrocall = LOWCASE(CATS('%',
                STRIP(TRANSTRN(SCAN(macropath,-1,'/'),'.sas',TRIMN("")))));
    IF FIND(macrocall,'no such file or directory','i') THEN
       DELETE;
    RUN;

   /* compile found programs */
    PROC APPEND BASE=_autocall_macros data=_macropgms;
    RUN;
%END;
```

Code Snippet 2: Get all File Names and Convert to Macro Call

The last step in this process is to remove duplicates by taking macros with identical names and selecting the one with the highest priority based on its search order in SASAUTOS. This results in a dataset with unique macro names, as well as information to determine the directory from which the code is referenced (Table 2.)

ORD	MACROCALL
1	%sec7combinedata
1	%sec7generatereport
1	%sec7generator
2	%get_potential_macros
2	%getsasautosfiles
2	%getstudymacros
3	%setlog
4	%array
9	%agearchive

Table 2: Sample of Macro Call Dictionary Dataset

LOOKING FOR \S*%\W+

Now that we have our dictionary of macro calls, it's time to inspect the code. The main challenges are efficiency and speed. Users provide the primary code directory or directories to search. Additionally, we need to search the SASAUTOS for submacro references. We have a lot of legacy compartmentalized code, with macros calling many levels of submacros, so recursion is also a must. The first thing we need to do is to add the global libraries to the code search path. Right now, we re-scan the global library each time the macro runs. This is inefficient, but as noted in the abstract, the process is relatively fast. The improvement in speed and accuracy of this task over the current manual method is material enough that further gains in efficiency have been relegated to the continuous improvement workstream.

Adding the global libraries to the search list is only a slight variation on the SASAUTOS dictionary code in Code Snippet 1, and now we have a complete list of the directories with code to be searched. For each directory in the search list, the process is the same.

1. Check the file name for a ".sas" extension, which signals that the file should be processed.
2. If the file is a SAS program, then scan the file for macro references.
3. If the file has no extension, then it's a directory. Restart the process at the next directory level.

Code Snippet 3 scans the file name. If it's a SAS program, it creates a SAS FILENAME for the file that will be used in the next part of the process. It also creates a dataset containing program names and their full paths. Otherwise, it calls the macro recursively. It's been split in this paper for easier comprehension and display. I wish I could take credit for the code concept in Code Snippet 3, particularly the recursion, but I adapted the methodology from this SAS documentation.

```
%DO i = 1 %to %SYSFUNC(DNUM(&did));
    %LET name = %QSYSFUNC(DREAD(&did,&i));

 /* Check file name for .sas extension; if found, get full path */
    %IF %QUPCASE(%QSCAN(&name,-1,.)) = %STR(SAS) %THEN %DO;
        DATA foundprog (DROP=rc fid);
        LENGTH dir fn fullpath $ 255 userid $ 16 lastmodstr $ 80;

/* pull directory and individual filename into the PDV */
        dir = SYMGET('dir');
        fn = SYMGET('name');

/* assemble filename with path */
        fullpath = CATX('/',dir,fn);
        CALL SYMPUTX('fullpath',fullpath);
        rc = FILENAME('thisfile',fullpath);
        fid = FOPEN('thisfile');

   /* Close the individual file so it can be read in next step */
        rc=FCLOSE(fid);
        RUN;
    ...to code snippet 4...
    %END; /* IF QUPCASE() */

/*   If file name doesn't have any extension assume it's a directory.
     Call routine recursively, crawl, and repeat where neccessary */
    %ELSE %IF %QSCAN(&name,2,.) = %THEN %DO;
         %get_potential_macros(&dir/&name);
    %END;
```

Code Snippet 3: Scan Directory for SAS Programs and Create a SAS FILENAME for Each One

The code in Code Snippet 4 scans each line of the file and uses a regular expression to output lines with potential macro calls. We also flag primary programs by examining the directory where they are located.

This lets us distinguish between primary SAS programs (to be listed in section 7) from other SAS programs. By standard, all primary program files must be stored in a “pgm_code” directory. The %END statement may look a little strange by itself, but the matching %DO is at the top of Code Snippet 3.

