Paper 278-2010

Windows and Unix Computers Now Have Multiple CPU's; Why Not Control Two or Three or More Parallel Executing SAS® Batch Jobs from One Master Job!

William E Benjamin Jr, Owl Computer Consultancy, LLC, Phoenix AZ

ABSTRACT

This paper will show the processes required to start multiple, parallel batch copies of SAS on the SAME Microsoft Windows or UNIX based computer. It may be easy to click the SAS Short-cut button or type SAS in a command prompt window a few times, but those jobs are independent of each other, and use resources to update the screen images for the operator (you). The process proposed here allows one SAS job to start others as child batch processes, with no screen update requirements, with their own parameters and monitor their completion. This paper exposes some of the tricks required to get two (or more) batch copies of SAS running on a Windows or Unix computer.

INTRODUCTION

If your first impression of this topic is "All I have to do is type SAS (on UNIX) two or three times, or click on my SAS icon (on Windows) and I have as many sessions going at once as I want." Then this presentation is not for you. It is for the programmer that wants to start and control all of the extra sessions from the first one, then do something automatically when the other jobs finish. But the next argument is that "I can do that now by including any code I want. Then, when each job finishes I can include the next SAS code file." And this is still true; but, this paper is not for you either. This paper is about getting your jobs to run in parallel, and detecting when they complete to continue forward with the next parallel set of jobs. Suppose three monthly jobs need to be executed, but need to wait until after weekly jobs for the end of month close-out of the accounting books (for some reason), and each job takes 8 hours to run. Simple math will tell you this job will run for two days. This paper is for the programmer that wants to make the job run in 8 to 16 hours.

THE PROBLEM

Any system, whether it is a river like the Mississippi River, or your office e-mail or internet connection can only produce results as fast as the slowest point will allow. If too much water comes down the river and cannot pass one point fast enough, it will back-up upstream until it over flows the river bank and floods somewhere. If too many people listen to internet radio, you cannot get your e-mails in or out. Well, it seems like any programming department will come up against a big job. Like most things in business, one big job rarely only happens one time. So, "Plan-For-It"!!!

If a job can be broken into parts that are independent and can run at the same time, then do it. This means find the parts that are the same, and the parts that need input from other steps. If four weekly jobs can run without input from any of the other job parts, then they can run in parallel. If the monthly job needs to sum or merge data from the four weekly jobs it has to wait for them to finish before it can run. It must wait. This will paper show you how to get independent parts to run together.

THE SOLUTION

Create a job that knows about and controls other jobs. First break your task into common units. Find something that can run independent of the other tasks. But do not make the tasks it too small, if the started tasks run for two to four hours that works well. Windows can handle about three SAS jobs at a time (including the control job), a dual CPU machine may handle more. Unix can handle more SAS jobs at one time, but you may be limited to the number of jobs that can be running at the same time. The number of CPS's in the UNIX complex will definitely play a roll in the number and speed of the jobs. Write and debug the job steps independently. You need to know that a job step will finish.

Take notice of whether or not all of the jobs use the same input files. If this is the case then the Master Control Program (MCP) will need to wait some time before starting the next task. The SAS function "SLEEP" works well to delay jobs. The reason the delay may be required is that file I/O contention can severely slow two or more jobs trying to read the same file. Jobs that are staggered may be reading the same files, but not in the same relative location. This could help speed up the I/O throughput.

Since the SAS jobs will be running independently of each other, the only way to communicate with each other (i.e. the started task wants to tell the MCP that it has finished.) will be by writing to a file for the MCP to read. The MCP should not check the files too often, checking every 20 seconds will write too much information to the SAS log file, this will cause the MCP to hang waiting for you to tell it if you want to delete the log, write it to a file or some other option. Time the file checks to be about four or five checks during the time you expect the jobs to run. This is not too often, and not so long between checks that it is a long wait after the jobs finish.

