Paper 85-28

Make Your Life a Little Easier: A Collection of SAS® Macro Utilities

Pete Lund, Northwest Crime and Social Research, Olympia, WA

ABSTRACT

SAS® Macros are used in a variety of ways: to automate the generation of SAS code, to simulate functions and subroutines, or even to "comment" out a section of code. This paper focuses on another use of SAS macros: utilities.

These macros don't generate any SAS code and they aren't "functions," but they all take tasks I found myself doing repeatedly and bundled them in an easy to reference package.

These are examples range in scope from displaying SAS date values to retrieving information about external files, from getting the values of current options to getting a list of variables in a dataset. They are meant to be just that – examples. I hope that they will give you some ideas about how to automate some of the tasks you find yourself doing repeatedly.

A SAS DATE VIEWER MACRO

I can't believe that I'm the only one who's ever had some data with unformatted SAS dates and wanted to know quickly what 12843 was. Or, needed to know the SAS date value of 4/23/1998.

Obviously, if you're writing code or in a viewer you can add a format, or remove one, to do this. Sometimes you just need to quickly check the value – now! This little macro takes a SAS date or a "date" value (mm/dd/yy or ccyy) format and displays the other to the log.

For example, the following calls to the macro would produce, in the log window, the results shown.

```
%SASdate(-580)
Note- SAS date -580 is May 31, 1958

%SASdate(5/31/58)
NOTE- 5/31/58 is SAS date -580
```

Notice that there is only one macro to handle both input formats. The code to accomplish this is really quite simple.

```
%macro sasdate(dt);
  %if %index(&dt,%str(/)) eq 0 %then ❶
    %put SAS date &dt is       ❷
%sysfunc(left(%qsysfunc(putn(&dt,worddate.))));
  %else %put &dt is SAS date
%sysfunc(inputn(&dt,mmddyy10.)); ❸
%mend;
```

❶ First, we'll check to see if the value contains a slash (/). If it does not, we'll assume that it's a SAS date. All we need to do is format it as a date string ❷. That's accomplished with:

```
%qsysfunc(putn(&dt,worddate.))
```

By default, the text will be right justified in the default space allocated to WORDDATE. (18 characters). For this reason, we need to LEFT it with `%sysfunc(left(....))` to avoid unwanted spaces embedded in the log note.

Note: the nested %QSYSFUNC() is used, as opposed to %SYSFUNC, to mask the comma that the WORDDATE. format produces. If it were not masked the comma would be seen as a parameter delimiter by the LEFT function. Also note that in Version 8 there is a %LEFT macro function. I'll leave this macro the way it is so that it will function in 6.12 as well.

Finally, if the incoming value does have a slash, we assume that it's a date string and all we need do is convert that to a SAS date❸:

```
%sysfunc(inputn(&dt,mmddyy10.));
```

A simple little macro that does a simple little thing, but one that comes in handy very often.

A DATASET VARIABLE LIST MACRO

There are often times when I need a list of the variables in a dataset. For example, when merging two datasets I want to output of the records that matched but I also want to find out what's up with non-matches. So, I output two additional datasets with the non-matches from each of the two input datasets. A problem occurs when the input datasets have a lot of variables. I don't want to keep all the extraneous variables in my non-match datasets and I don't really want to maintain a long KEEP or DROP list either.

Wouldn't if be nice if there were an option to keep only the variables that correspond to a particular dataset? This macro does the trick. For example, I often merge arrest and booking data:

```
data matches
     OnlyBook(keep=%VarList(Booking))
     OnlyArrest(keep=%VarList(Arrest));
  merge Booking(in=inB)
        Arrest(in=inA);
  by BookingNumber;

  if inA and inB then output Matches;
  else if inA then output OnlyArrest;
  else output OnlyBook;
run;
```

The %VarList macro will return a list of the variables in the dataset referenced. So, my unmatched datasets will only contain the pertinent variables.

