

A General Purpose Macro to Obtain a List of Files: Plus Macro Programming Techniques

Mel Widawski, UCLA, Los Angeles, California

ABSTRACT

Would you like to be able to accomplish any of these tasks automatically? 1) Access and convert all .dbf files in a given directory. 2) Write a SAS® macro to maintain an annotated list of files all files in a directory. 3) Create LIBNAME statements for all transport files in a directory. 4) Process all files of a given type in a directory for any reason.

In order to accomplish any of these tasks you would need to obtain a list of files. This macro produces a macro variable with a list of all of the files in a directory. It may be further restricted to produce a list of files of a given extension in that directory. In the course of presenting this macro the following techniques will be covered:

- **X** (host command),
- **&&** for delayed processing,
- **&&&** for ampersanding the contents of a macro variable.

Special provisions have been made so that a list of files longer than the 200 character limit for character variables may be created. This macro was developed for use on a PC but may be adapted for use on other systems.

INTRODUCTION

This macro grew out of extreme laziness. I had overcome the burden of copying and pasting code for performing similar tasks on a group of data sets, but I was still typing or collecting lists of these files to supply to these macros. My desire to automate this task led to this macro.

A key to using the **GETLIST** macro is to organize your files into reasonable directories. Isolating them is best, but the files may be mixed with other files as long as they are identifiable, perhaps by a common extension.

The macro itself is rather simple, but it has a myriad of uses. In writing the macro I made use of the versatility of the SAS system in interacting with its host environment. Specifically I am referring to the X statement for submitting commands to the host operating system.

In the sections that follow I will present the macro call, and dissect the macro explaining its components. As you take this macro from system to system you may need to modify parts of it. I will endeavor to point out the need for modification of the code as it occurs. Comments in the macro also point out locations where modifications are needed.

Examples of file paths in this report are in PC form. I am assuming that, if you are adapting this macro to another system that you know the path specification syntax for your operating system. Items that appear in the programs are often bolded in the text to make them easier to find.

USING THE GETLIST MACRO

You will have greatest success if you call the macro in the body of your program or in another macro. It produces a macro variable that will hold the list of files. You will supply the name of this variable when you call the macro. The following is an example of the way I like to invoke the macro.

```
%include 'd:\amel\sas\macros\getlist.sas';
/* Brings in the macro */

%let sasin=d:\amel\sasds\; /* path for input
                           directory */
%let outdir=d:\amel\lists\; /* path for output
                             directory */
%let insub=newfiles; /* name of sub-
                      directory */
%let listf=filelist; /* name of macro
                      variable to hold
                      the list */
%let xt=sd2; /* file extension to
              limit the search*/

/* ***** */
/* **** The macro call **** */
/* ***** */
%getlist (infref, &sasin, &insub, &outdir, &listf,
          &xt)

run;
```

%INCLUDE

You use the %INCLUDE statement to make the macro available to the program. Thus, you don't have to type or paste the macro code in every program that invokes the macro.

If you are going to create and use a variety of macros, you should set up an autocall library. I tend to use the %INCLUDE statement so that I can quickly distribute macros that will work without requiring others to set up an autocall library.

%LET

The statements that begin with %LET establish macro variables for use with the macro. They also assign values to these macro variables. I prefer to define these variables, rather than include all of the information in the macro parameters, for readability. This practice also allows you to better document your program. I like having all of the main variables that may need changing at the beginning of the program.

The macro variable "**sasin**" contains the full path name for the directory that you want to search. There is nothing magical about the name "sasin." You can use any name you wish, as long as you specify the name both in the %LET statement and in the parameter list on the macro call. I use the macro variable "**outdir**" to hold the path for the directory I will use to store the temporary list of files.

You can supply the getlist macro a subdirectory name to append to the search directory. I use the macro variable "**insub**" for this purpose. You may have a general directory defined and need to obtain lists of files from various subdirectories within the main directory. That is when you will want to use the insub macro variable. If you don't want to specify any subdirectories, then simply follow the equal sign in the %LET statement with a semicolon, as follows:

```
%let insub=;
```

The macro detects this condition and uses only the directory specified without an additional subdirectory.

You don't have to use these additional macro variables, but I show them here for two purposes. First, you will find many uses for macro variables in your general programming. You can specify macro variables to hold file names and variable lists at the top of your program, and refer to them throughout your program. This greatly simplifies the process of changing your program. Second, you will probably be calling the macro from other macros, and the information will already be stored in macro variables.

