Paper 004-30

Journeymen's Tools: Two Macros — ProgList and PutMvars — to Show Calling Sequence and Parameters of Routines

Ronald Fehd, SAS-L's macro maven
Centers for Disease Control, and Prevention, Atlanta GA USA

ABSTRACT

A program that is a job may define parameters for and call other programs which are routines. SAS® routines and subroutines may be implemented as %includes or macros. Routines call other routines and subroutines. When testing or debugging multi-level %includes, it is difficult to determine which parameter in which routine or subroutine produced an error or warning message in the log.
This paper discusses options used for testing, provides a new option, shows subroutine control using session constants, and illustrates two macros to show parameters and calling sequence of routines. Expected audience is intermediate to advanced programmers with experience using %includes with parameters.

Keywords: include fan-in fan-out job oplist parameter.

Contents

Introduction **1**
Discussion and Review 2
Example Jobs 2
Testing Sequence 3
Note-1: 4
Note-2: 4
Note-3: 4
Conclusion **5**
Suggested Reading 5
ProgList.sas 6
PutMvars.sas 7
TestSuite Logs of ProgList and PutMvars 8

INTRODUCTION

SAS software provides the option `source2` to echo statements contained in %included programs. When programs contain many levels of included programs, testing and debugging — i.e., locating which program called which other program to locate where the error or warning occurred — takes valuable time. Macros `ProgList` and `PutMvars` were developed as testing tools to show the complete calling sequence of jobs calling multiple routines and subroutines as %includes and the parameters of each routine or subroutine.

Macro `ProgList` writes the calling sequence of included programs to the log. `ProgList` uses a global macro variable named ProgList, to contain the job name and all its called routines. `ProgList` is called in one of three ways: at the top of the job where it initializes the global macro variable ProgList with the job name; and within each called routine or subroutine: at the top and bottom of the program. At the top it appends the provided name to the list. At each call the macro variable ProgList is written to the log. When exiting a routine, the current program name is removed from the list. `ProgList` may be turned on by 1. invocation options `-oplist` or `-verbose` or 2. within a session, either `options nosource2` or `%Let TestSite=1`.

After determining which routine is having problems, testing proceeds by examining the parameters provided to each routine. Macro `PutMvars` writes the parameter list used by included routines to the log. `PutMvars` may be turned on by 1. invocation options `-verbose` or 2. within a session:, `%Let TestSite=1`.

Routines may call other routines and subroutines. Routines usually contain statements to perform a specific task; they may call other routines to accomplish some parts of their task. Subroutines contain statements to perform a general task. Since they may be called many times by any set of routines, we wish to have a separate method of turning off their echo of statements to the log. Macro variable `Source2` is used to turn off the echo of subroutines, even while `options source2`, is used.

DISCUSSION AND REVIEW

Brooks [Bro95] observes: "My rule of thumb is 1/3 of the schedule for design, 1/6 for coding, 1/4 for component testing, and 1/4 for system testing." That is, half of a programmer's time is spent in testing. Let us see what testing involves and how to reduce time spent scanning program logs.

SAS jobs may use two methods of replicating statements: %includes or macros. Each of these program types may have parameters. Each method has specific options that can be used to echo or hide the contained statements.

include with parameters

```
1 options source2;
2 %Let ParmA = x;
3 %Let ParmB = y;
4 %Inc Pgm(DoThat);
```

macro with parameters

```
1 options mprint;
2 %DoThis(ParmA = x
3        ,ParmB = y
4        );
```

`Source2` is both a system option and an include statement option. The %include statement option overrides the system option.

using Source2

```
1 *echo statements to log;
2 options Source2;
3 %Inc Pgm(DoThat);
4
5 options noSource2;
6 %Inc Pgm(DoThat) /Source2;
```

using noSource2

```
1 *no echo;
2 options noSource2;
3 %Inc Pgm(DoThat);
4
5 options Source2;
6 %Inc Pgm(DoThat)/noSource2;
```

Macro echo can be turned on and off with `option mprint;` Tracing macro calls is straightforward since the contributing macro name is at the beginning of each line.