HIGH LEVEL VIEW OF HOW IT WORKS

This process will work in both UNIX and Windows, and with minor differences works the same way on both operating systems. The process is simple, you start SAS, call the first program into the SAS Editor and examine the way the job is set to run, then start the job. This program will control everything else. The Master Control Program (MCP) will start other programs in a background or batch mode. When windows starts a new job (with the method outlined below) the SAS start-up splash screen is visible for a few seconds, then a new window appears indicating there is a SAS job running in a batch mode. This window will identify the name and locations of the (1) running program, (2) List output file [OUTPUT window display information], and (3) the Log output [LOG window display information]. You can open or close this information window, but the program is on its own and out of your control. (Note – make sure the program works before sending it on its way alone).

The MCP can send a message to the started task via the SAS SYSPARM option of the “SAS start-up command on your system” by using the “X” command. So the MCP can send modifying information to the called program. If the called program writes a data file just before finishing, the MCP can test the contents for a completion code and determine when to continue. This cycle of steps can repeat as many times as needed to do the job. On windows the noxsync and noxwait will control the command window.

For example, the MCP starts four batch processes and tests the four flag files until all four are marked complete. Then starts another three jobs and tests the flags until they all are complete, and then starts five more jobs…. Etc. Eventually the MCP will be ready to quit. If at this point you are asking “How many times do I need to include the code to test these flag files?” The answer can be once, if you use the power of SAS Macros to execute the code repeatedly.

LET’S LOOK AT THE PIECES NEEDED

WHAT MAKES IT GO?

(1) The key that makes this work is the ability of a running SAS program to execute an operating system command in real time. That command can do anything that a DOS or UNIX line command (or a TSO command on a mainframe) can do in real time. If you can sit in front of a computer terminal and type a command into a DOS window, Unix window, or a TSO window (the base windows not a running program) then SAS can do it too.
(2) You need to know what to type.
(3) The called program needs to know how to communicate with the MCP.
(4) The MCP needs to know when to execute the next step (i.e. call the next program) or quit.

THE SIMPLE WAY – TYPE IT ALL OUT TO SEE

The next lines of SAS code will execute the command, but there is no flexibility at run time, the code is fixed and the programmer must make any changes to run it next time with different parameters:

Here is one method:

For Windows – to run two jobs type:

```
   data _null_;
x SAS 'Z:\SGF_2010\code\My_SAS_Program_1.sas' -LOG
'Z:\SGF_2010\logs\My_SAS_Log_1.txt' -PRINT 'Z:\SGF_2010\List\My_SAS_List_1.txt' -WORK
'Z:\SGF_2010\Work\PGM1_work_dir' -SASINITIALFOLDER 'Z:\SGF_2010\Init\PGM1_init_dir' -
SYSPARM 'Any Parm string you want - up to 200 characters long, but stay away from
quotes and semi-colons';
x SAS 'Z:\SGF_2010\code\My_SAS_Program_2.sas' -LOG
'Z:\SGF_2010\logs\My_SAS_Log_2.txt' -PRINT  'Z:\SGF_2010\List\My_SAS_List_2.txt' -
WORK   'Z:\SGF_2010\Work\PGM2_work_dir' -SASINITIALFOLDER
'Z:\SGF_2010\Init\PGM2_init_dir' -SYSPARM 'Any Parm string you want - up to 200
characters long, but stay away from quotes and semi-colons'; run;
```

For Unix – to run two jobs type:

```
   data _null_;
   x /my/path/to/the/unix/bin/SAS '/my/unix/path/My_SAS_Program_1.sas' -LOG
   '/my/unix/path/logs/My_SAS_Log_1.txt' -PRINT '/my/unix/path/List/My_SAS_List_1.txt'
   -WORK   '/my/unix/path/Work/PGM1_work_dir' -SYSPARM 'Any Parm string you want - up
   to 200 characters long, but stay away from quotes and semi-colons' &&;
   x /my/path/to/the/unix/bin/SAS '/my/unix/path/My_SAS_Program_2.sas' -LOG
   '/my/unix/path/logs/My_SAS_Log_2.txt' -PRINT '/my/unix/path/List/My_SAS_List_2.txt'
   -WORK   '/my/unix/path/Work/PGM2_work_dir' -SYSPARM 'Any Parm string you want - up
   to 200 characters long, but stay away from quotes and semi-colons' &&;
   run;
```

All right – that was deliberately unreadable source code, next we will look at something more readable.

