Combining Display Manager and Batch Mode Under UNIX

John Blodgett, University of Missouri St. Louis

Abstract

The two classic methods of developing and running SAS® programs are batch mode and interactive mode using the SAS Display Manager System. Each of these methods has its advantages and disadvantages and most SAS programmers have strong opinions about which is better. We describe a method that we use for running SAS in a UNIX environment that allows us to take advantage of both of these methods simultaneously. By the careful application of a few coding standards and with the help of some custom key definitions in display manager, we are able to utilize the features of the interactive environment for editing our source code and reviewing the results, while we virtually replace the standard SAS "submit" command with our own "batch submit" procedure. We find that using this technique has easily more than doubled our SAS productivity.

Overview of the Technique

The technique involves using the SAS Display Manager System to edit your programs, and then submitting them to run as independent UNIX background processes instead of using the display manager "submit" command. Results of these batch runs -- SAS log and output files -- are then reviewed using FSLIST (or BROWSE) windows in the foreground session (so the method requires having the SAS/FSP® installed.) The basic steps are:

1. Invoke SAS interactively with the DMS option.
2. Edit your program using the SAS text editor in the Program window.
3. Write your program to an external file in the current directory using the *file* command..
4. Use the *"x"* display manager command ("shell to system") to invoke a background process running SAS in batch mode with the program just saved as the SAS source file.
5. Review the results of the batch run -- the .log and .lst files -- using FSLIST/BROWSE windows.

(Repeat steps 2-5 as necessary.)

Some Simple Coding Conventions

The key to making this method fast, reliable and powerful is the adoption of some simple coding conventions and the strategic use of custom key settings in display manager. Here is an example of a SAS program that will be run in this "interactive batch" mode (parenthesized line numbers added for reference):

```
(1)  x cd /myjobs/proj101;   *<--point to directory for job--;
(2)  %let pgm=p101job1; *<--store program "name" in macro variable--;

(3)  filename pgm "&pgm..sas";  *<--Always use pgm as fileref. Coding
     convenience;
(4)  filename in '/mydata/proj101.indata';   *<--example of a filename stmt,
     optional--;
(5)  libname  sasout '/myjobs/sasdata';      *<--example of a libname stmt,
     optional--;
(6)  title 'Project 101, Job 1: Convert Raw Data to SAS';
(7)   ... <rest of program goes here, to run in batch only>  ....
```

The first 3 lines are the most critical (actually, without the comments these 3 "lines" usually fit on a single line.) With these statements we accomplish the following:

(1) Establish the current directory. This is important to simplify saving our program and telling SAS where to store the program and, when submitted for background processing, where to write the SAS log and output listing (.log and .lst) files.

(2) Defining the *pgm* fileref with the filename statement makes it easy for us to save our code, using the display manager command *file pgm <r>*. Note that we need only specify the file name - the path is taken care of by the cd command because of the special way that SAS processes the *x cd* statement under UNIX .

(3) The use of the *%let* statement to establish a global parameter called *pgm* with a value equal to our program name simplifies how we initiate and review the results of the background process(es) we invoke to run the program. This involves customizing some keys to make things simpler and faster.

Custom Key Settings

A critical element in making this method quick, easy and reliable is the ability to take shortcuts for the routine tasks that you need to repeat over and over as you edit, submit and review your jobs. The method could be employed without using these key shortcuts, but we're not sure we'd want to use it then. You set up these special key definitions usually by using the KEYS window, typing and saving the values (in your *sasuser.profile* catalog.) We use four keys on the numeric keypad of our extended keyboard but you can use any four keys that you prefer. Our custom key definitions are:

KEY	DEFINITION	
kp-enter:	x sas &pgm &	("Batch submit" key.)
kp-1:	FSLIST &pgm..log	("Browse log" key.)
kp-2:	FSLIST &pgm..lst	("Browse output" key.)
kp-3:	next FSLIST	("Swap log/list" key.)

The *kp-enter* key is used to submit the program. After entering the first few lines of my program (as shown in the sample above) I submit those lines for interactive processing. This establishes the pgm fileref and the value of the &pgm global parameter. Optionally (but highly recommended) it establishes the same data references (filerefs and librefs) for the interactive process as will be used for the batch process. After submitting these lines from the Program window of my foreground session, I immediately *recall* them so they are also part of the batch program I am about to submit.

