

Journeyman'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

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.

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

```
1  macro with parameters
2  options mprint;
3  %DoThis(ParmA = x
4  ,ParmB = y
5  );
```

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:

```

_____ 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 *...;

```

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.

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.

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.

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.

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.

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 occurring. 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.

```

----- 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);

```

```

----- 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);

```

```

----- 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

```

```

----- 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;

```

```

----- 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;

```

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 sub-routines 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.

```
----- 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
```

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>

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**

End note: 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

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

```

1      /*      name: ProgList.sas
2      program group: development and testing
3      program type: subroutine
4          SAS type: macro procedure
5          purpose: tool to trace calls of %include usage
6          note: turned on by either:      global mvar %Let TestSite = 1;
7              :                          or                          options nosource2;
8              or invocation -oplist
9              or invocation -verbose
10         description: write note to log of %global ProgList
11         input: program name, action, global mvar TestSite
12         process: action eq INIT: assign ProgName to %global ProgList
13                 action eq SAVE: append ProgName to %global ProgList
14                 write ProgList
15                 action eq EXIT: remove last item from %global ProgList
16         output: note in log
17         note: side effect: changes global mvar ProgList
18         note: see also PutMvars
19         author: Ronald.Fehd@cdc.hhs.gov
20         usage:
21 %Let TestSite = 1;%*in autoexec;
22 %ProgList(JobX,action=init);*in a job which calls routines;
23 options nosource2;
24 %Inc Pgm(RoutineA);
25 * - - - - RoutineA.sas - - - *;
26 %ProgList(RoutineA); *at top of called routine/subroutine;
27 *many other statements;
28 %ProgList(action=exit);*at end of called routine/subroutine;
29 ***** ***** ..... */
30 %MACRO ProgList
31     (Program
32     ,Action = Save /*init or exit */
33     ,Testing = 0 /*TestSite set in autoexec*/
34 )/des="write note to log of program calls &SysDate."
35 /* ** store source /* must have options mStored SASmStore=libref */
36 ;/*change notes:
37 RJF2 03Nov07 written
38 RJF2 04Jun07 polishing for fall SUG papers
39 ***** ***** ..... */
40 %global ProgList;.*returned value: side effect;
41 %local Pfx; %Let Pfx = @@;.*prefix of log message;
42 %local Sep_By; %Let Sep_By = ;.*separated by;
43 /*NOTE: global TestSite*/
44 %Let Testing = %eval(&Testing or &TestSite
45                 or %sysfunc(getoption(source2)) eq NOSOURCE2
46                 or %sysfunc(getoption(oplist)) eq OPLIST
47                 or %sysfunc(getoption(verbose)) eq VERBOSE );
48
49 %If &Testing %then %do; %Let Action = %upcase(&Action.);
50 %If &Action eq INIT %then %do;
51     %If &Proglst ne %then %Put &Pfx.end &ProgList.;
52     %Let ProgList = &Program.;
53     %Put &Pfx.begin &ProgList.; %end;
54 %Else %If &Action eq SAVE %then %do; %Let ProgList =
55     &ProgList.&Sep_By.&Program.;
56     %Put &Pfx.enter &ProgList.; %end;
57 %Else %If &Action eq EXIT %then %do; %Put &Pfx.exit &ProgList.;
58     %*remove last programName;
59     %Let ProgList =%substr(&ProgList.,1
60     ,%eval( %length(&ProgList.)
61     - %index(%sysfunc(reverse(&ProgList.))
62     ,&Sep_By.)); %end;
63     %end;%*If &Testing ... ProgList; %Mend;

