PharmaSUG 2013 - Paper CC34

Using SAS Driver Programs to Automate Workflows and Respond to the Unexpected

Brit Minor, REGISTRAT-MAPI, Lexington, KY

ABSTRACT

A driver is a program that runs another program. Typical uses for SAS drivers are to:

- Run several programs that must be run in a certain sequence.
- Parse program logs looking for potential problems.

Drivers can perform other routine tasks as well, such as:

- Send success/failure emails.
- Log each run of a program to an activity log.
- Upload program results to a study portal.

Thus, driver programs can automate workflows, improve response to failures, and enforce best practices. This paper will describe SAS macros, techniques, and batch command files for writing driver programs.

INTRODUCTION

We use SAS programs to manipulate, display, and analyze clinical data. For a clinical study, we may write many SAS programs. A common sequence of tasks, or workflow, is to run several programs, study their logs to make sure that the programs ran correctly, bring unusual data values to the attention of Data Management, and decide whether to proceed to the next workflow. We are accustomed to doing each of these steps manually. Once a program is debugged and in production, though, it makes sense to automate the workflow as much as possible, and driver programs help us do so.

Code examples in this paper were tested in SAS v9.2 running on Windows.

WHAT IS A SAS DRIVER PROGRAM?

For purposes of this paper, a SAS driver program is a wrapper around the SAS executable, usually "C:\Program Files\SAS\SASFoundation\9.2\sas.exe" under Windows. The driver program can be written entirely in SAS, most simply as

```
/* Driver command 1a.sas */
%SYSEXEC "C:\Program Files\SAS\SASFoundation\9.2\sas" -SYSIN "myProgram.sas" -ICON;
```

Proper use of quotes and positioning of options is important and is sometimes counterintuitive:

```
/* Driver command 1b.sas */
%let sascmd=C:\Program Files\SAS\SASFoundation\9.2\sas;
data _null_; call system("'&sascmd' -sysin myGoodProgram.sas -icon"); run;
```

This program runs as desired despite the fact that it has a macro variable in single quotes. The program will write its log, myGoodProgram.log, to the current directory, but if you move "-icon" ahead of "-sysin", you'll get no log.

Depending on what we want the driver program to do, we might instead write it as a series of SAS and utility programs run in a Windows batch command file:

```
: Driver command 1c.bat
SET sascmd="C:\Program Files\SAS\SASFoundation\9.2\sas" -icon
SET saspgm="myProgram.sas"
%sascmd% -sysparm NOWINDOWS -sysin %saspgm%
```

DRIVER ENVIRONMENTS

The same driver program may be run from several different environments:

- within interactive SAS;
- from within a programming editor, such as UltraEdit;
- from SAS running in batch;
- from a command script;
- via a scheduled task.

Each of these run environments may have different defaults and may return values in a format specific to that environment. For example,

- Default locations of .log and .lst files may depend on the environment.
- The &SYSPROCESSNAME value and format may depend on the environment
  - Program run from within UltraEdit: &SYSPROCESSNAME = Program 'myProgram.sas'.
  - Same program run in batch SAS: &SYSPROCESSNAME = Program "D:\Demos\SAS Drivers\myProgram.sas".
- Program behavior may depend on the environment. For example, in the Windows batch command file shown above, note that the –icon option does come before –sysin, yet we still get the log file that we need.

A BASIC DRIVER MACRO

Probably our most frequent use of SAS drivers at REGISTRAT-MAPI is to automate the checking of SAS logs for our production programs. A SAS program can't check its own log, but a driver that is running the SAS program can. A bare-bones driver macro to set up this process -- run a program, append its logfile name to a list of logfiles, and return the program's error code -- might look like this:

```
%macro runone(pgm=, logs=, retcode=);
    %* pgm is the name of the SAS program to run, without the .sas extension.;
    %* logs holds a pipe-delimited list of program logfile names.;
    %* retcode is a macro variable to hold the return code.;

    %* SAS executable and standard options;
    %let sascmd=C:\Program Files\SAS\SASFoundation\9.2\sas;

    %put Driver macro: SYSPROCESSNAME=&SYSPROCESSNAME;

    %* Run the program;
    data _null_; call system("'&sascmd' -sysin '&pgm..sas' -icon"); run;

    ** Append logfile name to list and add return code to sum.;
    %if %SYMEXIST(&logs)=0 %then %do;
          %global &logs &retcode; %let &logs=&pgm..log;
    %end;
    %else %do;
          %let &logs=&&&logs|&pgm..log;
    %end;
    %put Return codes: sysrc=&sysrc  syscc=&syscc;
    %if %SYMEXIST(&retcode)=0 %then %global &retcode; ;
    %let &retcode = &sysrc;
%mend runone;
```

This driver macro can be called several times in one driver program, for example, to build a set of analysis datasets. We will run three toy programs in our examples:

myNullProgram.sas:

```
%put *** This is my null program ***;
%put Driven program: SYSPROCESSNAME=&SYSPROCESSNAME;
```

myGoodProgram.sas:

```
data _null_;
    put "I am a good program.";
    length sortof 8;
run;
```

Note that myGoodProgram has an uninitialized variable.

myBadProgram.sas:

```
data _null_;
    num = 5; ch = '6';
    sum = num + ch;

    error "I am a bad program."; abort;
run;
```

Note that before throwing an error and aborting, myBadProgram adds a numeric and a character variable together.