Note: the entire macro is displayed at the end of the paper. I'll only reference parts of it here to demonstrate it's operation.

Here's how it works. First, we'll "open" the dataset so we get information about it:

```
%let dsid = %sysfunc(open(&dsn,i));
```

The OPEN function "opens" the dataset and assigns an id that we'll store in the macro variable &dsid. Now, we'll start to build a macro variable, &VarList, that contains the names of all the variables in the dataset. To start the list we'll place the first variable name into &VarList using the VARNAME function.

```
%let VarList = %sysfunc(varname(&dsid,1));
```

Notice that we reference the dataset with the id (&dsid) that was assigned by the OPEN function. Now that the first variable name is loaded we can loop through the rest of the variables in the dataset and add their names to &VarList, separating the names with a space.

```
%do i = 1 %to %sysfunc(attrn(&dsid,nvars));
  %let VarList = &VarList ← blank space here
%sysfunc(varname(&dsid,&i));
%end;
```

The ATTRN function can return a number of attributes about the dataset. In this case we're interested in the number of variables, which is returned with the NVARS parameter.

Note: There are a couple things to be aware of in the real macro. First, before opening and processing the dataset a check is made to see that the dataset exists. If it does not, the bulk of the macro is not executed. Second, there is another parameter, Sep= (separator), that can be passed to the macro. The default separator is a space, but any character (or group of characters), can be passed as the variable name separator.

After we've read all the variable names we need to "close" the dataset using the CLOSE function:

```
%let rc = %sysfunc(close(&dsid));
```

At this point, if our Arrest dataset had four variables (age, race, sex, id), &VarList would equal:

age race sex id ← **separated by a space**

Finally, it's time to pass the value of &VarList out to the compiler that's waiting for it. The code for this is simple:

```
&VarList      ← notice, no semicolon
```

In the merge example above the value of &VarList would be placed into the KEEP option:

```
...(keep=%VarList(Arrest)); becomes →
...(keep=age race sex id);
```

We could also reference this in a macro environment (dataset name shortened to A):

```
%put The vars in A are %VarList(a).;
```

The above would produce the following in the log:

```
The vars in A are age race sex id.
```

Again, a relatively simple macro that can be used in a number of settings. A list of variables can be obtained in a number of other ways: PROC CONTENTS, DICTIONARY table COLUMN or SASHELP view VCOLUMNS in a datastep or PROC SQL query. But these methods all require SAS code to execute. The %VarList macro does not and can thus be used in a wider variety of applications. Also, it is very fast compared with the other methods.

KEEPING TRACK OF SAS OPTIONS

I often have macros or programs that change SAS options. Most of the time it would be nice to reset these options to their prior state so that subsequent code behave as you, or your users, expect.

For example, in most of my macros I'll turn the SAS notes off (with OPTIONS NONOTES) so as not to clutter my log. I could turn them back on again at the end – but, what if they were already off when I started?

This macro will place the current value of an option in a macro variable that can be used later to reset the original value.

```
%macro HoldOpt(OptName=, ❶
               OptValue=XX);

  %if &OptValue eq XX %then ❷
    %let OptValue = Hold&OptName;

  %global &OptValue; ❸

  %let &OptValue = ❹
     %sysfunc(getoption(&OptName)); ❺
%mend;
```

Note: a more complete listing of the macro is included at the end of the paper.

Let's look quickly at how this works. ❶ The only required parameter is the option name (OptName). A macro variable name to hold the option value can also be passed (OptValue).

❷If no macro variable name value is passed, and the value of &OptName is still "XX", the macro variable created is "Hold" plus the option name. For example, if you were going to get the current value of the line size option (LS) and did not specify a macro variable name, a variable called &HoldLS would be created.

❸The new variable needs to be made global so that it can be used later to reset the option.

Now all we need to do it create the macro variable. ❹Notice in the %LET statement that a macro variable reference is used:

```
%let &OptValue = ...
```

In our example above &OptValue = HoldLS, so this statement will resolve to

```
%let HoldLS = ...
```

❺Finally, we simply use the GetOption() function to get the current value of the option.