MORE COMPLEX – BUT MORE READABLE TOO

Use SAS macro variables to represent the pieces of the execute (x) command:
For Windows:
```
options noxsync noxwait; * Turn off wait commands - run jobs together not in serial
;
options mprint mlogic symbolgen; *to help debug the processing;
Filename Job_1 'Z:\SGF_2010\logs\flag_file_for_job1.txt';
Filename Job_2 'Z:\SGF_2010\logs\flag_file_for_job2.txt';

%let dash    = -;      * a constant for a dash;
%let suffix  =  ;      * not needed in windows so make it a blank;
%let run_sas = SAS;    * set-up the command to run sas on this os;

* set-up the new work files for the new jobs;
%let SAS_Code = Z:\SGF_2010\code\; * SAS code folder;
%let SAS_Logs = Z:\SGF_2010\logs\; * SAS Logs;
%let SAS_List = Z:\SGF_2010\list\; * SAS Output ;
%let SAS_Work = Z:\SGF_2010\work\; * SAS Work files;
%let SAS_Init = Z:\SGF_2010\init\; * SAS Start Folder;

*create unique init folders for each job; *required in windows not unix;
%let init_01 = &dash.SASINITIALFOLDER "&SAS_Init.PGM1_init_dir";
%let init_02 = &dash.SASINITIALFOLDER "&SAS_Init.PGM2_init_dir";
```

For UNIX:
```
options mprint mlogic symbolgen; *to help debug the processing;
Filename Job_1 '/my/path/SGF_2010/flag_file_for_job1.txt';
Filename Job_2 '/my/path/SGF_2010/flag_file_for_job1.txt';

%let dash    = ;                      * a constant for a dash;
%let suffix  = &&;                    * && runs job in background on UNIX; %let run_sas
= /my/path/to/the/unix/bin/SAS; * Point to UNIX executable;

* set-up the new work files for the new jobs;
%let SAS_Code = /my/path/SGF_2010/code/; * SAS code folder;
%let SAS_Logs = /my/path/SGF_2010/logs/; * SAS Logs;
%let SAS_List = /my/path/SGF_2010/list/; * SAS Output ;
%let SAS_Work = /my/path/SGF_2010/work/; * SAS Work files;
%let SAS_Init = /my/path/SGF_2010/init/; * SAS Start Folder;

*create unique init folders for each job; *required in windows not unix;
%let init_01 = ;  %let init_02 = ;
```

Then string them together, like this:
```
*create a parm string;
%let parm = Any Parm string you want - up to 200 characters long, but stay away from
quotes and semi-colons;

* set-up the execute command to run SAS in Batch mode on UNIX or Windows;
* use double quotes to resolve the macro variables;

%let Pgm1_Execute =
x &run_sas      "&SAS_code.My_SAS_Program_1.sas"
  &dash.LOG     "&SAS_Logs.My_SAS_Log_1.txt"
  &dash.PRINT   "&SAS_List.My_SAS_List_1.txt"
  &dash.WORK    "&SAS_Work.PGM1_work_dir"
  &init_01      &dash.SYSPARM "&Parm."
  &suffix;

%let Pgm2_Execute =
x &run_sas      "&SAS_code.My_SAS_Program_2.sas"
  &dash.LOG     "&SAS_Logs.My_SAS_Log_2.txt"
  &dash.PRINT   "&SAS_List.My_SAS_List_2.txt"
  &dash.WORK    "&SAS_Work.PGM2_work_dir"
  &init_02      &dash.SYSPARM "&Parm."
  &suffix;
```