Our first driver program will run these three small programs Their three log file names will be concatenated into a pipe-delimited string in macro variable &mylogs, and return codes will be added together to get an overall success/failure code:

```
%* Driver program 1.sas;
%inc "runone.sas";

%runone(pgm=myNullProgram, logs=mylogs, retcode=rc);
%runone(pgm=myGoodProgram, logs= mylogs, retcode=rc1);
%runone(pgm=myBadProgram, logs= mylogs, retcode=rc2);

data _null_;
    put "mylogs=&mylogs";
    put "rc=&rc  rc1=&rc1  rc2=&rc2";
    put "success=%eval(&rc + &rc1 + &rc2)";
run;
```

The three lines written at the end of this first driver's log show the concatenated logfile names and the return code indicating failure from the third program:

```
mylogs=myNullProgram.log|myGoodProgram.log|myBadProgram.log
rc=0  rc1=0  rc2=3
success=3
```

Output 1. Macro variable values written to Driver program 1,log.

PARSING LOGS

Now that we have that list of log files, we can add to our driver program a macro that will parse those SAS logs and collect ERROR and WARNING messages, noteworthy NOTEs, and PROBLEM statements into one nicely formatted file for QC. A bare-bones version could look like this:

```
%macro checklogs(logs=, out=checklog.txt);
    %* checks a list of SAS log files for ERRORS, WARNINGS, and NOTES of interest;
    %* if any such lines are found, they are written out to &out and macro variable
&CLEAN is set to 0.;
    %global clean; %let clean=1;
    %do i=1 %to %numwords(&logs, dlm=|);
        %let log=%scan(&logs, &i, |); %put log=&log;
        data _null_;
            infile "&log" end=eof;
            %if &i=1 %then file "&out"; %else file "&out" mod; ;
              if _n_=1 then do;
                stars = repeat('*', 79); put stars; put "Parsing &log....";
                retain clean 1;
            end;

            input;

            length frstword $200 word $20;
            frstword = upcase(scan(_INFILE_, 1, ':'));
            if frstword in ('ERROR','WARNING') then do;
                put _INFILE_; clean = 0;
            end;
            else if frstword='NOTE' then do;
                checkwords = "uninit|repeats|invalid|numeric";
              found = 0;
              do i=1 to 4;
                    word = scan(checkwords, i, '|');
                    if find(_INFILE_, word, "it") then found = 1;
                end;
                if found then do; put _INFILE_; clean = 0; end;
            end;

            if eof then do;
                call symput("clean", put(clean, 1.));
                if clean then
                    put "Log is clean!";
            end;
        run;
        %put clean=&clean;
    %end;
%mend checklogs;
```

We now enhance our driver program by adding a call to check the logs:

```
%* Driver program 2.sas;
%inc "runone.sas"; %inc "checklogs.sas";

%runone(pgm=myNullProgram, logs=mylogs, retcode=rc);
%runone(pgm=myGoodProgram, logs=mylogs, retcode=rc1);
%runone(pgm=myBadProgram, logs=mylogs, retcode=rc2);
data _null_;
    put "mylogs=&mylogs";
    put "success=%eval(&rc + &rc1 + &rc2)";
run;
%checklogs(logs=mylogs);
```

The output file from parsing the three logs from the previous example looks like this:

```
********************************************************************************
Parsing myNullProgram.log...
Log is clean!
********************************************************************************
Parsing myGoodProgram.log...
NOTE: Variable sortof is uninitialized.
********************************************************************************
Parsing myBadProgram.log...
NOTE: Character values have been converted to numeric values at the places given by: (Line):(Column).
ERROR: Execution terminated by an ABORT statement at line 5 column 34.
ERROR: Errors printed on page 1.
```

Output 2. Results from %checklog macro, written to Driver program 2.log.

