PharmaSUG 2014 - Paper AD20

Managing bulk SAS job submissions with post execution analysis and notification.

Bruce Kayton, Simulstat Inc., San Diego, CA

ABSTRACT

Running all the programs in a study analysis with all their dependencies can be a time consuming or inefficient if done individually or sequentially. This paper outlines a methodology to use driving data to manage a job submission and monitoring process using defined dependencies. This enables programs to run efficiently in quick succession or often simultaneously to best utilize available resources.

Upon the completion of all programs a submission summary is generated with run times and log analyses highlighting any warning or error issues. The submitter is notified via email about job completion with a hyperlink to the log analysis in spreadsheet format. Detailed line items contain links to submitted program logs along with hyperlinks to identified errors or warnings if applicable.

Benefits:

- Hands off submission, gives users the ability to focus on other tasks whilst programs run unattended in the background.
- Efficient use of resources allowing programs that aren't dependent on each other to execute simultaneously.
- Timely awareness of issues upon job completion that can be linked to an automated notification to responsible parties.
- Quick access to program logs and error conditions. Summaries gives management a high level view of the status of an analysis.
- Quantified summary detail to determine time required to consistently run a full analysis.
- Standardized error and warning evaluation.

ASSUMPTIONS

This paper outlines a methodology to approach bulk submission and management of programs. It's developed specifically in a Clinical reporting environment using Unix and leans towards distributed SAS grid processing, but the methodology can easily be applied in a number of different environments.

There are basic assumptions and program location infrastructure that needs to be in place for this to work successfully.

- A hierarchical structure of required programs that are required to be executed.
- Typical Clinical programs would include SDTM, ADaM, Tables, Figures and Listings and their associated validation counterparts.
- Driving data with basic detail denoting derivation order and program name. Optional: path, email addresses or user ids of responsible parties.

  Table 1. is a sample of driving data detail:

Derive Order	SDF Label	Program Name	Programmer	Testing Program Name	Tester
4	Adverse Events	ae.sas	Programmer A	v-ae.sas	Programmer C
3	BMD	bm.sas	Programmer A	v-bm.sas	Programmer C
2	Concomitant Medication	cm.sas	Programmer B	v-cm.sas	Programmer C
2	Comments	co.sas	Programmer A	v-co.sas	Programmer D
2	Disorder Findings	df.sas	Programmer B	v-df.sas	Programmer D
1	Subject Demographics	dm.sas	Programmer A	v-dm.sas	Programmer D

  Table 1.

Process

1. Import driver data from external source into a SAS dataset, setting runstatus to 99.

   Can be in any format, typically an excel spreadsheet for easy maintenance.

2. Use driver data to control job submission. This step would be part of a looping process that would continue repeating until no observations had a runstatus of 99. i.e. All programs submitted.

```
filename subcmds library 'work.temp.submits.source';

data _null_;
  set driver end=eof;
  where runstatus=99;
  by validation derive_order seq;
  file subcmds noprint;
  put '%' 'runpgm(' pgmpath +(-1) ',' program_name +(-1) ');';
  if last.derive_order and not eof then do;
    call symput('wait','1');
    stop;
  end;
  runsubmit+1;
  /* submit up to 50 programs at a time, or whatever is acceptable */
  if runsubmit > 50 - input("&running",8.)-1 or eof then do;
    stop;
  end;
run;

%include subcmds;
```

   Macro %runpgm would contain the executable system SAS command. When called it would submit each program whilst updating runstatus to 2 for that program. At this point alternate runtime execution parameters could also be parsed. e.g. memsize or alternate log destinations.

3. Monitor job status. This can be done in a number of ways. Typically looking at jobs steams using ps command, or looking for the tail end of a log for completion. When a program is completed runstatus would be set to 0.

4. Reiterate process from step 2 until all runstatus codes are set to 0.

5. **NOTE:** The iterative process above can go into an infinite loop unless you implement a contingency exit strategy.

   a. Allow for a finite number of allowable iterations.

   b. If iteration count is exceeded, determine which program(s) are still running and determine expected run time from previous logs.

   c. Exit loop if program is running for longer than 5 times normal. This allows for strained resources, but won't let the process run indefinitely. At this point assume the program executing is looping.

   This should in most situations not occur as each program in theory by this stage would have been unit tested before being ready to run in bulk.

6. Once all jobs are completed, i.e. sum of runstatus for all programs is zero. Scan logs and extract WARNING, ERROR or other conditions that may warrant investigation. e.g. Cartesian joins, uninitialized variables. CPU, Elapsed start and end times can also be retrieved for detailed and summary reporting at this time.