```
   /* read code file allocated in previous step */
    DATA macrocalls;
    fn = SYMGET('name'); /* Pull filename into PDV */

/* Use filename assigned in Code Snippet 3 to get input file */
    INFILE thisfile PAD MISSOVER;

    INPUT @1 line $char255.;
    line = STRIP(LOWCASE(line));
    fullpath = SYMGET('fullpath');

/* Search for % immediately followed by a word and output line */
    IF PRXMATCH('/\s*%\w+/',line) THEN DO;
       IF INDEX(fullpath,'pgm_code') THEN /* distinguish program files */
          pgmflg = 'Y';
       OUTPUT;
    END;
    RUN;

/* NOW clear the filename after finished with file */
    %LET rc = %SYSFUNC(FILENAME('thisfile'));

/* put program name list together */
    PROC APPEND BASE=allsaspgms DATA=foundprog;
    RUN;

/* put potential macro calls together */
    PROC APPEND BASE=allpercentsigns DATA=macrocalls;
    RUN;

%END; /* This is the directory loop from cope snippet 3 */

/* housekeeping - close directory and clear directory filename reference */
%LET rc=%SYSFUNC(DCLOSE(&did));
%LET rc=%SYSFUNC(FILENAME(filrf));
```

Code Snippet 4: Scan SAS Programs for Macro Calls

The process is repeated for each user-provided directory and the global library. Both the SAS program name and the lines with %signs are compiled using PROC APPEND. Now we have a list of potential macro names, a dictionary of macros to compare them against, and a list and location of the programs where each macro was found.

File Name	Path
rt-ex-ratered.sas	/project/rpt/pgm_code/rt-ex-ratered.sas
rt-ae-im.sas	/project/rpt/pgm_code/rt-ae-im.sas
infmt-lib.sas	/project/rpt/pgm_code/infmt-lib.sas
rl-ae-aecmim.sas	/project/rpt/pgm_code/beforepc/rl-ae-aecmim.sas
rt-ef-os.sas	/project/rpt/pgm_code/beforepc/rt-ef-os.sas
dv_subset.sas	/project/project-macros/dv_subset.sas

File Name	Path
dv_fmt.sas	/project/project-macros/dv_fmt.sas
gensastoxpt.sas	local-macros/esub/gensastoxpt.sas
esubxpt2sas.sas	local-macros/esub/esubxpt2sas.sas
adeftmsrdinvnadir.sas	local-macros/adam/adeftmsrdinvnadir.sas
adrsborss.sas	local-macros/adam/adrsborss.sas
cutcat2.sas	local-macros/cutoff/cutcat2.sas
cutenddt.sas	local-macros/cutoff/cutenddt.sas
isblank.sas	global-macros/isblank.sas
nobs.sas	global-macros/nobs.sas
xobjexist.sas	global-macros/tlg/xobjexist.sas
filecopy.sas	global-macros/tlg/filecopy.sas

Table 3: Programs to Search and The Full File Name and Path

The important thing to note here is the recursion. The program collects not only the files in the specified search directory (e.g., “pgm_code”) but also its sub-directories (e.g, “pgm_code/beforepc”.) It does the same recursion for all directories in the search path, as seen in “local-macros” and “global-macros”. Only the top level is specified in the macro call. The code in Code Snippet 3 handles the rest.

DISTINGUISHING MACRO CALLS FROM WORDS STARTING WITH “%”

The next step is to find the potential macro calls in the flagged lines. It’s not uncommon to have multiple percent signs in a single line as in Example 1. It doesn’t matter that none of the % signs in this example are macro calls. We still need to check them against the dictionary.

```
%IF %QUPCASE(%QSCAN(&name,-1,.)) = %STR(SAS) %THEN %DO;
```

Example 1: Multiple Potential Macro Calls in a Single Line

All we want at this stage is one record per potential macro call, so we parse the flagged lines. We scan the selected lines and select any potential macro call words. There are obvious exceptions, and we don’t want to include the standard set-up macro %INIT (Code Snippet 5.)

```
DATA potential0;
SET allpercentsigns;
LENGTH macrocall $ 200.;
_percents = COUNTC(line,'%');
_strt = 1;
DO i=1 TO _percents;
   _strt = FIND(line,'%',_strt);
   macrocall = LOWCASE(SCAN(SUBSTR(line,_strt),1,' ,;('));
   IF NOT INDEX(macrocall,'*') and macrocall NOT IN ('%str','%init')
      AND ANYALPHA(macrocall) THEN
          OUTPUT;
   _strt+1;
END;
DROP _strt i; /* leave for debugging */
RUN;
```

Code Snippet 5: Parsing % Signs

We remove any macros that are defined within a program (i.e., %macro mymacro;) before detecting macro calls. That code isn't shown here, and there's another duplicate removal to yield POTENTIAL2, the one %-word per record dataset. We derived the macro call "dictionary," _MACROS2MATCH, in Code Snippet 2. A simple MERGE filters the macro calls from all other % words (Code Snippet 6.)