RESPONDING TO ERRORS

With the returned SAS return code, a driver can determine whether to continue running SAS programs or to abort the list. The following driver runs myBadProgram, which as we saw previously returns error code 3. The driver doesn't bother to run the next two programs and instead executes a somewhat anemic response:

```
%macro runmany;
    %* Driver macro illustrating response to error;
    %inc "runone.sas";
    %runone(pgm=myBadProgram, logs=logs, retcode=rc);
    %if &rc=0 %then %do;
           %* Run the next two programs;
           %runone(pgm=myNullProgram, logs=logs, retcode=rc);
           %runone(pgm=myGoodProgram, logs=logs, retcode=rc);
    %end;
    %else %do;
           %* Respond to program error;
           %sysexec echo ^G;
    %end;
%mend runmany;
%runmany;
```

SENDING E-MAIL

A much stronger response to unexpected program behavior is to send an email to appropriate recipients. Several daily and weekly scheduled tasks at REGISTRAT-MAPI run driver programs that in turn run production SAS programs. If one of these production programs fails for some reason, the driver program notifies the developer(s), affected user(s), and the program manager about the failure by sending an email. As a best practice, the driver also sends a success email to a much smaller list of interested parties. If neither a success nor a failure email is received, that signals some problem with the driver or with the system itself.

Here is code for a simple macro that will send an email:

```
%macro sendmail(sender=, TO=, CC=, BCC=, subject=, attach=, body=);
    %* Email report automatically;
    %* &TO, &CC, and &BCC must quote each email recipient.;
    %* Separate each list item with whitespace.;

    filename sendmail EMAIL emailid=&sender
           TO = ( &TO )  CC = ( &CC )  BCC = ( &BCC )
           SUBJECT = &subject
           %if &attach ne %then  ATTACH = &attach ;
    ;

    data _null_; file sendmail; put &body; run;
%mend sendmail;
```

We can add this email capability to our ever-expanding driver program:

```
%* Driver program 4.sas;
%inc "runone.sas";

%runone(pgm=myNullProgram, logs=mylogs, retcode=rc);
%runone(pgm=myGoodProgram, logs= mylogs, retcode=rc1);
%runone(pgm=myBadProgram, logs= mylogs, retcode=rc2);
%let success=%eval(&rc + &rc1 + &rc2);
data _null_;
    put "mylogs=&mylogs";
    put "rc=&rc  rc1=&rc1  rc2=&rc2";
    put "success=&success";
run;

%inc "checklogs.sas";
%checklogs(logs=&mylogs, out=example4_loglines.txt);

%inc "sendmail.sas";
%macro notify;
    %* A SAS error code of 1 often can be ignored;
    %if &success<2 %then %do;
        %let subj=Example4 ran; %let body=All done!;
        %let TO="project_manager@registratmapi.com";
        %let CC="user@registratmapi.com";
    %end;
    %else %do;
        %let subj=Example 4 failed; %let body=Sorry man,try again.;
        %let TO="bminor@registratmapi.com";
        %let CC="project_manager@registratmapi.com";
    %end;
    %sendmail(sender="dummy@registratmapi.com", TO=&TO, CC=&CC,
            subject="&subj", body="&body");
%mend notify;
%notify;
```

Running this version of the driver resulted in the following mail being sent:

Sent: Sunday, March 31, 2013 11:32 PM
To: Brit Minor
Cc: project_manager@registratmapi.com
Subject: Example 4 failed

Sorry man,try again.

Output 3. Failure email sent by Driver program.sas.

WRITING PROGRAM ACTIVITY TO AN AUDIT LOG

A SAS driver program can perform any number of house-keeping tasks in addition to the ones that have been illustrated so far. One last example might be to write to an audit log for production programs to record the date/time and return code that a program ran.

```
%macro activitylog(logpath=., logname=ActivityLog, progname=, rc=&syscc);
%* Writes a summary record to a program activity log.;
    libname __act "&logpath";

    %if %sysfunc(exist(__act.&logname))=0 %then %do;
    data __act.&logname;
        length drivername progname user $100;
        drivername = ' '; progname = ' '; user = ' ';
```

```
        rundate = .; runtime = .; retcode = .; logissues = .;

        format rundate date9. runtime time5.;
        stop;
    run;
    %end;

    data logrec;
        length drivername progname user $100;
        drivername = "&sysprocessname";
        if "&progname"=' ' then progname = "%m_getprogname";
        else progname = "&progname";
        user = "&sysuserid";
        rundate = date(); runtime = time();
        retcode = &rc; logissues=0;
    run;

    proc append base=__act.&logname data=logrec; run;
%mend m_activlog;
```