WHY SO MANY PARTS AND WHAT DO THEY MEAN?
Let’s look at the command one part at a time, by the numbers:

```
1) %let Pgm1_Execute =
2) x &run_sas      "&SAS_code.My_SAS_Program_1.sas"
3)   &dash.LOG     "&SAS_Logs.My_SAS_Log_1.txt"
4)   &dash.PRINT   "&SAS_List.My_SAS_List_1.txt"
5)   &dash.WORK    "&SAS_Work.PGM1_work_dir"
6)   &init_01      &dash.SYSPARM "&Parm."
7)   &suffix
8) ;
```

Line by line explanation of the code parameters. Note the double quotes resolve the macro variable values before submitting the ‘X’ command while the %let is compiling. Use %put &Pgm1_Execute to print the value on the SAS log:

Line 1:		- This line starts the definition of a macro variable called Pgm1_Execute.
Line 2:	x	- The SAS command to execute an operating system command.
	&run_sas	- A macro variable defined for either windows or Unix to begin the execution of SAS under the operating system. This can be either the full path to the executable or a command the O/S knows that will execute SAS. Above it was defined as “SAS” for Windows and “/my/path/to/the/unix/bin/SAS” for Unix.
	&SAS_Code	- The directory path to the SAS source code. This is defined this way to allow the same code to execute on both Windows and Unix. By placing the “path names” into a macro variable the fact that Windows uses a “\” and Unix uses a “/” can be over come at run-time by the programmer .
	.	The first period immediately following any macro variable name means that the macro compiler should not place a space between the macro variable value and the next text character found (i.e. no space between the path name and the program name.
	My_SAS_Program_1.sas	– The name of the SAS program to execute.
Line 3:	&dash	- A macro variable constant that is the character “-“ for Windows and a “ “ (space) for Unix
	LOG	- Part of the command syntax that indicates that the SAS LOG output data should be sent to the file identified next.
	&SAS_Logs	- The directory path to the SAS log output file. This is defined this way to allow the same code to execute on both Windows and Unix. By placing the “path names” into a macro variable the fact that Windows uses a “\” and Unix uses a “/” can be over come at run-time by the programmer.
	My_SAS_Log_1.txt	– The name of the SAS Log output file.
Line 4:	&dash	- A macro variable constant that is the character “-“ for Windows and a “ “ (space) for Unix
	PRINT	- Part of the command syntax that indicates that the SAS OUTPUT window output data should be sent to the file identified next.
	&SAS_List	- The directory path to the SAS Listing output file. This is defined this way to allow the same code to execute on both Windows and Unix. By placing the “path names” into a macro variable the fact that Windows uses a “\” and Unix uses a “/” can be over come at run-time by the programmer.
	My_SAS_List_1.txt	– The name of the SAS Listing output file.
Line 5:	&dash	- A macro variable constant that is the character “-“ for Windows and a “ “ (space) for Unix
	WORK	- Part of the command syntax that indicates that the SAS WORK output data should be sent to the directory identified next.
	&SAS_Work	- The directory path to the SAS Work library directory. This is defined this way to allow the same code to execute on both Windows and Unix. By placing the “path names” into a macro variable the fact that Windows uses a “\” and Unix uses a “/” can be over come at run-time by the programmer.
	PGM1_work_dir	– The name of the SAS Work directory.

Line 6:	&init_01	- A macro variable that is needed only for Windows. The value is set to &dash.SASINITIALFOLDER "&SAS_Init.PGM1_init_dir" as in the other lines the &SAS_Init is the path and PGM1_init_dir is the directory to be used. This is not needed for Unix and is set to a space. SAS, when running on Windows, requires an initial start-up folder that is normally assigned to a fixed location by Windows. An attempt to run a second job in a batch mode will cause a conflict that will prevent the second copy from running. Note that running a second copy of SAS by using the "Start" button gives a message about not being able to open the SASUSER.PROFILE file, a WORK.PROFILE file is opened instead.
	&dash	- A macro variable constant that is the character "-" for Windows and a " " (space) for Unix
	SYSPARM	- Part of the command syntax that indicates that the System Parameter String follows.
	&Parm	- Any string of characters, up to 200 characters.
Line 7:	&suffix	- Not required for Windows, on UNIX if &suffix is set to "&&" it will cause the job to run in the background and if set to a space it will cause the job to run in the foreground and tie up the terminal.