macro Array test suite with Mprint

```
1 options mprint;
2 %MACRO Demo(Test = 0, Dim_Var = 10, Nobs = 0);
3 %If &Test. eq 2 %then %do;%Nobs(Dim_Var, Data=V04);  run;
4                           %Array(Var,Data=Vq04,Var=Name);
5                        %end;run;                    %Mend;
```

In the log at the position of an error or warning, one can scan back up the log to see the name of the calling macro. Here the calling sequence is `Demo:Nobs`, then `Demo:Array`.

example macro log with Mprint

```
1 309       +%Demo(Test = 2);
2 MPRINT(NOBS):   run;
3 @@Nobs: "Dim_Var"=<3> data=Vq04
4 MPRINT(DEMO):  ;
5 MPRINT(DEMO):   run;
6 MPRINT(ARRAY):   PROC SQL noPrint;
7 MPRINT(ARRAY):   select Name into :Var1 - :Var999 from V04;
8 MPRINT(ARRAY):  quit;
```

Tracing include usages is similar with `options source2;`

example Include

```
1 options source2;
2 %Inc Pgm(exec_if1);*contains %Inc Pgm(Exec_Text);
```

which writes notes to the log of the file currently providing statements. Note in lines 7 and 8 that the names of the called and calling programs are not given. I developed ProgList to overcome this lack of information.

example Include log

```
1 89        +%Inc Pgm(exec_if1);*contains %Inc Pgm(Exec_Text);
2 NOTE: %INCLUDE (level 1) file PGM(exec_if1) is file
3                          C:\temp\exec_if1.sas.
4 153       + %Inc Pgm(Exec_test);
5 NOTE: %INCLUDE (level 2) file PGM(Exec_test) is file
6                          is file \exec_test.sas.
7 NOTE: %INCLUDE (level 2) ending.
8 NOTE: %INCLUDE (level 1) resuming.
```

When tracing multi-level includes, the names of the calling and called programs are needed. In large logs of thousands of lines, these two items of information can be many pages apart.

EXAMPLE JOBS

A job is a program in a project that accomplishes some particular task, for instance, a specific report. Every project has an autoexec; when not testing, the two macro variables `Source2` and `TestSite` are set off. Note the asterisks in column 1 of lines 2 and 4 which disable the testing values.

project autoexec

```
1 %Let Source2  = noSource2;*hide;
2 *Let Source2  =   Source2;*echo;
3 %Let TestSite = 0;        *testing: false;
4 *Let TestSite = 1;        *testing: true;
```

After developing and testing a particular report I may combine them, this effect is called fan-out: one job or routine calling many different programs.

Job 0

```
1 Options noNotes noSource
2                 noSource2;
3 %Inc Pgm(Job1);
4 %Inc Pgm(Job2);
5 *...     Jobs;
6 %Inc Pgm(JobN);
```

A program that is a job depends on having an autoexec and is distinguished by its definition of parameters and calling of routines. Any of the %included jobs in `Job 0` may look like this example `Job 1`:

Note that echo of statements in the routines called in lines 9 and 13 is controlled by the `options source2` statement in line 5. When not testing, the statement is disabled with an asterisk; the option may be turned on by either of the calling job, `Job 0`, or removing the asterisk in `Job 1`, line 5, when submitting this job alone.

Job 1

```
1  %Let       ProgName = Job1;
2  title2   "&ProgName.";
3  %ProgList(&ProgName., action = init);
4  *Let TestSite = 1;
5  *Options source2;
6  *...;
7  %Let        ParmM1 = a;
8  %Let        ParmM2 = b;
9  %Inc Pgm(RoutineM);
10 *...;
11 %Let        ParmN1 = c;
12 %Let        ParmN2 = d;
13 %Inc Pgm(RoutineN);
14 *...;
```

Any program that is a routine does a specific task. It may call other routines and subroutines to realize that task. It cannot execute without its parameters being defined by another calling program. Any of the included routines called by any jobs in `Job 0` may look like this example `Routine M`.