The command *x sas &pgm &* starts a subshell and runs the UNIX command "*sas &pgm &*" - except that the *&pgm* reference will be replaced with the name of the program. In our example this becomes "*sas p101job1 &*". The trailing "&" tells the shell to run the program in the background. The process begins and control is immediately passed back to the display manager session. In fact when you hit the key that submits the job you will typically not see anything happen if you are running in X-windows mode, and will simply be prompted to hit enter to continue with your foreground session otherwise.

Reviewing Results of the Batch Job

There is no built-in "notify" feature in SAS or UNIX to tell you when your background job is completed but you can create your own. We use a utility module that runs a short SAS data step which invokes the *sound* function several times to provide an audible signal when the job has completed. We

have this short piece of code stored in a subdirectory which we name *sascode* and the program name is *notify.sas*. So, the last line of each program that we run using this technique is

%include sascode(notify);.

The source code for this program is included in Appendix A.

Of course, you can use a UNIX "ps" command to check on your job. Most jobs finish in just a few seconds. When the job first starts up SAS creates (or recreates, erasing the former contents, for reruns of the same job) the .log file that it will be writing. By pressing the *browse log* key (kp-1) you will be invoking an FSLIST window and automatically loading the current contents of the SAS log file associated with the background job. Wait for the audible alarm indicating the job is finished before hitting this key.

The "next FSLIST" key (kp-3) is useful for swapping back and forth between the log and output displays. It is important to understand that you should *not* use the FSLIST command to return to a window that you already have open. If you do, SAS will open a new window and will display the file in it, just as you'd expect. But now you will have multiple windows open displaying the same file and this can lead to lots of confusion when you have submitted the job 4 times and have 4 different windows open showing the results of 4 different runs. When you are finished browsing a file with FSLIST you should always close the window with the "end" command. Be sure to do this with each FSLIST window you open using this technique, especially when resubmitting the job. As an alternative you can implicitly close it by issue the *BROWSE* command from a FSLIST window, which closes the current window and invokes a new FSLIST window. To help enforce this convention, we have defined the kp-1 and kp-2 keys for our FSLIST window to issue the commands browse &pgm..log and browse &pgm..lst, respectively - i.e. we have replaced the FSLIST command with BROWSE when we are issuing the command from inside an FSLIST window.

Advantages of Running in Batch Mode

We will not go into all the advantages and disadvantages of batch vs. interactive mode here, but one key factor for applications that involve large log or output files is the speed of execution. Writing these files to display manager windows requires considerably longer (it checks for things and uses colors and bolding to make these windows easier to read). A program that produces several hundred lines of output in either or both of those windows will run significantly faster in background. The FSLIST window does not have all the nice color-coded features of the display manager LOG or OUTPUT windows, but we find that for most of our applications it is just fine. You cannot use the *save* or *print* commands from FSLIST but if you need to do this it is easy to use the NOTEPAD window to bring in the .log or .lst file. Of course, you can easily send these files directly to a printer using UNIX printer commands. And the *save* command is only relevant if you decide to create an extra copy since with batch mode it is already saved on a file -- the one you are browsing.

Another major advantage of batch mode is that it avoids some of the pitfalls of the interactive display manager method, which can be particularly troubling for display manager beginners. These include forgetting to save (all) your program and having a step "hang" and losing all your work because you have to "kill" the session. Another is forgetting to include a "run;" statement at the end of your submitted code and having display manager display the message "Data (or PROC xxxxx) Step Running" when in fact it is not (it is waiting for something to tell it the step is complete before actually running it.) And, of course, there is the classic glitch of trying to recover from having left off a closing quote or forgetting a *%mend* statement. The former is particularly common, and display manager veterans learn how to cope with it (you submit a line such as

'; run cancel; proc options option=ls; run;

hoping that the extra leading tic will balance off the quotes and let you then recall, edit and try again; the "proc options" is a test to see if SAS is processing statements again.)

This method is a good way to wean middle-aged "I've been doing it this way for 20 years and I just prefer it" programmers away from less productive or platform-dependent methods. Sometimes during the lulls when they are waiting for their batch jobs to complete they may get desperate and try exploring the on-line help or the sample libraries.