   Log **Error! Reference source not found.** shows the last lines of a log to determine CPU and Elapsed time.

```
NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
NOTE: The SAS System used:
      real time           12.54 seconds
      cpu time            6.46 seconds
```

7. Retrieve creation time of the log. This is the end time for the program.

```
%macro getlogdt(path,logname);
   %let enddttm=01JAN1960:00:00:00;
   data _null_;
     length infoname infoval $ 180;
     format enddttm datetime.;
     drop rc fid infonum i close;
     rc=filename("lfile","&path./&logname");
     fid=fopen("lfile");
     infonum=foptnum(fid);
     do i=1 to infonum;
        infoname=foptname(fid,i);
        infoval=finfo(fid,infoname);
        if infoname='Last Modified' then do; /*Mon Apr 29 16:20:50 2013*/
          enddttm=input(scan(infoval,3,' ')||scan(infoval,2,' ')||
                  scan(infoval,-1,' ')||':'||scan(infoval,4,' '),datetime32.);
          call symput('enddttm',put(enddttm,datetime.));
        end;
     end;
     close=fclose(fid);
     stop;
   run;
%mend;
```

Macro variable &enddttm contains end time for program, derived from creation time of log. Deduct 'real time' above to determine actual program start time.

8. Generate report from accumulated data. Table 2 contains an example of the program execution summary.

Program Type	Number of Programs	Aggregate Elapsed Time	Aggregate CPU Time	Average Elapsed Time	Average CPU Time	Programs with Check script issues
SDTM QC	17	0:02:32.17	0:01:16.62	0:00:08.95	0:00:04.51	14
SDTM Total	17	0:02:32.17	0:01:16.62	0:00:08.95	0:00:04.51	14
ADAM	13	0:03:49.02	0:01:06.51	0:00:17.62	0:00:05.12	4
ADAM QC	13	0:03:53.79	0:01:40.85	0:00:17.98	0:00:07.76	7
ADAM Total	26	0:07:42.81	0:02:47.36	0:00:17.80	0:00:06.44	11
TABLES	13	0:01:43.51	0:01:00.47	0:00:07.96	0:00:04.65	7
TABLES QC	5	0:00:18.76	0:00:07.56	0:00:03.75	0:00:01.51	3
TABLES Total	18	0:02:02.27	0:01:08.03	0:00:06.79	0:00:03.78	10
ALL Programs	61	0:12:17.25	0:05:12.01	0:00:12.09	0:00:05.11	35

Table 2.

Table 3 contains program execution detail, containing hyperlinks to log files, Error or Warning detail and/or links to responsible programmers assigned to programs needing attention.

					Output from check script					
Program Type	Start Date	End Date	Time, Start to Finish	CPU Time	Has Errors?	Has Warnings	Other	Log Name	Status	Programmer to notify
ADAM	11-Feb-2014 04:30:30	11-Feb-2014 04:30:41	0:00:11.11	0:00:02.89	N	N	N	aslinfo.log		
ADAM	11-Feb-2014 04:30:49	11-Feb-2014 04:31:15	0:00:25.98	0:00:08.41	N	N	N	acm.log		
ADAM	11-Feb-2014 04:30:51	11-Feb-2014 04:31:06	0:00:14.99	0:00:03.19	N	N	N	aex.log		
ADAM	11-Feb-2014 04:30:54	11-Feb-2014 04:31:15	0:00:20.93	0:00:05.47	N	N	Y	aae.log		Programmer A
ADAM	11-Feb-2014 04:30:54	11-Feb-2014 04:31:11	0:00:16.65	0:00:04.32	N	N	N	abmdxa.log		
ADAM	11-Feb-2014 04:30:54	11-Feb-2014 04:31:16	0:00:22.30	0:00:05.82	N	Y	N	amh.log		Programmer B

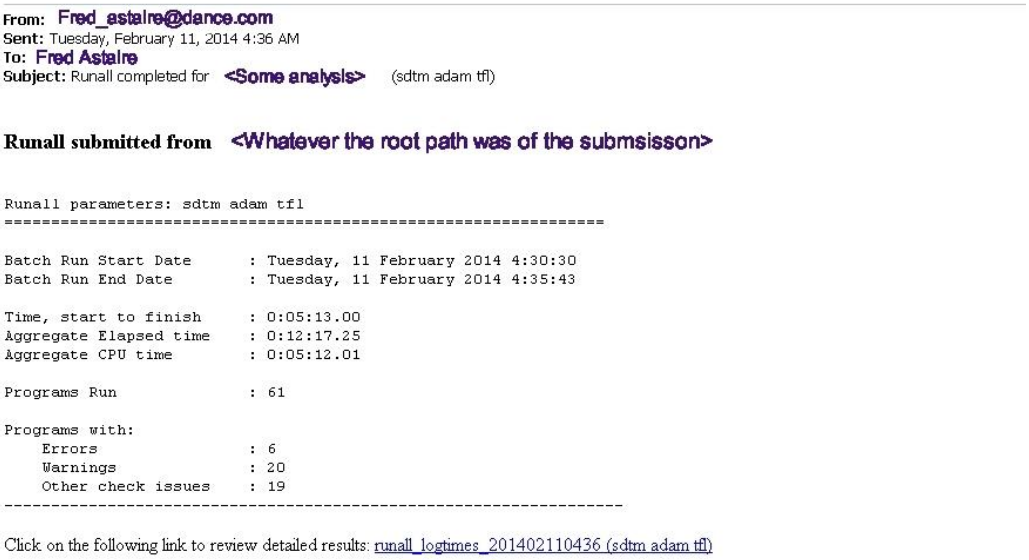
Table 4 is a subset of the log containing details of Error, Warning or Other criteria flagged for investigation.