Note that the calls to `SubRtnX` in lines 7 and 11 will not be echoed to the log because the value of the macro variable `Source2` is `noSource2`. This overrides the setting of the option `Source2` set by its calling job.

Routine M

```
1  %ProgList(RoutineM);
2  %PutMvars(ParmM1 ParmM2);
3  *process ParmM1 and ParmM2
4   which result in values for:;
5  %Let       ParmX1 = e;
6  %Let       ParmX2 = f;
7  %Inc Pgm(SubRtnX) /&Source2.;
8  *...;
9  %Let       ParmX1 = g;
10 %Let       ParmX2 = h;
11 %Inc Pgm(SubRtnX) /&Source2.;
12 *...;
13 %ProgList(action = exit);
```

Any program that is a subroutine is like a routine except it does not call any other programs. A subroutine does a common general task, and is called by many other routines; this effect is called fan-in: many routines calling the same subroutine. Any of the included subroutines called by any of the jobs or routines in `Job 0` may look like this example `subRoutine X`.

subRoutine X

```
1  %ProgList(SubRtnX);
2  %PutMvars(ParmX1 ParmX2);
3  *fan-in: do something useful
4           for many routines here
5           how about Nobs?;
6  %ProgList(action = exit);
```

Note: While `Job 1` and `Routine M` show parameters assigned with the macro variable `%Let` statements, in practice macro variables may also be derived from data variable values using either `call symput('Mvar',<value>)` or `Proc SQL; select <value> into :Mvar.`

TESTING SEQUENCE

Let us examine our `Job 0` log. Even with `options noNotes noSource noSource2`, we see in line 5 that error messages get written to the log. Each line begins with the special characters `@@` in order to identify it as produced by the two tracing subroutines. We can see that the error was in `Job 1` and `Routine M`, and that it happened after the call to and exit from `SubRtnX`.

Job 0 example log

```
1  @@begin Job1
2  @@enter Job1:RoutineM
3  @@enter Job1:RoutineM:SubRtnX
4  @@exit  Job1:RoutineM:SubRtnX
5  ERROR: badness happening
6  @@exit  Job1:RoutineM
7  @@enter Job1:RoutineN
8  @@enter Job1:RoutineN:SubRtnX
9  @@exit  Job1:RoutineN:SubRtnX
10 @@exit  Job1:RoutineN
11 @@end   Job1
12 @@begin Job2
```

We return to Job1 and replace the asterisk at the beginning of line 4 with a percent sign which assigns `TestSite` the value of one. This turns on macro `PutMvars` which will write the parameters of each routine and subroutine to the log.

Job 1 test 1: turn on PutMvars

```
1  %Let       ProgName = Job1;
2  title2   "&ProgName.";
3  %ProgList(&ProgName., action = init);
4  %Let TestSite = 1;
5  *Options source2;
```

Let us examine our `Job 1` log.

COBOL programmers remember that the error is usually on the previous line. I use the the following quip in my SAS-L signature:
Remember perspective: the error is not always where it seems to occur! Often a log showing the parameter values is enough to indicate to the experienced programmer where the error is actually occuring. Otherwise . . .

We return to `Job 1` and remove the asterisk at the beginning of line 5 which changes the option `Source2` so that %included statements of routines will be echoed to the log. Subroutine statements are still hidden because their %includes have the macro variable `Source2` set to `noSource2`. The log gets long at this point so no further example is provided.

Note-1: Both `ProgList` and `PutMvars` use the global macro variable `TestSite` to reset their local macro variable `Testing`:
`%Let Testing = %eval(&Testing or &TestSite).`
See `ProgList`, line 44 and `PutMvars`, line 42.
One could remove this global macro variable reference within each of the macros and then add the particular testing macro variable assignment to each macro call.

Note-2: Another way to turn on these trace subroutines is to use the invocation options `oplist` or `verbose`. Both write invocation command line options settings to the log during startup. In addition `verbose` writes options specified in any and all configuration files used during startup. Neither of these options have further effects during a session or batch job.