Advantages of Using the Display Manager Environment

While we have focused on the fundamentals of how to access the SAS log and output files, the real strength of using display manager as your batch control environment is that it lets you easily examine your *data* as well as your code and printed output. The FSBROWSE and FSVIEW windows can be used to interactively explore either input or output SAS data. The LIB-DIR-VAR window group can be an invaluable tool allowing you to quickly verify names of SAS data sets and/or variables. You can also use the full-screen SAS data browsing tools (FSBROWSE and FSVIEW--the former is invoked when you type a "b" command in a DIR window) to examine observations. This is why we encourage putting all your filename and libname statements at the top of the program (it's good coding form anyway) and submitting those in the foreground as well as part of your batch jobs. Doing this makes it very easy to view your data sets and data files using the powerful display manager tools.

Using a USER Library to View Intermediate Results

If you have a job with many steps that creates a lot of intermediate SAS data sets, it is a common part of debugging to be able to examine those intermediate sets to make sure that things are what you expect. This typically requires using partial PROC PRINT or sometimes PROC FREQ steps to view partial contents of these data sets. But with this method an alternative to this strategy is to define the special *user* libref. When this is done all single-level SAS set references assume a first level of user instead of work. So these data sets are retained in the directory associated with the user libref. If you submit the *libname user <directory>*; statement in your foreground session and it is also part of your batch jobs then when the batch job completes you can examine the contents of all the intermediate sets from your foreground session using FSBROWSE and/or FSVIEW. This not only saves time, but saves having to write a lot of temporary diagnostic code in the programs. Of course, you need to be sure to delete all these temporary files once the program is debugged. (Another advantage of this technique is that if your program runs, say, 3 of 5 steps successfully, and the output of step 3 is a "temporary" data set used by step 4, then you can "comment out" the first 3 steps (by bracketing the code for them with "/*" and "*/") and pick up with step 4 which will be able to read the "temporary" data set that is still stored on the user library. This can be a major time saver.)

Interactive Session and Batch Session

This method allows you to work on an interactive session interleaved with your background programs. Here you code your batch program and submit it for background processing. While that is running you clear the PROGRAM window and work on another program (possibly some new steps that will eventually be incorporated into the batch program.) Any time you want to check on the progress of the batch job just hit *browse log* key. If you see you need to make changes and resubmit your batch program you can use a NOTEPAD window. A good idea (especially if you use NOTEPAD windows like I do for editing other files) is to specify a module name when you invoke the NOTEPAD window. This name is added to the title bar of the window and when you issue a save command in the window it automatically saves the text (SAS code) as a source module with that name in your profile catalog. Just use the command *note &pgm* to open the window and establish the name of the source module you'll be editing. If an "old" copy of the program is in the window (automatically copied for you by SAS which looks for a source module of that name in your sasuser.profile catalog) use *clear* to erase it from the window. Then type *"inc pgm*" to load

the current version (the one you just submitted), do your editing, type *file pgm r*, hit your submit-to-background key, and return to the program window and your interactive task. Of course, on subsequent submissions of the program the source will already be in that NOTEPAD window - no need to close it between submits.

Glitches and Gotchas

While we are generally very enthusiastic about this method, it is not without its limitations and pitfalls. The limitations we can generally live with, the pitfalls need to be understood and carefully avoided. We have been using the system for about two years now, and the pitfalls are rarely a problem any more. But they were a significant nuisance when we first started, and our intent here is to forewarn new users of what to watch out for.

Most of the limitations of the method are the limitations of batch processing in general – no interactivity. If your application requires interactive user input, this will obviously just not work (though you could probably use it to debug portions of the task following the user input.) We have also noted that the method will hang if you try to invoke an AF application with the DISPLAY command - even if the application being displayed does not have a window and does not require interactive input from the user. Also, since your foreground SAS session is making active use of your sasuser.profile catalog, your batch session will not be able to access it – and you'll see notes on the log saying that the batch session was not able to open that catalog. So you'll either need to devise a special scheme to use two different profile catalogs - one for interactive and one for batch - or simply avoid using *sasuser* as a place to store data and source code that may need to be accessed by your batch jobs.