```

```

1  /*      name: PutMvars.sas
2  program group: development and testing
3  program type: subroutine
4      SAS type: macro procedure
5      note: uses SAS-supplied macro function cmpres
6      purpose: tool to review parameters of includes
7      note: turned on by global mvar %Let TestSite = 1;
8              or invocation -verbose
9      description: write macro variable names and values to log
10     input: list of macro variables, global mvar TestSite
11     process: read each Mvar in list, put name and value
12     output: to log
13     note: see also ProgList
14     author: Ronald.Fehd@cdc.hhs.gov
15     usage:
16 %Let TestSite = 0;%*off, set in autoexec, or job;
17 %Let TestSite = 1;%*on;
18 %PutMvars(list = SysDate SysTime
19     ,Msg = text);
20 %Let TestParms = 1;*alternate use: show mvar used to turn on;
21 %PutMvars(list = SysDate SysTime
22     ,testing = &TestParms.);
23 ***** ***** ..... */
24 %MACRO PutMvars
25     (List      =          /*of macro variables*/
26     ,Msg       = parms: /*infix          */
27     ,Testing   = 0       /*write to log?   */
28 )/des = "write list of mvars + values to log &SysDate."
29 /*store source /*must have options mStored SASmStore=libref*/
30 /* change log ----- **
31 RJF2 03Nov05 written
32 RJF2 04Jun08 polishing for fall SUGs
33 %If &SysVer. ge 9.1 %then %Let Testing =
34     %eval(&Testing or %symchkif(TestSite));
35 ***** ***** ..... */
36 %local I      ; %Let I      = 1; %*loop index;
37 %local Item  ; %Let Item   = ; %*Mvar Name to process;
38 %local Pfx   ; %Let Pfx   = @@; %*prefix of log message;
39 %local nItems; %Let nItems = 1; %*items in list = N(spaces) +1;
40 %local Width ; %Let Width  = 1; %*max width of item;
41              /*NOTE: global TestSite*/
42 %Let Testing = %eval(&Testing or /**&TestSite);**/
43              %symchkif(TestSite)
44              or %sysfunc(getoption(verbose)) eq VERBOSE );
45
46 %If &Testing. and &List ne %then %do; %**how many Items in List?;
47     %Let List = %cmpres(&List); %**change 2+ spaces to 1 space;
48     %** nItems = number of spaces between items plus one;
49     %Let nItems = %eval(1 + %length(&List)
50         - %length(%sysfunc(compress(&List))));
51     %** to align values in column <...> need;
52 %Do I = 1 %to &nItems.; %Let Width = %sysfunc(max(&Width
53     ,%length(%scan(&List.,&I.)))); %end;
54
55 %Do I = 1 %to &nItems.; %Let Item = %scan(&List.,&I.);
56 %**string is 32 chars long:          + 1 + 2 + 3 ;
57 %**note: dot leader          :      12345678901234567890123456789012;
58 %Put &Pfx.&Msg. %substr(&Item.%str( ) . . . . . *
59     ,1,&Width.) <&&Item.>; %end;
60 %end; %*If &Testing ... PutMvars; %Mend;

```

TestSuite Logs of ProgList and PutMvars

```
TestSuite: ProgList.log  
1 198      +*** TestSuite of Filename<proglis>;  
2 199      +*** SysDate<13JUL04> SysTime<08:41> SAS version<9.1>;  
3 200      +%Let TestSite = 1;  
4 201      +%ProgList(zTest,action=init);  
5 @@begin zTest  
6 202      +%ProgList(RoutineA);  
7 @@enter zTest:RoutineA  
8 203      +%ProgList(RoutineB);  
9 @@enter zTest:RoutineA:RoutineB  
10 204      +%ProgList(SubRtnC);  
11 @@enter zTest:RoutineA:RoutineB:SubRtnC  
12 205      +%ProgList(action = exit);  
13 @@exit zTest:RoutineA:RoutineB:SubRtnC  
14 206      +%ProgList(action = exit);  
15 @@exit zTest:RoutineA:RoutineB  
16 207      +%ProgList(SubRtnC);  
17 @@enter zTest:RoutineA:SubRtnC  
18 208      +%ProgList(action = exit);  
19 @@exit zTest:RoutineA:SubRtnC  
20 209      +%ProgList(action = exit);  
21 @@exit zTest:RoutineA  
22 210      +%ProgList(zTestNext,action=init);  
23 @@end zTest  
24 @@begin zTestNext
```

```
TestSuite: PutMvars.log  
1 187      +*** TestSuite of Filename<putmvars>;  
2 188      +*** SysDate<13JUL04> SysTime<09:06> SAS version<9.1>;  
3 189      +%Let TestSite = 0; %*off;  
4 190      +*before;  
5 191      +%PutMvars(list = SysDate SysTime);  
6 192      +*after: no note written to log;  
7 193      +%Let TestSite = 1; %*on;  
8 194      +%PutMvars(list = SysDate SysTime TestSite);  
9 @@parms: SysDate <13JUL04>  
10 @@parms: SysTime <09:06>  
11 @@parms: TestSite <1>  
12 195      +%Let TestSite = 0;  
13 196      +%Let TestParms = 1;  
14 197      +%Let aLongMacroVarName = x;  
15 198      +%PutMvars(list= TestParms aLongMacroVarName  
16 199      +      ,msg = ThisRoutineName parameters  
17 200      +      ,testing = &TestParms.);  
18 @@ThisRoutineName parameters TestParms . . . . <1>  
19 @@ThisRoutineName parameters aLongMacroVarName <x>
```