Note-3: An Alert Reader suggests that `PutMvars` could be used at the end of a routine or sub-routine to list the values assigned or calculated by that (sub-) routine.

Job 1 test 1: log showing parameters

```
1  @@enter Job1:RoutineM
2  @@parms: ParmM1<a>
3  @@parms: ParmM2<b>
4  @@enter Job1:RoutineM:SubRtnX
5  @@parms: ParmX1<g>
6  @@parms: ParmX2<h>
7  @@exit  Job1:RoutineM:SubRtnX
8  ERROR: badness happening
9  @@exit  Job1:RoutineM
10 ...
```

Job 1 test 2: PutMvars and Source2

```
1  %Let       ProgName = Job1;
2  title2   "&ProgName.";
3  %ProgList(&ProgName., action = init);
4  %Let TestSite = 1;
5   Options source2;
```

Job 9: alternate use of global mvars

```
1  %Let ShowIncs  = 0;
2  %Let ShowParms = 0;
3  %ProgList(&ProgName., action = init
4           ,testing = &ShowIncs.);
5  %ProgList(RoutineQ
6           ,testing = &ShowIncs.);
7  %PutMvars(ParmQ1 ParmQ2
8           ,testing = &ShowParms.);
9  %ProgList(action  = exit
10          ,testing = &ShowIncs.);
```

invocation: use of session constants

```
1  sas -sysin JobX -oplist
2
3  sas -sysin JobX -verbose
```

subRoutine X shows its return values

```
1  %ProgList(SubRtnX);
2  %PutMvars(ParmX1 ParmX2);
3  *...;
4  %PutMvars(XreturnsA XreturnsB);
5  %ProgList(action = exit);
```

CONCLUSION

Macros ProgList and PutMvars provide easy-to-use testing tools for tracing routine and subroutine calls. Each requires only a few lines of code added to a program. Both are unobtrusive and can be turned on with either `%Let TestSite = 1`, or invocation and session options. These two tools make tracing, testing, and locating the source of errors and warnings in parameterized %include programs faster and easier.

REFERENCES

[Bro95] Brooks, Frederick P., Jr. (1995), *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, 2/E,* Addison-Wesley.
`http://www.aw-bc.com/catalog/academic/product/0,,0201835959,00%2ben-USS_01DBC.html`
`http://www.amazon.com/exec/obidos/tg/detail/-/0201835959/002-4796015-3029601?v=glance`

SUGGESTED READING

The program header information used in each of the programs ProgList.sas and PutMvars.sas is described in this paper:

Fehd, Ronald (2005), *Journeymen's Tools: The Writing for Reading and Reuse Program Header*, Proceedings of the 30th Annual SAS® Users Group International Conference, 2005.

Acknowledgments I would like to thank Toby and Sara Dunn for their critique and encouragement. Dianne Rhodes also provided valuable commentary and occasionally whispered Klingon or SQL to me.

To receive the latest edition of these macros ProgList, and PutMvars, send an e-mail to the author with the subject:
`request ProgList and PutMvars`

Author: Ronald Fehd **`mailto:Ronald.Fehd@cdc.hhs.gov`**
Centers for Disease Control MS-G23 **e-mail: RJF2@cdc.gov**
4770 Buford Hwy NE
Atlanta GA 30341-3724 **bus: 770/488-8102**

Document Production: This paper was typeset in LaTeX. For further information about using LaTeX to write your SUG paper, consult the SAS-L archives:

```
http://www.listserv.uga.edu/cgi-bin/wa?S1=sas-l
Search for                  :
The subject is or contains: LaTeX
The author's address        : RJF2
Since                       : 01 June 2003
```