The pitfalls are generally simple and easy to avoid. You just have to be careful. They include:

- It's easy to assume that since you have entered the commands to specify the current directory and the definition of the *pgm* fileref that SAS will somehow *know* this. Not unless you actually submit those commands in your interactive session. If you forget to do this and then type *file pgm* it "works", writing your current program to a file named pgm.sas in the current (usually your home) directory. Trying to view the results using the "fslist &pgm..log" command is not going to work.
- The command to write your program to the file where your batch job can access it is *file* pgm, not *save* pgm. The latter *does* write your code somewhere (to your sasuser.profile catalog) but not where you want it.
- You need to issue the *file pgm r* command before each batch submission. Otherwise, you are just going to get a wasted rerun or the old code.
- You need to not only *submit* the commands to define the pgm fileref , but you need to *verify* that those commands contained no syntax or other errors. This is especially important if you are switching to a new program and intend to change the definition of the pgm fileref. Otherwise when you type the command *file pgm r* you could be writing over your previous program. Our favorite way to make sure we are going to save (file) our code where it belongs is to type a *filen* command and verify the entry in the FILENAME window.
- Once you have submitted a batch run do *not* modify the pgm file before that batch run completes. It's perfectly all right to go on editing the code in the window, perhaps coding step 2 while doing a test run of step 1, but if you issue a "*file pgm r*" command while the background job is still running, remember that the background process is reading that file as its SAS source file. We used to get some very strange SAS logs that made us think there was something fundamentally wrong with SAS. What was happening, of course, was that SAS was "losing its place" in the input file which was being altered while it was being read (most operating systems would not let you do this, but UNIX -- at least AIX -- will.) Note that this also means that if you have a program that you want to run a number of times, just changing a few parameters for each run, you cannot do it using this method. Since all the batch runs read the same SAS source file, they cannot be running simultaneously. There are strategies for getting around this, of course, using UNIX scripts.
- Forgetting to close FSLIST windows in which you are viewing the SAS log and output files can cause confusion. Its very easy to click on one of these "old" windows and think you have the latest one. You'll be scratching your head wondering what went wrong -- you could swear you fixed that bug -- when in fact the window showing that your program is now running just fine now has either not yet been opened, or is opened but you just stumbled onto another one by mistake. This is why we emphasize the importance of closing these windows between submissions.

Other Platforms

We have really only used this method on a UNIX system with an X-Windows interface. We have tried running something comparable under Windows95 but with only limited success. Turns out many of the key features that we rely upon do not work under Windows. For example, that operating system does not like to let different programs (tasks) have the same file open at the same time, something that happens all the time with our method in UNIX. This requires you to carefully close all your interactive browsing windows -- including the FSLIST/BROWSE windows for looking at the log and output files as well as but also the FSBROWSE, VAR, and FSVIEW windows which you may have open to check for information about SAS data sets being used by the application. This also means that if you have a program that is taking quite a while to complete, you cannot go out and browse the log file to see what's going on – you have to wait for the whole job to finish.

A nice thing about running in windows is that you can specify the "-icon" option when invoking the batch job. This causes an icon representing your job to appear in your task bar menu across the bottom of your screen. When that icon goes away, the job is finished.

Summary

We have found the methods we have described for running SAS jobs in batch mode but with the SAS Display Manager interactive environment as the "control center" significantly improve the time required to develop our programs. The system requires a relatively small number of coding conventions and processing cycle steps that need to be carefully followed. While it takes a few days to get familiar and then comfortable with these conventions, doing so will save many man-days of programming development time over the long haul.

Appendix A:

Source code for the sascode(notify) module:

```
options nonotes nosource;  *--put "notify" msg on stderr file--;
options obs=max; *--in case of error obs=0 will be set;
 options no$syntaxcheck;
%let se=&syserr;
data _null_;
 call sound(1,1);   call sound(1,1);  call sound(1,1);
 file log;
  put "**** &pgm has completed &sysday &sysdate &systime
syserr=&se *****";
 file stderr;
  put "**** &pgm has completed &sysday &sysdate &systime
syserr=&se *****";
 run;
```


John Blodgett
Urban Information Center, Room 211 Lucas Hall
8001 Natural Bridge Rd.
St. Louis, MO 63121
(314) 516-6014
e-mail: c1921@umslvma.umsl.edu