NOW HOW DOES THIS WORK?
This author recommends that only tested code in mature environments be built into systems like this. The Master Control Program is the guide for the job; this program sets up, starts, and monitors the rest of the work. Each started job needs to set a flag when it completes. Furthermore, the programmer needs a way to monitor the job progress. Know where the flag files are written, and find a way to examine them while the job is running. Make a copy of a Windows file to check it, the UNIX "tail" or "cat" command can be used there. Since the process being described here starts with a job submitted by the user (you) that means there is a SAS job running in one SAS session that is tied to a terminal. Given the definition of the macro variables above (`Pgm1_Execute` and `Pgm2_Execute`) examine the following code to delay execution for 90 seconds, assuming macro variable shorttime is set to 90:

Another point to bring up is that the next piece of code will only work if it is part of a macro. The %IF, %THEN, %DO, %END, and %UNTL commands; are only active as part of a macro definition. These commands decide at run-time what code will execute. In fact any piece of code with one of these conditional commands will only work inside of a macro.

```
%macro submit_jobs;
%let os_name = &sysscpl;  * set a macro variable os_name to the name of the O/S;
%put &os_name;            * display the O/S Name on the SAS log;
* note use conditional code steps to run x commands one at a time;
   %if &os_name = XP_PRO %then %do;    *  O/S name for Windows XP Professional;
       %do;
          Data _null_;
               * create comm file 1;
               file job_1 noprint notitles linesize=80;
               put 'JOB NOT STARTED';
          Run;
          &Pgm1_Execute;                      * run job one;
          Data _null_;
               A = sleep(&shorttime);          * wait to start next job;
          Run;
       %end;
       %do;
          Data _null_;
               * create comm file 2;
               file job_2 noprint notitles linesize=80;
               put 'JOB NOT STARTED';
          Run;
          &Pgm2_Execute;                 *run job one;
        %end;
   %end;
   %if &os_name = SunOS %then %do;      * one version of UNIX;
       %do;
          Data _null_;
               * create comm file 1;
               file job_1 noprint notitles linesize=80;
               put 'JOB NOT STARTED';
          Run;
          &Pgm1_Execute;                   * run job one;
          Data _null_;
               Call sleep(&shorttime,1);           * wait to start next job;
          Run;
       %end;
       %do;
          Data _null_;
               * create comm file 2;
               file job_2 noprint notitles linesize=80;
               put 'JOB NOT STARTED';
          Run;
          &Pgm2_Execute;              *run job one;
        %end;
   %end;
%mend submit_jobs;
```

* NOTE – you could use a %else instead of a second %if, but why take the chance that the first condition is always false (incorrectly) and always execute the second code segment (sometimes incorrectly)?

WHAT DOES THE SYSPARM DO?

This allows the Master Control Program (calling SAS program) to tell the called program what to do, because the data is being passed when SAS is being invoked for the called program. Given a sysparm string with 4 parameters (job number,YES, NO, and OK) then the following code could convert the values to global macro variables for the called program. The macro variables are then available anywhere in the called program, the sysparm = "1YESNO OK " – ten bytes, without the quotes, including 2 spaces.