How might we use this? Let's say we had a program that has code that reads a number of datasets and generates a lot of log notes and has some output that requires a 132-line page. We can guarantee that the OBS option is set to max, that the LINESIZE is set to 132 and that the NOTES and SOURCE are turned off – and then we can set everything back to what it was.

```
%HoldOpt(OptName=notes)
%HoldOpt(OptName=obs)
%HoldOpt(OptName=ls)
%HoldOpt(OptName=source,OptValue=HoldSrc)

options nonotes obs=max nosource ls=132;
        :
        :  ← your SAS code here
        :
options &HoldNotes obs=&HoldObs
        &HoldSrc ls=&HoldLS;
```

All the options are now set back to their original values and we didn't even have to know what those values were.

GETTING INFO ON EXTERNAL FILES

A common situation is to have a process that runs on a regular basis and reads in a different raw data file each time, but with the same fileref. Wouldn't it be nice to place information about the file (size, date, name) in a title or footnote so that there's never a doubt as to the file that was used to create the dataset that any analysis was based on?

The following macro gets the name, size and date of an external file. A fileref is passed to the macro and global macro variables are created that contain the information.

Note: As with some of the other macros above, a trimmed down version is given below for discussion purposes. A more complete version, which includes some error handling and a little more user control, is included at the end of the paper. Also, the version presented here will run on Windows platforms. To run on UNIX platforms backslash references would need to be changed to

slashes and the method of accessing directory information would change (from dir to ls).

```
%macro FileInfo(FileRef);
 %global _FileSize _FileDate _FileName _Path;

 %let xpath =
   %sysfunc(pathname(&FileRef)); ❶

 %let stop =
   %index(%sysfunc(reverse(&xpath)),\); ❷
  %let _path =
    "%substr(&xpath,1,
            %eval(%length(&xpath)-&stop))";

  %let _FileName = %scan(&xpath,-1,\); ❸

  filename _temp pipe "dir %bquote(&_path)";❹

  data _null_;
    infile _temp lrecl=120 missover pad;

    length bigline $120;
    input bigline $120.;

    size = substr(bigline,25,14);  ❺
    date = substr(bigline,1,8);
    name = substr(bigline,40,60);

    if upcase(name) = upcase("&_FileName")❻
     then
      do;
        call symput("_FileSize", size);
        call symput("_FileDate", date);
      stop;
    end;
  run;

  filename _temp;  ❼
%mend;
```

For the following explanation, let's assume that we have the file *c:\monthly data\april.dat* referenced by the fileref *MnthData*.

The macro receives a fileref as its only parameter and uses the PATHNAME function to get the path and file name. ❶ The macro variable XPATH will equal *c:\monthly data\april.dat* – the complete path and file name.

❷ The next step is to break &XPATH into path name and file name pieces. We really know how far from the end of the &XPATH sting the path name ends by reversing the &XPATH and getting the position of the first backslash:

```
%index(%sysfunc(reverse(&xpath)),\)
```

The `%sysfunc(reverse(...))` in the above statement resolves to:

```
%index(tad.lirpa\atad ylhtnom\:c,\)
```

So, in our example result would be 10. We'll store this in &STOP and use this to create &PATH, holding the path name. The statement for creating &PATH now becomes:

```
"%substr(&xpath,1,%eval(%length(&xpath)-10))"
```

Notice that the value includes double quotes. This is important and will be discussed later.

❸ It's only a one step job now to extract the file name portion of &XPATH. It will always be the last segment of &XPATH if broken by backslashes. Version 8 makes this much easier to do with the addition of negative SCAN function. The "-1" tells scan to return the first segment from the end of the string – in other words, the last segment, which contains the file name.

Now that we know the path and file names, it's time to get some information about the file. We'll use the Windows directory command to get the size and date of the file. ❹ By using the PIPE option on the FILENAME statement we can access the results of an operating system command, in this case DIR, from a datastep.