A FULLY ENDOWED SAS DRIVER

We will end with an example of a driver program that is adapted from one that's in use at REGISTRAT-MAPI. The production driver program is set up as a Windows scheduled task to run a daily report. The driver includes code to determine whether it is being run in a development or production environment and directs success or failure emails appropriately.

```
%* Driver program 5.sas to run Daily Reports.;

%* This driver program will run one or more SAS programs, parse each SAS log for
errors or other issues, and send an email announcing success or failure.;

%*----- location of sas 9.2 executable file for &m_runprog macro --;
%let sasloc=C:\Program Files\SAS\SASFoundation\9.2;

%******************;
%let prog=DailyReports;   %* SAS program filename without .sas extension;
%let progpath=.;          %* program path is current directory by default.;

%* The following two lines run the program and then parse its log for errors, etc.;
%inc "runone.sas"; %inc "checklogs.sas";
%runone(pgm=&prog, logs=reportlogs, retcode=progerror);
%checklogs(logs=&reportlogs, out=dailyreport_loglines.txt);

%* The runone macro stores final error status of the program in &progerror.;
%*  0=success, 1=success with warnings, >1=failure;
%* The checklog macro stores the "cleanliness" status of the program log in the
global &clean. 0=clean log, >0=at least one "bad word" found in log: error,
warning, uninit, problem, repeats, etc.;
%put &prog has run: progerror=&progerror  log check=&clean;

%* write a record to the study activity log.;
%inc "activitylog.sas";
%activitylog(logpath=&progpath, logname=myStudyLog,
    progname=&prog, rc=&progerror);
```

```
%******************;
%* If more than one program needs to be run as part of this batch, then store the
current values of &progerror and &clean and then set up, run, and parse another
program.;

    %* EXAMPLE;
    %*let curerrors=&progerror; %*let curclean=&clean;
    %*let prog=<program name 2>; %*let progpath=.;

    %*runone(pname=&prog, logs=reportlogs, retcode=progerror);
    %*checklogs(logs= reportlogs, out=progname2_loglines.txt);
    %*put &prog has run: progerror=&progerror  log check=&clean;

    %*activitylog(logpath=&progpath, logname=myStudyLog,
    progname=&prog, rc=&progerror);

    %* Assuming that all programs and logs must pass for success, add all the error
and clean values together;
    %*let progerror=%eval(&curerrors+&progerror);
    %*let clean=%eval(&curclean+&clean);

    %* repeat as needed.;

%******************;
%* When all programs in the batch have run, compose and send a success or failure
email.;

%let studyname=Study; %let progname=Daily Report;
%let prodoutput=\study\DailyReport.xls;
%let developer1=bminor@registratmapi.com;
%let dev1name=Brit Minor; %let dev1phone=+1 800 381 7878 (Extension 4391);

%* run macro that checks whether this is the development environment;
%DEVorPROD;

%macro notify;
    %let today=%sysfunc(today(),date9.);
    %if &progerror>0 %then %do;
        /* send failure email */
           %if &dev %then %do;
               %let emailTO="&developer1" "&developer2";
               %let emailCC=;
           %end;
           %else %do;
               %let emailTO="&developer1" "&developer2";
               %let emailCC="&user1" "&user2" "&pm" "&group";
           %end;
           %m_sendmail(sender="&SYSUSERID@registratmapi.com",
               TO=&emailTO, CC=&emailCC,
               subject="&studyname &progname *** FAILED ***",
               attach=, body=%str("Hi developers," //
               "There was an error running &studyname &progname today, &today.
                The program is &curdir\&prog..sas.  Questions welcome." //
            "&dev1name" /
            "Tel.: &dev1phone"
               )
           );
    %end;
    %else %do;
        /* send success email */
           %if &dev %then %do;
```

```
                %let emailTO="&developer1"; %let emailCC=;
            %end;
            %else %do;
                %let emailTO="&user1" "&pm"; %let emailCC="&group";
            %end;
            %m_sendmail(sender="&SYSUSERID@registratmapi.com",
                TO=&emailTO, CC=&emailCC,
                subject="&studyname &progname has run",
                attach=, body=%str("Hello all," //
                "The &studyname &progname for &today has run. Output is available for
review at %str(&prodoutput)." //
              "&dev1name" /
              "Tel.: &dev1phone"
                )
        );
    %end;
%mend notify;
%notify;
```

CONCLUSION

Much of a programmer's time is spent running programs, checking whether they ran correctly, and responding appropriately when they don't, over and over again. SAS driver macros and programs can help automate these repetitive workflows and thereby reduce the likelihood of programmer error. Timely notifications of problem conditions can help reduce interruptions in other workflows as well.

ACKNOWLEDGMENTS

The author would like to thank Tim Moore and Ron Vanderhouten of Quality Research Partner, Inc, Oshkosh, WI, for sharing their sophisticated SAS driver and log checker macros. The author also thanks the Programming group at REGISTRAT-MAPI for constructive suggestions.

CONTACT INFORMATION

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

Name: Brit Minor
Enterprise: REGISTRAT-MAPI
Address: 2524 Commonwealth Dr.
City, State ZIP: Charlottesville, VA 22901
Work Phone: +1 859 223 4334 x4391
Fax: +1 434 973 1019
E-mail: bminor@registratmapi.com