The Macro Call

The `%getlist` statement invokes the macro. The parameters that make the macro work are enclosed in parentheses following the name of the macro. You already know the functions of each of these macro parameters from the discussion above. To refresh your memory the macro call follows:

```
%getlist (infref, &sasins, &insub, &outdir, &listf,
          &xt)
```

Each of the macro parameters is positional. The order determines how the macro interprets them. I will use the contents of the macro call above to define what goes in each position.

- infref** This parameter supplies a fileref for reading the list of files you are storing on disk. I include it so that you can avoid conflicts with filerefs you are currently using without having to modify the macro. You would use `&outdir` to supply the full path specification of the directory for this file. I use the same name as supplied by `&listf` with `.txt` appended to it as the name for this file.
- &sasins** This parameter specifies the path to the directory you want searched. You do not have to use a macro variable here, as the full path name could be typed in the macro call.
- &insub** If it is convenient for you to be able to specify a subdirectory for your search, then use this parameter for that purpose. Often your directory will be defined by `&sasins` and you specify a subdirectory to be searched.
- &outdir** The macro creates a file on disk that will hold the list of files. Use this parameter to specify the path to the directory to contain this file. You do not have to use a macro variable here; just specify the full path between the commas in the fourth position.
- &listf** This is where you tell the macro what macro variable name to use to hold the list of files. You will use that name in other macros to process these files.
- &xt** Limit the scope of your search by specifying this parameter. It holds an extension like `.sd2` for SAS data files.

As mentioned previously, none of these parameters have to be macro variables. This is how the macro call would look with no macro variables as parameters:

```
%getlist (infref, sasins=d:\amel\sasds\, newfiles,
          d:\amel\lists\, filelist, sd2)
```

The path specifications are relatively short here, but you might imagine how contorted this could be with long path specifications.

That is why I prefer using macro variables. If you are not using a subdirectory, the specification would be:

```
%getlist (infref, sasins=d:\amel\sasds\,,
          d:\amel\lists\, filelist, sd2)
```

Notice that the subdirectory name is left out and the commas closed up.

The macro variable `numfms` will contain the number of files in the list. If you need to keep this variable through multiple calls to the macro just assign its value to another macro variable using a `%LET` statement.

PROGRAMMING THE MACRO

Here are some of the techniques I used in programming this macro. I have grouped the discussion around techniques and the problem. Where appropriate I will discuss necessary changes for adapting the macro for use with other operating systems.

The X (host) Statement

You can use this statement to submit any host command. I use it here to submit the `DIR` command to the PC host. The `DIR` command lists directory information. This information is directed into a file through the use of the greater than sign (`>`). This command appears as follows:

```
x "dir &dirin&insub.*&tststr>&dirout.&listf..txt/O:N ";
```

Notice how the macro variables, `dirin` and `insub` are used to supply complete path information. The above statement works for Windows 3.1[®] and Windows 95[®]. To make the macro work with Windows NT[®] you must add `/-N` to the statement.

```
x "dir &dirin&insub.*&tststr>&dirout.&listf..txt/O:N
  /-N";
```

This operand turns off the new formatting of the `DIR` command. This allows the same code to read both from Windows NT, and Windows 95 and Windows 3.1. The code here is system dependent, and will need to be updated from time to time. The macro does not easily handle the longer file names now allowed on Windows 95 and Windows NT.

When you request all files by omitting the specification of the extension, The operands `/B` and `/A:-D-H` are added to the command, yielding a brief listing with no directories or hidden files listed. If no extension is specified, this is done automatically by the macro. The following checks to see if there is an extension specified in the macro variable `tststr`:

```
%if &tststr ne %then
  %let tststr=.&tststr;
%else
  %let tststr= /B /A:-D-H;
```

If there is an extension, then the macro adds a period before the extension name, and the result is stored in the macro variable, `tststr`. If the extension is null, the macro adds the operands, `/B` and `/A:-D-H`. These operands are supplied to the following `X` command in the macro variable, `tststr`. Output of the two commands is quite different. The first produces a fixed file with separate file name and file extension, while the second produces a list with *filename.extension*.

NOTE: The command used in UNIX would be entirely different. UNIX uses the `LS` command to obtain directory information. The output of this command is similar to the output of the `DIR` command with the brief (`/B`) operand.