```
DATA only_macro_calls;
MERGE potential2 (IN=ina) _macros2match (IN=inb);
BY macrocall;
IF ina AND inb;
RUN;
```

Code Snippet 6: Getting Macro Calls

PROGRAMS CALLING MACROS VS. MACROS CALLING MACROS

Now that we have all our macro calls, we separate primary programs from macros called within a primary program or another macro. Determining primary program is simple: we use the PGMFLG variable created in Code Snippet 4. The rest? That takes a little bit of handwaving, but the code is straightforward.

```
DATA pgm_macros (DROP=submacname)
     submacros (rename=(macrocall=submaccall));
SET only_macro_calls;
LENGTH submacname $ 34;
BY macrocall;
IF pgmflg EQ 'Y' THEN
   OUTPUT pgm_macros;
ELSE DO;
   submacname = LOWCASE(CATS('%',SCAN(fn,1,.')));
   OUTPUT submacros;
END;
RUN;
```

Code Snippet 7: Separating Program Macros from Submacros

We sort the primary program macros to remove duplicates. That will come in handy when we put everything together. The duplicate dataset, DUPS, is used for debugging. If you aren't familiar with the NOUNIQUEKEY option, refer to this paper. Now, let's deal with the issues of macros being called from other macros. If it isn't a program file, then it's a submacro, and therefore, the macro being called is a submacro, so we refer to it as a submacro call, and rename it accordingly. (Line 1, Code Snippet 7.) We also give it a submacro name (line 8.)

Now let's filter the list of submacros by some standard rules. (Code Snippet 8.)

```
PROC SQL NOPRINT;
CREATE TABLE submacros2 AS
SELECT *
FROM submacros
WHERE LOWCASE(submacname) NE LOWCASE(submaccall) AND /* submacros don't
call themselves */
      NOT INDEX(fn,'-') AND /* macros don't use hyphens */
      FULLPATH NOT CONTAINS 'template' AND /* don't want templates */
      FULLPATH NOT CONTAINS 'retired' AND     /* Only active macros */
      SCAN(fullpath,-1,'/') NE 'old' /* don't want old macros */
ORDER BY submacname, submaccall
;
QUIT;
```

Code Snippet 8: Initial Submacro Filtering

Next step is matching the submacros with their calling programs. Note that the second query isn't a cartesian join; it's just selection that replaces a DATA step and a PROC SORT.

```
/* get macros used in non-program macros, keep calling program filename  */
PROC SQL NOPRINT;
CREATE table pgm_submacros AS
SELECT p.fn, p.pgmflg, p.macrocall, s.submacname, s.submaccall
FROM pgm_macros p, submacros2 s
WHERE p.macrocall EQ s.submacname
ORDER BY fn, macrocall, submaccall
;

/* want filename that calls submacro and submacro call
   rename submacro call for next step */
CREATE TABLE pgm_submacros2 AS
SELECT DISTINCT fn, submaccall AS macrocall
FROM pgm_submacros
ORDER BY fn, macrocall
;
QUIT;
```

Code Snippet 9: Get Non-Program Macro Calls

PUTTING EVERYTHING TOGETHER

After that, we add the assembled submacros to the list of macros called from primary programs using the de-duped PGM_MACROS dataset created in Code Snippet 7. The list of macro calls needs to be de-duped by filename, since submacros can (and often do) use macros that are also called in the primary program. We transpose that to create one record per primary program file, with one variable per macro called. All that's left is to convert the multiple variables into one long, nicely formatted character variable. A simple PROC REPORT handles the formality of output production into a user-specified file. All the code for this is shown in Code Snippet 10.

```
/* add list of submacros to list of macros called by main programs */
DATA staging1;
SET pgms pgm_submacros2;
BY fn macrocall;
RUN;

PROC SORT DATA=staging1 OUT=staging2 NODUPKEY;
BY fn macrocall;
RUN;

/* 1 record per filename, all macro and submacro calls */
PROC TRANSPOSE DATA=staging2 OUT=staging3 (DROP=_name_) PREFIX=mac;
BY fn;
VAR macrocall;
RUN;


