PharmaSUG 2021 - Paper AP-008

Don't send the Macros, send the Catalog! The SAS Macro Catalog: a perfect way to send your macros to the client without sharing the code

Steve Black MSPH, Precision for Medicine Inc.

ABSTRACT

If you've ever gotten squeamish about sending a macro or a lot of macros to a client, SAS® has created a way to help ease the tension. A SAS Macro Catalog allows you to send your macro along with the option of hiding the code or not. Saved as single file, the catalog is easy to transfer and apply to any study. A simple call will activate all the macros stored and then they can be used just as they are set up in your system.

In this paper I will show how to create a SAS Macro Catalog, add macros to the catalog, and manage the catalog once it has been set up. I'll then demonstrate how to create a macro which adds a large number of macros to the catalog that have already been created. With this newfound skill, you can create and send out one file and rest easy knowing that your super fancy macro logic is safe and secure.

INTRODUCTION

In the Clinical Research Organization or CRO industry, it is very common for a sponsor to request the programs and datasets used to create the various generated outputs, which we happily provide, but when it comes to providing all the macro's used for a study, we tend to get a little squeamish, and for good reason.

Most macros that are created for various dataset (ADAM, SDTM) and outputs (tables, listings, and figures) are pretty simple and not really considered intellectual property and are simply created to speed up the process or keep a consistency in generating output. There are, however, a number of macros that are unique to the company, which required many hours to create and perfect so that they can run either for a number of situations/studies or are built for a very specific process. These macros are held close to the chest and generally are not shared with all clients.

However, sometimes a client will want to be able run the programs that they have been sent in order to perform their own checks against the data. This can be massively problematic if the programs utilize a lot of macros that are not included directly in the programs provided. There are a few ways of going about overcoming this obstacle, such as recreating the programs with all of the macro logic built in, which is doable using mprint and a number of other steps –a huge pain. Or you can utilize the SAS Macro Catalog to store all the macros in one file, which then can be used by the client to run the whole set of programs. You can also hide the source code for some macros and allow others to be displayed.

Setting up the macro catalog is quite easy. Placing a macro in the catalog is also quite easy. Calling the Macro catalog is also very easy. So what's the hard part? Placing 100+ macros in the catalog that have already been created. But first let's work on the easy parts, then we can look at the hard part.

SETTING UP THE MACRO CATALOG

A SAS Macro catalog is essentially a single file that contains all the macros that are deliberately stored therein. This file can then be transferred and stored just like any other file. A macro catalog can be set up to be super secure so that no one can actually see the source code, only the names of the macros stored, or you can also set up so that the code is visible. This can be done per each macro stored.

Setting up a catalog is super easy. Simply set up a libname with a name and location:

```
libname macrocat "C:\Biostats\Programs\Macros";
```

Then set up the SAS options to store the macros to the libname created:

```
option mstored sasmstore = macrocat;
```

Once set, SAS holds this macro catalog open during the duration of the SAS session. You literally have to close down the SAS session to close the macro catalog – in case you want to move it or delete the file, and you can only create one SAS macro catalog per SAS session, which is kind of a pain but not a deal breaker. An alternative is to deactivate the libname by using the clear option:

```
libname macrocat clear;
```

ADDING IN A MACRO

Once that catalog is set up, you need to add a little bit of code to each macro to get it into the catalog.

```
%macro cool_logic / store;
%macro first_flag (subset, data, data2, sort, first, flag, seq) / store;
```

If you want to store the code in the catalog simply add the option: source

```
%macro adamf / store source;
```

Once re-run the macro will be included into the catalog.

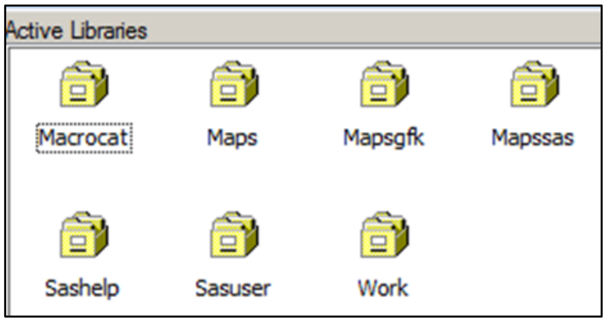
To see the source code in the log use: `%Copy MACRO_NAME /Source;` You can also specify an out file instead of the log. Note: This only works if the Source option was used when created.