ProgList.sas

ProgList.sas

```
 /*      name: ProgList.sas
-------------: User Requirements:
      purpose: tool to trace calls of %include usage
  description: write note to log of %global ProgList
-------------: Program Specifications:
program group: development and testing
program  type: subroutine
    SAS  type: macro procedure
         note: turned on by either:    global mvar %Let TestSite = 1;
                                    or             options nosource2;
                                    or invocation -oplist
                                    or invocation -verbose
        input: program name, action, global mvar TestSite
      process: action eq INIT: assign ProgName  to   %global ProgList
               action eq SAVE: append ProgName  to   %global ProgList
                               write ProgList
               action eq EXIT: remove last item from %global ProgList
       output: note in log
         note: side effect: changes global mvar ProgList
         note: see also PutMvars
       author: Ronald.Fehd@cdc.hhs.gov
usage:         %Let TestSite = 1;%*in autoexec;
%ProgList(JobX,action=init);*in a job which calls routines;
options nosource2;
%Inc Pgm(RoutineA);
* - - - - RoutineA.sas - - - *;
%ProgList(RoutineA);   *at top of called routine/subroutine;
*many other statements;
%ProgList(action=exit);*at end of called routine/subroutine;
****** ******** ..................... */
%MACRO ProgList
         (Program
         ,Action  = Save /*init or exit            */
         ,Testing = 0    /*TestSite set in autoexec*/
)/des="write note to log of program calls &SysDate."
 /* ** store source /* must have options mStored SASmStore=libref */
;/*change notes:
RJF2 03Nov07 written
RJF2 05Feb04 polishing for SUGI30
******* ********* ..................... */
%global ProgList;%*returned value: side effect;
%local  Pfx;     %Let Pfx    = @@; %*prefix of log message;
%local  Sep_By;  %Let Sep_By =  :; %*separated by;
                   /*NOTE: global TestSite*/
%Let Testing = %eval(&Testing or &TestSite
                        or %sysfunc(getoption(source2)) eq NOSOURCE2
                        or %sysfunc(getoption(oplist))  eq   OPLIST
                        or %sysfunc(getoption(verbose)) eq   VERBOSE );
%If &Testing              %then %do; %Let Action   = %upcase(&Action.);
%If       &Action eq INIT %then %do;
    %If &Proglist ne      %then      %Put &Pfx.end   &ProgList.;
                                     %Let ProgList = &Program.;
                                     %Put &Pfx.begin &ProgList.;   %end;
%Else %If &Action eq SAVE %then %do; %Let ProgList =
                                         &ProgList.&Sep_By.&Program.;
                                     %Put &Pfx.enter &ProgList.;   %end;
%Else %If &Action eq EXIT %then %do; %Put &Pfx.exit  &ProgList.;
                                     %*remove last programName;
                                     %Let ProgList =%substr(&ProgList.,1
                           ,%eval(  %length(&ProgList.)
                                  - %index(%sysfunc(reverse(&ProgList.))
                                           ,&Sep_By.) ));           %end;
                                %end;%*%If &Testing ... ProgList; %Mend;
```