Controlling Your SAS Code

The %IF statement with %THEN and %ELSE determines the form of the input statement when the code runs.:

```
input %if &teststr eq %then   name $;
      %else name $8. @10 fext $char3.;
```

When there is an extension, the macro reads the fixed field list produced by the host command. When there is no extension, the free list is read using free field input. You need to do this because the records are of varying length. This assumes no imbedded blanks in the field you are reading. The output macro variable will contain names in different forms for the two purposes. When you select a specific extension, the program lists only the filename portion. Selecting all files results in the list containing the complete file name and extension separated by a period.

NOTE: the latter code will probably work in UNIX, but more programming would need to be done to extract the filename portion from the full name. You could add this code to the GETLIST macro or create a separate macro to call whenever this type of parsing is needed.

Collecting Names in Macro Variables

The 200-character limitation problem meant that I had to assign the list to a macro variable each time the list got near the limit. I decided that creating and retaining an indeterminate number of variables in the DATA step would be inefficient. The SAS SYMPUT function assigns values to a macro variable, and has the advantage of allowing the creation of new macro variables as needed. The following three SYMPUT calls illustrate the point:

```
call symput(mvar,namelst);
call symput('nummvs',j);
call symput('tnumfs',nfs);;
```

The second and third call to SYMPUT are fairly straightforward. The second assigns the value of the variable j to the macro variable nummvs. The first call while simple, exhibits the power of the method. Note that mvar is a SAS variable, and it contains the name of the macro variable. The next statements demonstrate the DATA step statements that take advantage of that power:

```
j=j+1;
mvar=trim(left('namelst'))||trim(left(j));
```

Each time that the SAS variable namelst reaches its limit, and the value is assigned to the macro variable named in the SAS variable namelst, the pointer j is incremented. This pointer is then used in creating the name of the next macro variable in the sequence. This allows you to create only those macro variables you need. At the final step the number of files in the list (nfs) and the number of temporary macro variables you created (j) are stored in the macro variables tnumfs and nummvs respectively.

Combining Into a Single Macro List

All you have to do now, is combine the temporary macro variables into a single macro variable. The following macro %DO loop accomplishes this task:

```
%let &listf=;
%do i=1 %to &nummvs;
  %let &listf=&&listf %trim(&&namelst&i) ;
%end;
```

From the DATA step we know how many macro variables were used (**nummvs**) and how many files there are in the list (**tnumfs**). So it is a simple matter to loop through the temporary variables to

append them to the output macro variable of your choice. The macro variable listf contains the name of your macro variable. We are now going to unravel the mystery of the ampersands, especially the elusive && and &&&. Notice the %LET statement contains all of the ampersand combinations. As **listf** contains the name of your macro variable, it resolves to that name, and your macro variable gets assigned the values of each list concatenated to it. At the end of the loop it contains all of the files.

The second occurrence of **&&&listf** contains three ampersands. Remember that we want the value of your macro variable, the base list, to join with the next temporary variable. The function of the **&&&** is to allow the macro variable whose name is stored in listf to be evaluated. That is to say, we want the list, not the name of the variable containing the list.

Finally, the last set of macro variables, **&&namelst&i**, evaluates as follows. The double ampersand forces namelst evaluation to wait until **&i** is evaluated into a number (e.g. 1 through 3). Then that number is tacked on to the namelst forming **namelst1**, **namelst2**, etc. The remaining ampersand covers the result. The remaining variable (**&namelst1**) is now evaluated, and its contents are concatenated to the contents of your macro variable. This, then, accomplishes our task. We now have a macro variable containing a list of files and another containing the number of files.

CONCLUSION

Now that you have a macro to get a list of files you can start putting it to use. Here is a hint: a macro do loop ranging from 1 to the number of files works very nicely with the %SCAN macro function for extracting words from a list. And a good project that makes a good first macro might use the CONTENTS procedure to document all those data sets you have sitting around in your SAS data set directory.

The macro will be furnished to those requesting copies by email.

REFERENCES

SAS Institute Inc. (1990), *SAS® Guide to Macro Processing, Version 6, Second Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1993), *SAS® Companion for the Microsoft Windows NT Environment, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

I would like to acknowledge my various employers whose tedious tasks drove me to learn the SAS macro language, my colleagues who braved my endless prattling about macros, and Barbara Widawski, without whose editing this manuscript would be illegible.

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION

Mel Widawski
Office of Academic Computing
UCLA
Los Angeles, CA

mel@ucla.edu