Paper 006-2010

PRAGMATIC APPROACH TO RESUBMIT FAILED JOBS IN PRODUCTION ENVIRONMENT(s)

Salman Ali, Element Technologies Inc, Piscataway, NJ

Abstract

Companies spend thousands of dollars for operations support and maintenance. During production runs, companies have full-time staff to monitor the execution of SAS® jobs, and in case a job aborts, to resubmit them through manual means. Sometimes it can become painful to monitor all SAS® jobs running in production through the scheduler. For instance, a production support analyst cannot monitor every job in production, and he or she might not be aware if a legacy SAS® job aborts in the middle of the night. This can create serious problems especially in companies which rely on real time data. If a job aborts and the production support team is not aware of it, the business users can lose real time data and will not be able to make accurate decisions. This paper will focus on reading the ftplogs to check for failed FTP transmission and successfully resubmitting the bat files associated with the failed transmissions without any manual intervention. The paper will also discuss the use of 'SYSPARM' automatic variable to conditionally execute the SAS® code.

Introduction

In this paper, I will summarize using SAS® Base programming to create a utility which will search for failed jobs in a production environment and resubmitting them for successful execution. This paper will provide some detail surrounding how to create a SAS® process that will monitor log files generated by FTP scripts through a SAS® program in a production environment. Also, the paper will discuss programming techniques to effectively read the directory structure and to use Perl regular expressions to search for failed jobs in a production environment. This part of the discussion is also relevant to the UNIX environment; however, the primary focus of the discussion will be within the Windows environment. The SAS® utility explained in this paper will provide real-time monitoring of programs and is written to ease the support of personnel running SAS® batch jobs in a Production environment. If deployed correctly, it can be an indispensable tool for quickly detecting and resubmitting failed jobs without any manual intervention. It is recommended to submit this process during minimal activity on your production server, the point being to search the failed FTP logs when no FTP transmission is taking place.

Figure 1 depicts a typical environment in which a SAS® job creates one or more datasets and then outputs reports and/or flat files. The output reports and/or flat files are then sent via FTP to a remote server location so that they can be further transferred to a final destination (business users or warehouse).

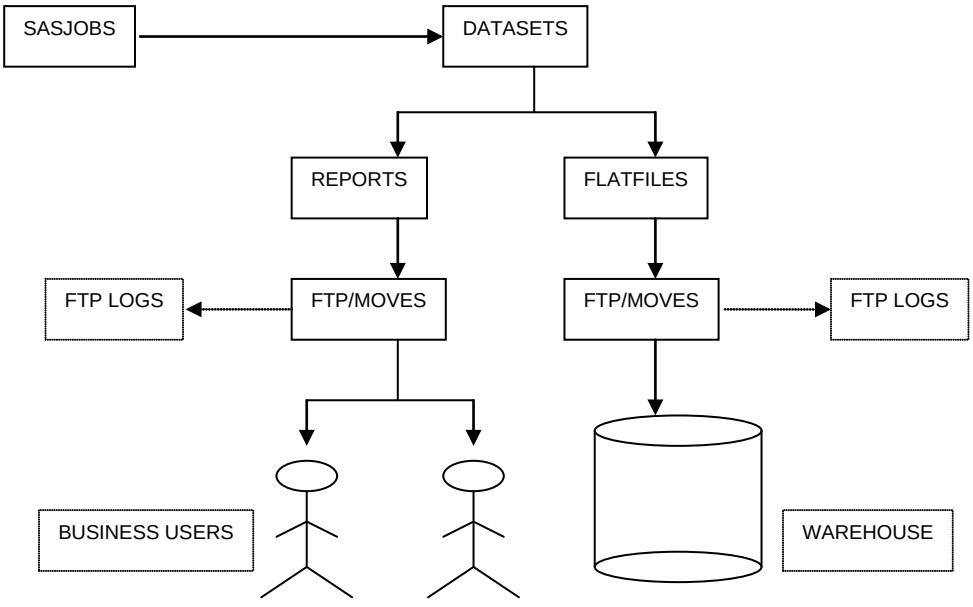

Figure 1

Batch Submission

Companies have been moving toward increasing reliance upon scheduled products to coordinate the scheduling and execution of a large percentage of regular SAS® batch jobs (including languages other than SAS®). Scheduling for a large number of SAS® jobs is usually handled by a scheduler product. However, there are several options for scheduling SAS® jobs to run at later times and on a repetitive basis, but this paper will not address this particular functionality.

The Ultimate Solution

In a nutshell, a SAS® program can produce any number of outputs on a production platform. Some of the few are flat files, reports, datasets, logs, bat files etc. In turn, any of these outputs on a production platform can be distributed to multiple locations. For instance, a SAS® program can direct flat files to an output directory, or a log file can be directed to a log directory, etc. More explicitly, suppose that a developer wrote a program called sample_jobA.sas and that it created flat files from a transactional database. The developer can use FTP commands to upload those flat files to one or more final destinations.