The following code would be in My_SAS_Program_1.sas and My_SAS_Program_2.sas with a different parameter value sent to the second program.

```
%global flag test1 test2 test3;
Data _null_;
     Parm_string = sysparm();
     retain flag ' ' Test_a Test_b Test_c '   ';
     flag   = substr(Parm_string,1,1);        Test_a = substr(Parm_string,2,3);
     Test_b = substr(Parm_string,5,3);        Test_c = substr(Parm_string,8,3);
     Call symput('flag' ,flag  );* set macro variable flag to the file number to use;
     Call symput('test1',Test_a);* set macro variable test1 to YES (three bytes);
     Call symput('test2',Test_b);* set macro variable test2 to NO  (three bytes);
     Call symput('test3',Test_c);* set macro variable test3 to OK  (three bytes);
RUN;
* Show the values;
%put parms for flag=&flag test1=&test1 test2=&test2 test3=&test3;
Run;
…  More SAS code  …
Run;
```

HOW DO THE CALLED PROGRAMS TELL THE MPC THEY ARE DONE?

The programs need a way to tell the MCP their status, this is done with a flag file. The calling program should create a file unique for each started job. Then, when the new job starts it should change that file to indicate that the program started successfully, and change it again as the job ends. While not within the scope of this paper the clever use of macros could possibly be setup to run when the job also failed. The implementation here is content to wait in a holding pattern until the user checks the status. But, if a segment fails to complete successfully the programmer will have to take action to shut the system down. A half done program can be restarted. But, a half right program is completely worthless and needs to be discarded. This wastes all of the run-time. A simple “DATA _NULL_” step that writes to the file will work to send messages in all three code locations.

In the Calling program:
First a way to identify and read the files created by the called programs, the calling program has to know the names.

```
* Define all of the test files for the MCP;
Filename Job_1 'c:\any\file\location\unique_file_name_for_each_job1.txt';
Filename Job_2 'c:\any\file\location\unique_file_name_for_each_job2.txt';

*Macro in the MCP to test the files written by the called programs;

%macro test_flag(Flag);
*======================================================================*;
*  check one job for complete.                                         *;
*======================================================================*;
   data _null_;
      test = "&COMPLETE.";   * the macro variable complete is defined below;
      infile Job_&Flag truncover linesize=80;
      input text $char20.;            * read the called program output file;
      if (text = 'COMPLETE') then do;  * did this step finish yet?;
         substr(test,&Flag,1) = '1';    * replace a 0 with a 1;
         call symput('COMPLETE',test); * update variable if a step is done;
      end;
      %put &COMPLETE;    *write the status back to the log;
   run;
%mend test_Flag;
```

Again remember that the next piece of code will only work if it is part of a macro. The %IF, %THEN, %DO, %END, and %UNTL commands; are only active as part of a macro definition. These commands decide at run-time what code will execute. In fact any piece of code with one of these conditional commands will only work inside of a macro.

```
%let longtime = 900; * the number of seconds in 15 minutes;
*==========================================================================*;
*  run step to wait for all of the jobs to finish processing.              *;
*==========================================================================*;
 data _null_;
    call symput('COMPLETE','00'); * one zero for each called program step;
 run;
*=======================================================================*;
*  loop waiting for all jobs to complete.                               *;
*=======================================================================*;
    %do %until(&COMPLETE = 11);
        * check each jobs flag file for the word "COMPLETE" *;
        %test_flag(1);
        %test_flag(2);
        *  wait a while then do it again.        *;
        %if &COMPLETE ne 11 %then %do;
           data _null_;
              %if &os_name = SunOS %then %do;  *UNIX;
                  call sleep(&longtime,1);
              %end;
              %if &os_name = XP_PRO %then %do;  *Windows;
                  a = sleep(&longtime);
              %end;
           run;
        %end;
    %end;
```

In each Called program:

```
* establish a communication link by identifying the called program;
%let flag = Some_value_sent_via_a_parm_from_MCP_like_a_1_or_a_2;
```

```
* Define one of the test files for the called program;
Filename Job_&flag 'c:\any\file\location\unique_file_name_for_each_job1.txt';

     *when the job starts;
     data _null_;
       file job_&flag noprint notitles linesize=80;
       put 'JOB STARTED';
     run;

     … MORE SAS CODE …
     *when the job ends;
     data _null_;
       file job_&flag noprint notitles linesize=80;
       put 'COMPLETE';
     run;
```