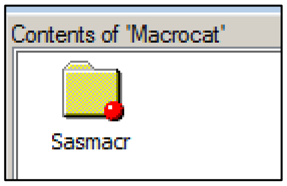
MANAGING THE CATALOG

If you want to see the macros that have been included in the catalog, you can do this two ways:

```
proc catalog catalog=macrocat.sasmacr;
contents;
quit;
```

Or visually by clicking on the macro catalog created with the SAS Explorer Window



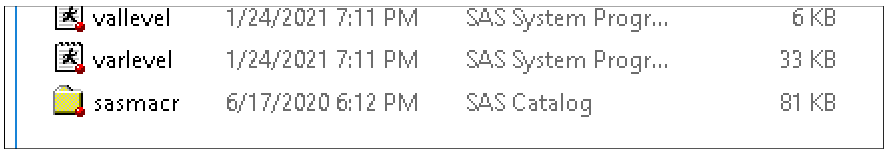


Display 1. Example of Macrocat Library and SASMCR file within SAS Explorer Window.

Once shown, you can manually delete those macros that are not needed within the Explorer window.

SAS also provides a clear prioritization as to when these macros would be utilized. The highest priority are those macros that are compiled during the current SAS session. Next up are those within the SASMacr catalog when activated using the MSTORED and SASMSTORE options, followed by those macros in the autocall library when the SASAUTOS is activated using the SASAUTOS = and MAUTOSOURCE options. Lastly any stored and compiled macros in the SASMacr catalog within the SASHelp library.

The catalog itself is stored as a file wherever you have specified it using the libname statement.



vallevel	1/24/2021 7:11 PM	SAS System Progr...	6 KB
varlevel	1/24/2021 7:11 PM	SAS System Progr...	33 KB
sasmacr	6/17/2020 6:12 PM	SAS Catalog	81 KB

Display 2. Example of SASMACR file within browser window.

ISSUES FOUND

So that's pretty easy: setup the libname, add in the options, add in the code to the macro, then run the macro, done! But wait! If I want to catalog a lot of different macros, say 100 of them, then this gets to be a bit of a pain to add the code to every single program and then re-run. Can't SAS do this? Luckily, the answer is YES it can!!

LET'S BUILD A MACRO!

This next section of the paper will provide lots of valuable information regarding macro creation and dynamic programming. I'll explain my steps generally but won't dive into every option available, just the ones that I have chosen to use for the purposes of creating the macro.

Step one – setup and bring in the filenames.

Sometimes it's nice to see the end from the beginning, and macros can be that way, especially when creating the parameters. Here is a run-down of items we need to create this macro:

We need to know where the macro programs are stored (macro_dir), we need a name for the macro catalog (Macro_cat_name;, we need a toggle for keeping the source code or not (keep_source), a toggle to delete the temporary directory and files (delete_temp), and a list of potential excluded macros (exclude).

So first let's call the macro MACROCAT and create the parameters:

```
%macro macrocat (macro_dir, macro_cat_name, keep_source, delete_temp,
exclude);
```

Next, let's bring the list or directory of the macros found in the &macro_dir parameter. I like using the X function to do this within a data _null_ datastep. I use the noxwait option so I don't need to click anything to get it to work. In this step I'll create a temp folder to store the modified macros so nothing gets overwritten.

** closes command window after prompts are run **;

```
options noxwait;
```

*** send directory to c drive and creates temp drive for macro storage ***;

```
data _null_;
x %sysfunc(quote(dir "&macro_dir.\*.sas" /n/-c > c:\dir.txt));
x %sysfunc(quote(mkdir "&macro_dir.\temp"));
run;
```

Next, I'll read in that directory files using the infile statement and limit the files to just SAS programs.

*** bring in directory of files ***;

```
data _in_dir; length linein filename $200 program $50 ;
infile "C:\dir.txt";
input;
linein=lowcase(_infile_);
```

*** keeping only .SAS files ***;

```
if index(linein,'.sas');
*** create filename and program variables ***;
filename=upcase(strip(substr(linein,40)));
program=substr(filename,1,length(filename)-4);
```

```
run;
```

Next, I'll use the values in the exclude statement to remove those files not needed to be included in the macro catalog. You could do this outside of SAS, but in our company there are a standard number of macros that would never need to be sent to a client as they do not assist in the dataset creation but are more system macros. In the example below, I assume that the exclude is a list of macros program names separated by a comma.

```
data _null_;
exclude="&exclude";
call symput('max_exclude',count(exclude,',')+1);
run;

/*  %put &max_exclude;*/
%do x = 1 %to &max_exclude;
%let excl&x=%upcase(%scan(&exclude,%eval(&x),','));
/*  %put &&excl&x;*/
%end;

proc sort data=_in_dir out=_in_dir_final;
by program;
where filename not in (
%do x = 1 %to &max_exclude;
    "&&excl&x"
%end;
);
```

Now with the final list of macros needed we can macrotize this list and loop through each program updating the code where needed. We can do this using PROC SQL.