The DIR command expects any path containing spaces to be enclosed in double quotes. This is why we put the quotes on the value of &PATH when we created it. However, consider:

```
filename _temp pipe "dir &_path"; ← resolves to
filename _temp pipe "dir "c:\monthly data"";
```

You can see that we have a problem here. The first quote around the path name is going to end the string and the FILENAME statement will fail. That's why we "quote" the value of &PATH with the %BQUOTE function. This removes any meaning from the quotes and they are just treated as a piece of text.

Imagine what you see when you go to a DOS prompt and type DIR. You see all the file information fly by on the screen. This is exactly what the input to our datastep will be when we reference this "piped" fileref in an INFILE

statement. Each line printed to the screen is an input line of data.

There will be a line of data about each file in the directory referenced containing the file name, creation date and time, size, etc. ❺ All we need do now is parse it into the pieces we want. An important note: the values listed here are the defaults for Windows NT. They are different for Windows 95/98. The actual macro has parameters for the start position and length of each element (size, date, name).

❻ When we come to the file we're interested in we will create macro variable containing the date and size of the file (we already have the name and path stored). We can stop the datastep now, since we're only looking for information about the one file.

❼ A final step is to clean up after ourselves and release the fileref. It's always nice to leave as little evidence as possible that you were there!

So, how might we use this? Let's look at an example:

```
%let Mnth  = April;
filename monthly "c:\monthly data\&Mnth..dat";

%FileInfo(monthly)

data &Mnth;
  infile monthly;
        :
        : ← file processing here
run;

title "&Mnth Data";
footnote1 "&Mnth file is &_Filename";
footnote2 "   Date: &_FileDate";
footnote3 "   Size: &_FileSize bytes";

proc ...; ← some analysis here
run;
```

Another way I've used this is to use the information in the label of the dataset being created. For example:

```
%let Mnth  = April;
filename monthly "c:\monthly data\&Mnth..dat";

%FileInfo(monthly)

data MyData.&Mnth
      (label="&_FileName from &_FileDate");
  infile monthly;
        :
        : ← file processing here
run;
```

Now, the file name and date are stored in the label of the dataset and I can be much more sure of the raw data used to create the dataset.

CONCLUSION

As you can see, these macros are related only in that they all take tasks that you could do "by hand" and automate it. I hope that this spurs you on to think of other things that you might let SAS do for you.

AUTHOR CONTACT

Pete Lund
Northwest Crime and Social Research
215 Legion Way SW
Olympia, WA 98501
(360) 528-8970
pete.lund@nwcsr.com
www.nwcsr.com

Electronic versions of these, and other, macros can be obtained via e-mail from the author.

TRADEMARK INFORMATION

COMPLETE MACRO TEXT IS DISPLAYED ON FOLLOWING PAGES

VARLIST Macro

```
/*************************************************************/
/* Macro:      GetVarList                                    */
/* Programmer: Pete Lund                                     */
/* Date:       September 2000                                */
/* Purpose:    Create a macro variable that contains all the */
/*    variable names in a dataset, separated by any char-    */
/*    acter(s) specified  Can be used in KEEPs, DROPs, SQL   */
/*    SELECTs, etc.                                          */
/*                                                           */
/* Parameters:                                               */
/*    DSN  - the name of the dataset (libref.member)         */
/*    Sep  - character(s) to separate the variable names     */
/*           (default is a space)                            */
/*************************************************************/

%macro VarList(dsn,
               sep=%str( ));

  %local i dsid varlist;

  /* Make sure that the dataset exists */

  %if %sysfunc(exist(&dsn)) eq 0 %then
    %do;
     %put WARNING: &DSN does not exist;
     %let varlist = ;
     %goto Quit;
    %end;

  /* If it does exist, open the dataset.  We're going to create */
  /* a variable (varlist) which contains the list of variables  */
  /* in the dataset, separated by the specified separator char- */
  /* acter(s).  Start VarList off with the name of the first    */
  /* variable in the dataset.                                   */

  %let dsid=%sysfunc(open(&dsn,i));
  %let varlist = %sysfunc(varname(&dsid,1));

  /* Look through the rest of the variables in the dataset and   */
  /* add them to the VarList value (with the separator char in   */
  /* between values.                                             */

  %do i = 2 %to %sysfunc(attrn(&dsid,nvars));
    %let varlist =  &varlist&sep%sysfunc(varname(&dsid,&i));
  %end;

  /* Close the dataset.                                          */

  %let rc = %sysfunc(close(&dsid));

  %Quit:

  /* Write out the variable name list.                           */

  &varlist
%mend;
```