PutMvars.sas

PutMvars.sas

```
 /*      name: PutMvars.sas
-------------: User Requirements:
      purpose: tool to review parameters of includes
  description: write macro variable names and values to log
-------------: Program Specifications:
program group: development and testing
program  type: subroutine
    SAS  type: macro procedure
         note: uses SAS-supplied macro function cmpres
         note: turned on by global mvar %Let TestSite = 1;
                    or invocation -verbose
        input: list of macro variables, global mvar TestSite
      process: read each Mvar in list, put name and value
       output: to log
         note: see also ProgList
       author: Ronald.Fehd@cdc.hhs.gov
       usage:
%Let TestSite  = 0;%*off, set in autoexec, or job;
%Let TestSite  = 1;%*on;
%PutMvars(list = SysDate SysTime
         ,Msg  = text);
%Let TestParms = 1;*alternate use: show mvar used to turn on;
%PutMvars(list = SysDate SysTime
         ,testing = &TestParms.);
****** ******** ..................... */
%MACRO PutMvars
         (List    =        /*of macro variables*/
         ,Msg     = parms: /*infix             */
         ,Testing = 0      /*write to log?     */
)/des = "write list of mvars + values to log &SysDate."
 /*store source /*must have options mStored SASmStore=libref*/
;/* change log ------------------------- **
RJF2 03Nov05 written
RJF2 05Feb04 polishing for SUGI30
%If &SysVer. ge 9.1 %then %Let Testing =
                        %eval(&Testing or %symchkif(TestSite));
****** ****** ..................... */
%local I     ; %Let I      =  1; %*loop index;
%local Item  ; %Let Item   =   ; %*Mvar Name to process;
%local Pfx   ; %Let Pfx    = @@; %*prefix of log message;
%local nItems; %Let nItems =  1; %*items in list = N(spaces) +1;
%local Width ; %Let Width  =  1; %*max width of item;
                       /*NOTE: global TestSite*/
%Let Testing = %eval(&Testing or /**&TestSite);**/
                            %symchkif(TestSite)
                            or %sysfunc(getoption(verbose)) eq VERBOSE );

%If &Testing. and &List ne %then %do;   %***how many Items in List?;
    %Let List   = %cmpres(&List);       %***change 2+ spaces to 1 space;
    %*** nItems = number of spaces between items plus one;
    %Let nItems = %eval(1 + %length(&List)
                          - %length(%sysfunc(compress(&List))));
    %*** to align values in column <...> need:;
%Do I = 1 %to &nItems.; %Let Width = %sysfunc(max(&Width
                                    ,%length(%scan(&List.,&I.)))); %end;

%Do I = 1 %to &nItems.; %Let Item  = %scan(&List.,&I.);
    %***string is 32 chars long:           +    1    +    2    +    3  ;
    %***note: dot leader       :       12345678901234567890123456789012;
    %Put &Pfx.&Msg. %substr(&Item.%str( ). . . . . . . . . . . . . . . *
                            ,1,&Width.) <&&&Item.>;                 %end;
                                 %end; %*If &Testing ... PutMvars; %Mend;
```

TESTSUITE LOGS OF PROGLIST AND PUTMVARS

TestSuite: ProgList.log

```
198       +*** TestSuite of Filename<proglist>;
199       +*** SysDate<13JUL04> SysTime<08:41> SAS version<9.1>;
200       +%Let TestSite = 1;
201       +%ProgList(zTest,action=init);
@@begin zTest
202       +%ProgList(RoutineA);
@@enter zTest:RoutineA
203       +%ProgList(RoutineB);
@@enter zTest:RoutineA:RoutineB
204       +%ProgList(SubRtnC);
@@enter zTest:RoutineA:RoutineB:SubRtnC
205       +%ProgList(action = exit);
@@exit  zTest:RoutineA:RoutineB:SubRtnC
206       +%ProgList(action = exit);
@@exit  zTest:RoutineA:RoutineB
207       +%ProgList(SubRtnC);
@@enter zTest:RoutineA:SubRtnC
208       +%ProgList(action = exit);
@@exit  zTest:RoutineA:SubRtnC
209       +%ProgList(action = exit);
@@exit  zTest:RoutineA
210       +%ProgList(zTestNext,action=init);
@@end   zTest
@@begin zTestNext
```

TestSuite: PutMvars.log

```
187       +*** TestSuite of Filename<putmvars>;
188       +*** SysDate<13JUL04> SysTime<09:06> SAS version<9.1>;
189       +%Let TestSite  = 0;%*off;
190       +*before;
191       +%PutMvars(list = SysDate SysTime);
192       +*after: no note written to log;
193       +%Let TestSite  = 1;%*on;
194       +%PutMvars(list = SysDate SysTime TestSite);
@@parms: SysDate  <13JUL04>
@@parms: SysTime  <09:06>
@@parms: TestSite <1>
195       +%Let TestSite  = 0;
196       +%Let TestParms = 1;
197       +%Let aLongMacroVarName = x;
198       +%PutMvars(list= TestParms aLongMacroVarName
199       +         ,msg = ThisRoutineName parameters
200       +         ,testing = &TestParms.);
@@ThisRoutineName parameters TestParms . . . . <1>
@@ThisRoutineName parameters aLongMacroVarName <x>
```