Log Name	
aae.log	[line 18627] NOTE: Invalid argument to function INPUT at line 3124 column 75.
	[line 18632] NOTE: Invalid argument to function INPUT at line 3124 column 75.
	[line 18637] NOTE: Invalid argument to function INPUT at line 3124 column 75.
	[line 18642] NOTE: Mathematical operations could not be performed at the following places. The results

9. Notify job submitter of overall job completion by email. The report output above would be written to a permanent location wheras the email notification would repeat some of the summary information, but have embedded hyperlinks to the report for detailed information.

```
filename outbox email
         to="&sysuserid.@XXXXX.com"
         bcc="bkayton@XXXXX.com"
         type='text/html'
         subject="Runall completed for &protocol &anatype (&sysparm.)";
data _null_;
  file outbox;
  put '<html><head>';
  put '<style type="text/css">';
  put 'table {font-family : arial,times,courier;font-size : 10pt}';
  put 'thead {background  : lightblue;';
  put '       color       : blue;';
  put '       font-weight : bold;';
  put '       }';
  put '.bt   {font-weight : bold}';
  put '.cb   {color       : blue}';
  put '</style>';
  put '</head>';
  put '<body>';
  put "<h3>Runall submitted from %sysget(PWD), by &sysuserid</h3>";
  put ' ';
  put '<pre>';
  put "Runall parameters: &sysparm";
  put "============================================================";
  put ' ';
  put "Batch Run Start Date      : &startdt &starttm";
  put "Batch Run End Date        : &enddt &endtm";
  put ' ';
  put "Time, start to finish     : &elapsed";
  put "Aggregate Elapsed time    : &totrun";
  put "Aggregate CPU time        : &totcpu";
  put ' ';
  put "Programs Run              : &pgmcnt";
  put " ";
  put "Programs with:";
  put "    Errors                : &errcnt";
  put "    Warnings              : &warncnt";
  put "    Other check issues    : &othcnt";
  %if &looping ne 0 %then %do;
    put " ";
    put "IMPORTANT: Program(s) running for an abnormally long time. Potential looping.";
    put "=======================================";
    put "    Programs  : &looping &looppgms";
    put " ";
  %end;
  put '------------------------------------------------------------------';
  put ' ';
  put '</pre>';
  put 'Click on the following link to review detailed results:' @;
  put '<a href="\\server\' "%sysfunc(translate(%substr(&xlsout,11),/,\))" '">' "%scan(&xlsout,-2,/.)" '</a>';
  put '</body></html>';
  stop;
run;
```

Sample of email notification for distribution to submitter.



From: Fred_astaire@dance.com
Sent: Tuesday, February 11, 2014 4:36 AM
To: Fred Astaire
Subject: Runall completed for <Some analysis> (sdtm adam tfl)

Runall submitted from <Whatever the root path was of the submsisson>

```
Runall parameters: sdtm adam tfl
============================================================

Batch Run Start Date      : Tuesday, 11 February 2014 4:30:30
Batch Run End Date        : Tuesday, 11 February 2014 4:35:43

Time, start to finish     : 0:05:13.00
Aggregate Elapsed time    : 0:12:17.25
Aggregate CPU time        : 0:05:12.01

Programs Run              : 61

Programs with:
    Errors                : 6
    Warnings              : 20
    Other check issues    : 19
------------------------------------------------------------------
```

Click on the following link to review detailed results: runall_logtimes_201402110436 (sdtm adam tfl)

CONCLUSION

This methodology is not meant to replace the systematic execution and unit testing of individual programs, but facilitates an automated process for rerunning stable programs where only data changes may affect the outcomes. Its reporting mechanism adds a layer of communication to the program execution which is helpful to individual programmers and management. Its functionality lends itself well to run using crontabs that can run hands off during the night, or during the day whilst freeing the user to focus on other required tasks. It is able to be customized to accommodate unusual circumstances, for example file locks, and can resubmit programs if needed should those circumstances arise.

CONTACT INFORMATION

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

Bruce Kayton
Simulstat Inc.
San Diego CA, 92128
brucekayton@gmail.com