*** macrotize filenames and programs ***;

```
proc sql noprint;
select distinct filename, program into: file_name1 -: file_name900,
:prog_name1 -: prog_name900
from _in_dir_final;
%let num_files=&sqlobs;
quit;
```

Now with this information macrotized, I know how many total files I need to convert over (&num_files) and I have both the filename and the program saved as macro variables with a numeric suffix.

The next step I again use the data _null_ datastep and use the infile statement to read in the SAS program as if it were a dataset. I'll search for the first line with %macro in the text and make the adjustments needed.

```
data _null_;
*** loop through each filename ***;
%do x = 1 %to &num_files;
    data _infiles;
    infile "&macro_dir.\&&file_name&x";
    input;
    linein=_infile_;
if index(upcase(linein),"MACRO &&prog_name&x") then found=1;
```

One major assumption is that the filename and the macro name are the same. If not, then some logic would need to be added to account for the differences. In the next step I use the toggle for keeping the source (Y or not Y) and modify the logic accordingly:

```
%if &keep_source=Y %then %do;
    if found=1 then linein=tranwrd(linein,';',' / store source;');
%end;
%else %if &keep_source ne Y %then %do;
    if found=1 then linein=tranwrd(linein,';',' / store;');
%end;
run;
```

I add a little code to warn me if the variable found is not being created properly:

```
proc sql noprint;
select count(found) into: max
from _infiles;
quit;
%if &max ne 1 %then %put "Warning: No matching macro statement in
&&file_name&x";
```

Now that the files have the update needed, I can push them back out as SAS programs into the temp folder created again in the data _null_ datastep using the infile statement. The '$varying32000.' solves a lot of potential issues with how SAS re-writes the code with the spacing and removal of hidden characters. At the end of this step, we also close the main %Do Loop:

```
data _null_ ;
set _infiles (keep=linein);
FILE  "&macro_dir.\temp\&&file_name&x";
varlength = lengthn(trim(linein));
PUT linein $varying32000. varlength;;
run;
%end;
```

Now we can create the SAS macro catalog and place these newly created files within the catalog. We can use the macrotized variables from both the parameter list and those created earlier in the PROC SQL step to accomplish much of this:

```
/*Create libname for macro catalog*/
libname &macro_cat_name "&macro_dir";
/*Set up the macro cat options:*/
 option mstored sasmstore=&macro_cat_name;
%do x = 1 %to &num_files;
    %include "&macro_dir.\temp\&&file_name&x";
%end;
```

Final steps: I want to make sure that nothing went wrong in the new macro creation process and that the macros really are stored in the macro catalog. So, using proc catalog, I'm able to get a list of all the macros within the catalog and then compare the programs in the original directory look to see if we have any missing that we expected to be included.

```
ods listing close;
    ods output Catalog_Random=_curr_mac_cat;
    proc catalog catalog=&macro_cat_name..sasmacr;
    contents;
    quit;
ods listing;

data _curr_mac_cat;
set _curr_mac_cat;
length program $50;
program=objname;
```

```
drop objname;
run;

data _compare;
merge _curr_mac_cat (in=a) _in_dir_final (in=b);
by program;

if a and not b then put "NOTE: program is in the macro catalog but not in
the list of valid macros:" program= ;
if b and not a then put "WARN""ING: program did not get created in the
macro catalog, may need to open program then clear the white space (CTL+A
then CNTL+SHFT+W) and re-run individually:" program= ;
run;
```

Lastly, I use the delete_temp toggle parameter to delete or leave alone the temporary folder and files created. Using the X command again to let SAS do all the work. Then I close the macro with a mend statement.

```
%if &delete_temp =Y %then %do;
    x %sysfunc(quote(del /q "&macro_dir.\temp"));
    x %sysfunc(quote(rmdir -r "&macro_dir.\temp"));
%end;
%mend;
```

So my final macro call would be:

```
%macrocat (MACRO_DIR=%str(C:\Biostats\Programs\Macros),
MACRO_CAT_NAME=MACROCAT, KEEP_SOURCE=N, DELETE_TEMP=N,
EXCLUDE=%str(annogrid.sas, batch.sas, batch_dmc.sas, batch_qc.sas) );
```

Once run, this macro would create the macrocat file stored in the location specified. To utilize this file, simply create the library again and then use this code to activate the macros:

```
libname macrocat "C:\Biostats\Programs\Macros";
option mstored sasmstore = macrocat;
```

CONCLUSION

SAS macro catalogs are super easy to create, modify and move. They provide a secure way of enabling others to use your macros without being able to see them – if wanted. They do require a touch of work when dealing with macro programs that have already been created, but thanks to SAS this problem can be handled dynamically.

CONTACT INFORMATION

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

Steve Black, MSPH
Precision for Medicine
760-814-9620
steve.black@precisionformedicine.com