/* final product - turn transpose list into nicely formatted text */
DATA final;
SET staging3;
LENGTH mlst $ 9216 pgm_name $ 200;
BY fn;
pgm_name = TRANSTRN(fn,'.txt','.sas');
ARRAY maclst{*} mac:;
DO i=1 TO DIM(maclst);
   mlst = CATX(', ',mlst,maclst{i});
END;
IF CMISS(mlst) THEN
   mlst = "No macros called within this program.";
LABEL
  pgm_name="Program Name"
  mlst = 'Macros Used'
;
DROP i mac:;
RUN;

ODS _ALL_ CLOSE;
ODS RTF FILE="&outdir/&outfn..rtf";
PROC REPORT DATA=final NOWD;
COLUMNS pgm_name mlst;
RUN;
ODS RTF CLOSE;
```

Code Snippet 10: Creating the Report

A sample of the RTF output produced by this process is shown in Table 4. It's important to remember that this itemizes the macros used by each program, not the macros used in each project. There's a lot of duplication across programs. This is to be expected when submission tables, listing, and figures have a high degree of automation through macro usage.

Program Name	Macros Used
c-rl-dm-stratdiscrep.sas	%closertf, %colz, %createbmsrtf, %mkrtfstyle7, %openrtf, %xfmt, %xftncode, %xobjexist, %xtitcode
dpp2meta-test.sas	%createtmplib, %dpp2meta, %getabspath, %xfmt, %xlsx2dsn, %xtidy, %zls
mapping-fmtlib.sas	%commvars, %compact, %deldsns, %fisher, %getkeyparam, %invkmac, %pic, %substrx, %trim, %vfreq, %xgetvarattr, %xobjexist
rg-ae-mo.sas	%closertf, %createbmsrtf, %mkrtfstyle7, %openrtf, %xfmt, %xftncode, %xobjexist, %xtitcode
rg-bm-bp.sas	%closertf, %createbmsrtf, %mkrtfstyle7, %openrtf, %xfmt, %xftncode, %xobjexist, %xtitcode
rl-ae-ae.sas	%closertf, %colz, %createbmsrtf, %dtcchg, %genobsexist, %mkrtfstyle7, %openrtf, %xfmt, %xftncode, %xobjexist, %xtitcode
rl-dm-accr.sas	%closertf, %colz, %createbmsrtf, %dtcchg, %mkrtfstyle7, %openrtf, %xfmt, %xftncode, %xobjexist, %xtitcode
rt-lb-wo.sas	%atleastn, %chkprt, %closertf, %colz, %commvars, %compact, %createbmsrtf, %deldsns, %fmtpct, %getkeyparam, %invkmac, %mkln, %mkorder, %mkrtfstyle7, %npct, %openrtf, %pic, %qtrim, %substrx, %trim, %ws_intersect, %xfmt, %xftncode, %xgetdsnattr, %xgetdsnlbl, %xgetvarattr, %xobjexist, %xtitcode, %zdent

Table 4: Sample Output

SUMMARY

The code in this paper presents a reasonably efficient and thorough way of inspecting many files for external macro calls and documenting them. Are there alternate ways of doing this? Of course. This is SAS, after all. Are there more elegant ways of coding this? Certainly. Are there efficiencies to be had? Yes. I could speed up the execution and file manipulation using HASH objects instead of DATA steps and PROC SQL, among other SAS tricks. However, in terms of maintenance, this solution is accessible to all but the most beginning of SAS coders. Each piece uses basic SAS language, and the code doesn't require an expert to maintain.

But the bar for this process improvement is low: this replaces a painful, error-prone manual process. Anything under an hour would be a significant reduction in time (and programmer effort since they can perform other tasks while this is running.) Given the benchmarking we've done so far, we can extrapolate that even with 600+ primary program files, we easily make that mark. In short, this process is a significant improvement in efficiency and accuracy of this task, and will speed the creation of ADRG section 7.

REFERENCES

Programming Documentation for SAS® 9.4 and SAS® Viya® 3.5, SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513, USA. https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/mcrolref/n0js70lrkxo6uvn1fl4a5aafnlgt.htm, 2026-02-23.

PROC SORT (then and) NOW, Derek Morgan, Western Users of SAS® Software 2024 Conference Proceedings, https://www.lexjansen.com/wuss/2024/144_FINAL_paper_pdf.pdf, 2026-02-23

ACKNOWLEGEMENTS

As always, Derek would like to thank SAS Technical Support for all the help they've given him over the course of his career. Tatiana would like to thank Roman Igla who basically defined her path in statistical programming and led to her understanding of the need for standardization.

CONTACT INFORMATION:

Further inquiries are welcome to:

Derek Morgan
E-mail: derek.morgan@bms.com