NOW, LET'S PUT IT ALL TOGETHER

The following code runs a complete set of programs on Windows. The second half of the code is used to make two programs. The UNIX variations are explained above and left as a student exercise to implement.

```
*************************************************************************;
**   Owl Computer Consultancy, LLC  - Author - William E Benjamin JR      **;
**                                                                        **;
**   Written for WUSS-2008 Coder's Corner presentation                    **;
**   Updated for SGF-2010                                                 **;
**   Purpose: Run two SAS jobs in background on a Windows platform        **;
**   Three code segments need to be placed into three separate files to   **;
**   run, because the main program calls the other two, and they run in a **;
**   background (batch) mode. (blank lines were removed to conserve space)**;
**************************************************************************;
options noxsync noxwait; * Turn off wait commands - run jobs together not in serial  ;
options mprint mlogic symbolgen; *to help debug the processing;
Filename Job_1 'Z:\SGF_2010\logs\flag_file_for_job1.txt';
Filename Job_2 'Z:\SGF_2010\logs\flag_file_for_job2.txt';

%let dash    = -;     * a constant for a dash;
%let suffix  =  ;     * not needed in windows so make it a blank;
%let run_sas = SAS;   * set-up the command to run sas on this os;

* set-up the new work files for the new jobs;
%let SAS_Code = Z:\SGF_2010\code\; * SAS code folder;
%let SAS_Logs = Z:\SGF_2010\logs\; * SAS Logs;
%let SAS_List = Z:\SGF_2010\list\; * SAS Output ;
%let SAS_Work = Z:\SGF_2010\work\; * SAS Work files;
%let SAS_Init = Z:\SGF_2010\init\; * SAS Start Folder;

*create unique init folders for each job; *required in windows not unix;
%let init_01 = &dash.SASINITIALFOLDER "&SAS_Init.PGM1_init_dir";
%let init_02 = &dash.SASINITIALFOLDER "&SAS_Init.PGM2_init_dir";

*create a parm string;
%let parm1 = 1YESNO OK;
%let parm2 = 2NO YESOK;

* set-up the execute command to run SAS in Batch mode;
* use double quotes to resolve the macro variables;

%let Pgm1_Execute =
x &run_sas     "&SAS_code.My_SAS_Program_1.sas"
  &dash.LOG    "&SAS_Logs.My_SAS_Log_1.txt"
  &dash.PRINT  "&SAS_List.My_SAS_List_1.txt"
  &dash.WORK   "&SAS_Work.PGM1_work_dir"
  &init_01     &dash.SYSPARM "&Parm1."
  &suffix;
```

```
%let Pgm2_Execute =
x &run_sas     "&SAS_code.My_SAS_Program_2.sas"
  &dash.LOG    "&SAS_Logs.My_SAS_Log_2.txt"
  &dash.PRINT  "&SAS_List.My_SAS_List_2.txt"
  &dash.WORK   "&SAS_Work.PGM2_work_dir"
  &init_02     &dash.SYSPARM "&Parm2."
  &suffix;

  %put &Pgm1_Execute; * show the command on the log without executing it here;
  %put &Pgm2_Execute; * This is a great debugging tool, to see the command built here;
%let shorttime = 10;  * set to 10 seconds but 900 is the number of seconds in 15 minutes;

%macro submit_jobs;   * Start the definition of a macro - execute this macro at bottom of code;
%let os_name = &sysscpl;              * set a macro variable os_name to the name of the O/S;
%put &os_name;                         * display the O/S Name on the SAS log;
*note the use of conditional code steps to run x commands one at a time - add as many as you need;
   %if &os_name = XP_PRO %then %do;   *  O/S name for Windows XP Professional;
          Data _null_;
               * create comm file 1;
               file job_1 noprint notitles linesize=80;
               put 'JOB NOT STARTED';
          Run;
          &Pgm1_Execute;                       * run job one;
          Data _null_;
               A = sleep(&shorttime); * regulate when the next job starts by increasing time;
          Run;
   %end;
   %if &os_name = XP_PRO %then %do;   *  O/S name for Windows XP Professional;
          Data _null_;
               * create comm file 2;
               file job_2 noprint notitles linesize=80;
               put 'JOB NOT STARTED';
          Run;
          &Pgm2_Execute;                  *run job one;
   %end;
%mend submit_jobs;

%macro test_flag(Flag); *Start the definition of a macro - execute this macro in data _null_ step;
*====================================================================*;
*  check one job for complete.                                       *;
*====================================================================*;
   data _null_;
      test = "&COMPLETE.";   * the macro variable complete is defined below;
      infile Job_&Flag truncover linesize=80;
      input text $char20.;            * read the called program output file;
      if (text = 'COMPLETE') then do;  * did this step finish yet?;
         substr(test,&Flag,1) = '1';   * replace a 0 with a 1;
         call sympout('COMPLETE',test); * update variable if a step is done;
      end;
      %put &COMPLETE;    *write the status back to the log;
   run;
%mend test_Flag;

%let longtime = 60; * 900 is the number of seconds in 15 minutes;
%macro test_for_done; * Start the definition of a macro - execute this macro at bottom of code;
*========================================================================*;
*  run step to wait for all of the jobs to finish processing.            *;
*========================================================================*;
   data _null_;
      call sympout('COMPLETE','00'); * one zero for each program step;
   run;
   * loop waiting for all jobs to complete*;
   %do %until(&COMPLETE = 11);
      *check each jobs flag file for the word "COMPLETE"*;
      %test_flag(1);
      %test_flag(2);
      *  wait a while then do it again. *;
      %if &COMPLETE ne 11 %then %do;
         data _null_;
                a = sleep(&longtime);
         run;
      %end;
   %end;
%mend test_for_done;
*do the work - run the jobs;
%submit_jobs;
%test_for_done;
* MCP job done;
```