SASDATE Macro

```
/*************************************************************/
/* Macro:      SASDate                                       */
/* Programmer: Pete Lund                                     */
/* Date:       June 1998                                     */
/* Purpose:    Returns the SAS date if a date (mm/dd/yy)     */
/*    value is passed.  Returns the date (mm/dd/yy) if a SAS */
/*    date value is passed.                                  */
/*                                                           */
/* Parameters:                                               */
/*    DT   - either a SAS date (numeric) value or a date     */
/*           (in mm/dd/yy or ccyy format)                    */
/*                                                           */
/*            %SASdate(-580)    returns                      */
/*                SAS date -580 is May 31, 1958              */
/*            %SASdate(5/31/58) returns                      */
/*                 5/31/58 is SAS date -580                  */
/*************************************************************/

%macro sasdate(dt);
  %if %index(&dt,%str(/)) eq 0 %then
    %put SAS date &dt is %sysfunc(left(%qsysfunc(putn(&dt,worddate.))));
  %else %put &dt is SAS date %sysfunc(inputn(&dt,mmddyy10.));
%mend;
```

HoldOpt Macro

```
/*************************************************************/
/* Macro:      HoldOpt                                       */
/* Programmer: Pete Lund                                     */
/* Date:       September 2000                                */
/* Purpose:    Holds the value of a SAS option in a macro    */
/*    variable.  The value can then be used to reset options */
/*    to a current value if changed.                         */
/*                                                           */
/* Parameters:                                               */
/*    OptName    - the name of the option to check           */
/*    OptValue   - the name of the macro variable that will  */
/*                 hold the current value of the option      */
/*                   The default name is made up of the word */
/*                   "Hold" and the option name.  For        */
/*                   example, if OptName=Notes, OptValue     */
/*                   would equal HoldNotes                   */
/*    Display    - Display current value to the log (Y/N)    */
/*                   The default is N                        */
/*                                                           */
/*************************************************************/



%macro HoldOpt(OptName=,      /* Option to check and hold value   */
               OptValue=XX,   /* Macro var name to hold value     */
               Display=N);    /* Display current value to the log */


  %if %substr(&sysver,1,1) eq 6 and %length(&OptName) gt 4 %then
    %do;
      %put WARNING: Default variable name of Hold&OptName is too long for V&sysver..;
      %put WARNING: Please specify a shorter name with the OptValue= macro parameter.;
      %goto Quit;
    %end;
  %if &OptValue eq XX %then %let OptValue = Hold&OptName;

  %global &OptValue;

  %let &OptValue = %sysfunc(getoption(&OptName));

  %if &Display eq Y %then
   %put The current value of &OptName is &&&OptValue;

  %Quit:
%mend;
```