Split here and make two identical jobs using the following code, first make them work, then make them different:

```
*first read the parms;
%global flag test1 test2 test3;
Data _null_;
     Parm_string = sysparm();
     retain flag ' ' Test_a Test_b Test_c '   ';
     flag   = substr(Parm_string,1,1);     Test_a = substr(Parm_string,2,3);
     Test_b = substr(Parm_string,5,3);     Test_c = substr(Parm_string,8,3);
     Call symput('flag' ,flag  );   * set macro variable flag to the file number to use;
     Call symput('test1',Test_a);   * set macro variable test1 to YES/NO (three bytes);
     Call symput('test2',Test_b);   * set macro variable test2 to YES/NO (three bytes);
     Call symput('test3',Test_c);   * set macro variable test3 to OK  (three bytes);
RUN;
* Show the values;
%put parms for flag=&flag test1=&test1 test2=&test2 test3=&test3;
Run;

* Define one of the test files for the called program;
Filename Job_&flag "Z:\WUSS_2008\logs\flag_file_for_job&flag..txt"; * two periods are required;
*when the job starts;
data _null_;
   file job_&flag noprint notitles linesize=80;
   put 'JOB STARTED';
run;
Data _null_; *job one and two; * put your real job here;
    A = sleep(90);
RUN;
*when the job ends;
data _null_;
   file job_&flag noprint notitles linesize=80;
   put 'COMPLETE';
run;
```

CONCLUSION

The techniques shown here go beyond the simple macro usage seen in most programs, but hopefully will launch a whole new level of understanding about the real power of the simple SAS macro, and how it can be used to increase throughput of processing your tasks.

CONTACT INFORMATION

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

Name	William E Benjamin Jr
Enterprise	Owl Computer Consultancy, LLC
Work Phone:	602-942-0370
Fax:	602-942-3204
E-mail:	William@owlcomputerconsultancy.com
Web:	owlcomputerconsultancy.com