FileInfo Macro

```
/*****************************************************************************/
/* Macro:      FileInfo                                                      */
/* Programmer: Pete Lund                                                     */
/* Date:       September 2000                                                */
/* Purpose:    Gets information about an external file (path, file name, date and  */
/*    size) based on a fileref.                                              */
/*                                                                           */
/* Parameters:                                                               */
/*    FileRef - the fileref of the file for which information in desired     */
/*    S,D,N   - the position-size of the Size, Date and Name portions of the  */
/*              DIR command data lines.  The default values are for Windows NT.  */
/*              For Windows 98, the values would be: S=15-12,D=29-8,N=45-60  */
/*    FS,FD,FN,FP - The names of the macro variables that will contain the file  */
/*              size, date, name and path.  Defaults are _FileSize, _FileDate,  */
/*              _FileName and _Path.                                         */
/*****************************************************************************/


%macro FileInfo(FileRef,S=25-14,D=1-8,N=40-60,
                FS=_FileSize,FD=_FileDate,FN=_FileName,FP=_Path);
  %global &FS &FD &FN &FP;
  %local stop path xpath ErrMsg;

  /* Initialize all the global variables */

  %let ErrMsg = ;
  %let _FileSize = (unknown);
  %let _FileDate = (unknown);
  %let _FileName = (unknown);

  /* Check to see if the fileref passed really exists and points to */
  /* a real file.  If not, create an error message and jump out of  */
  /* the macro.                                                     */

  %if %sysfunc(fileref(&FileRef)) gt 0 %then
    %do;
      %let ErrMsg = ERROR: Fileref "&FileRef" has not been assigned;
      %goto Quit;
    %end;
  %else %if %sysfunc(fileref(&FileRef)) lt 0 %then
    %do;
      %let ErrMsg = ERROR: Fileref "&FileRef" references a nonexistent file;
      %goto Quit;
    %end;

  /* Grab the values for NOTES and XWAIT options so that they can be */
  /* reset at the end.                                               */

  %HoldOption(OptName=notes)
  %HoldOption(OptName=xwait)

  options nonotes noxwait;

  /* Get the path and filename of the passed fileref                 */

  %let xpath = %sysfunc(pathname(&FileRef));
```

```
  /* If the path is in parenthesis it is (probably) a concatenated path */
  /* and I don't want to deal with those.  Note this and get out.       */

  %if %qsubstr(&xpath,1,1) eq %str(%() %then
    %do;
      %let ErrMsg = ERROR: FILEINFO macro does not support concatenated path filerefs;
      %goto Quit;
    %end;

  /* Find the position of the last backslash in the path name.  This will */
  /* be used to split out the path and file portions of the name.         */
  /* Actually, now we can use a negative number in the SCAN function to   */
  /* get a chunk at the end of a string.  Use this to get the filename.   */

  /* Note that the path is enclosed in double quotes.  The DIR command,   */
  /* used below, requires that paths with spaces be in double quotes.     */

  %let stop = %index(%sysfunc(reverse(&xpath)),\);
  %let _path = "%substr(&xpath,1,%eval(%length(&xpath)-&stop))";
  %let _FileName = %sysfunc(scan(&xpath,-1,\));

  /* Pipe a directory listing - need to blind quote the path name so that  */
  /* double quotes in the path don't screw things up.                      */

  filename _temp pipe "dir %bquote(&_path)";

  /* Read the directory where the file resides - when the file is found,   */
  /* grab the size and date (we already have the name) and write out macro */
  /* variables containing the values.  We can stop at this point since we  */
  /* only looking for information about one file.                          */

  data _null_;
    infile _temp lrecl=120 missover pad;

    length bigline $120;
    input bigline $120.;

    size = substr(bigline,%scan(&S,1,-),%scan(&S,2,-));
    date = substr(bigline,%scan(&D,1,-),%scan(&D,2,-));
    name = substr(bigline,%scan(&N,1,-),%scan(&N,2,-));

    if upcase(name) eq upcase("&_FileName") then
      do;
        call symput("_FileSize",trim(left(size)));
        call symput("_FileDate",trim(left(date)));
        stop;
      end;
  run;

  /* Clean things up - clear the fileref and reset the NOTES and XWAIT options */

  filename _temp;
  options &HoldNotes;
  options &HoldXWait;

  %Quit:
    %put &ErrMsg;
%mend;
```