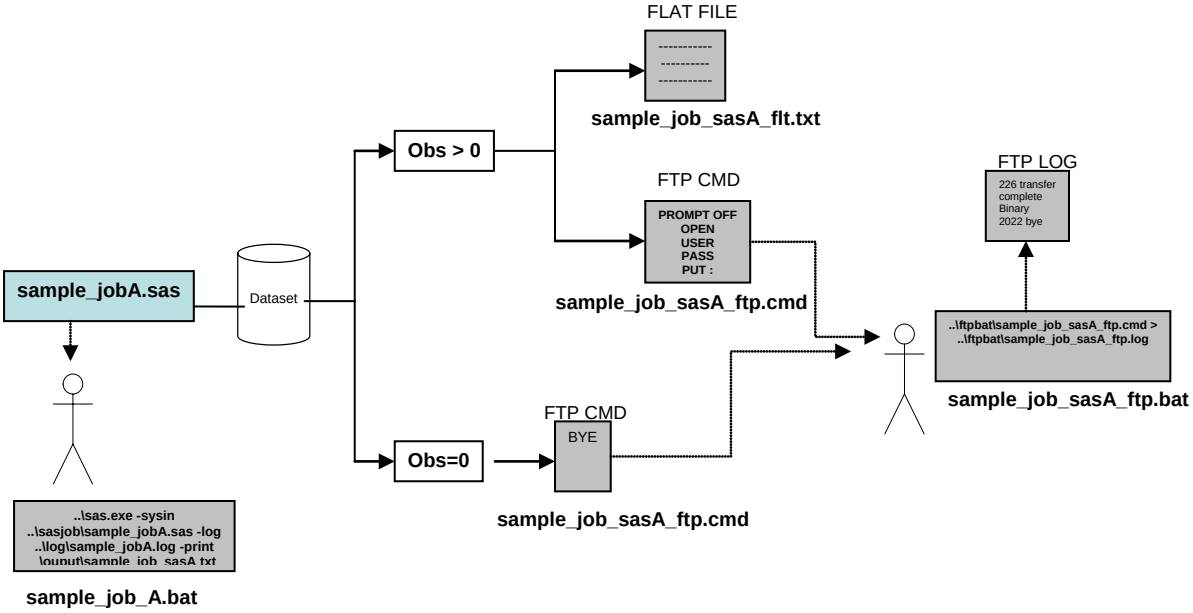

Figure 1A

Create a Simple BAT File

To submit a SAS® job in batch mode, you need to create a simple BAT file whose contents will look something like those in Figure 2 below.

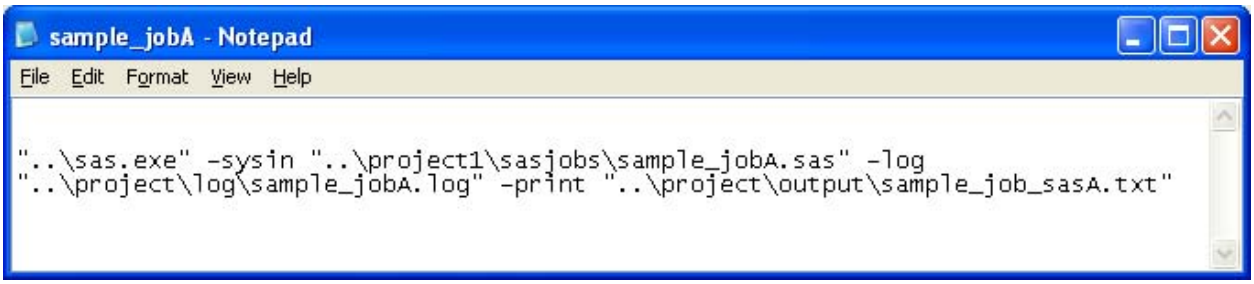

Figure 2 : sample_jobA.bat

Once the sample_jobA.bat is created in an FTPBAT folder, you can use your scheduling product to execute the bat script. Once output files, reports, flat files, or whatever form the output has taken have been created, a scheduling system will distribute/move the necessary files off the production platform to their destinations via FTP.

Create a Simple SAS® Job

In order to continue the example using sample_jobA.bat, assume that the sample_jobA.bat is executed through another program called run_sample_jobA.sas. Figure 3 below uses the SAS® editor window and X command (a way for SAS® to execute DOS command directly) to call the BAT script to run the original SAS® job sample_jobA.sas.

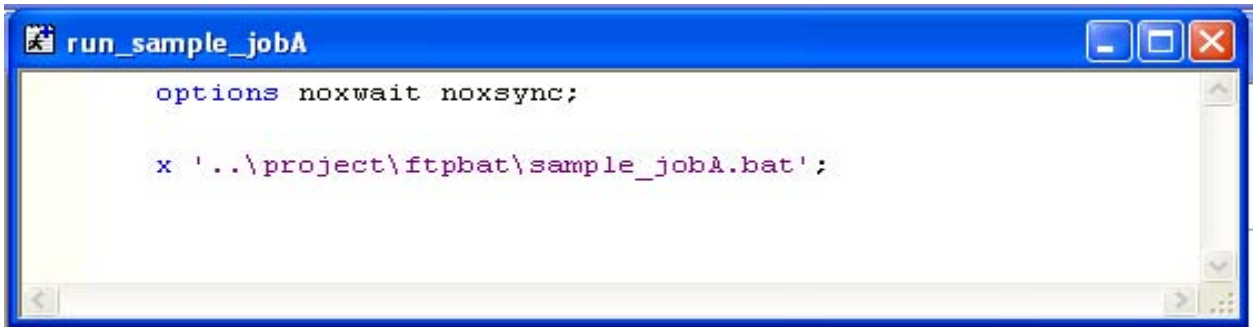

Figure 3 : run_sample_jobA.sas

To move the output to the final landing area or to the warehouse, an FTP script can be setup to dynamically upload the flat files and reports to their respective landing areas. An alternative method to an FTP script is to use a macro within a SAS® program file which will write FTP commands and instruction to a command file that is later called by an FTP process.

Creating an FTP Script and Executing It

Remember that sample_jobA.sas is programmed to write instructions for the FTP process - not actually executing the FTP process. The actual FTP execution is run later by the scheduler. SAS® job sample_jobA.sas will write FTP instructions to a command file called sample_job_sasA_ftp.cmd that will be later called by an FTP process through a scheduler. Figure 4 below shows the contents of the sample_job_sasA_ftp.cmd file which is created by a macro within the SAS® job sample_jobA.sas.

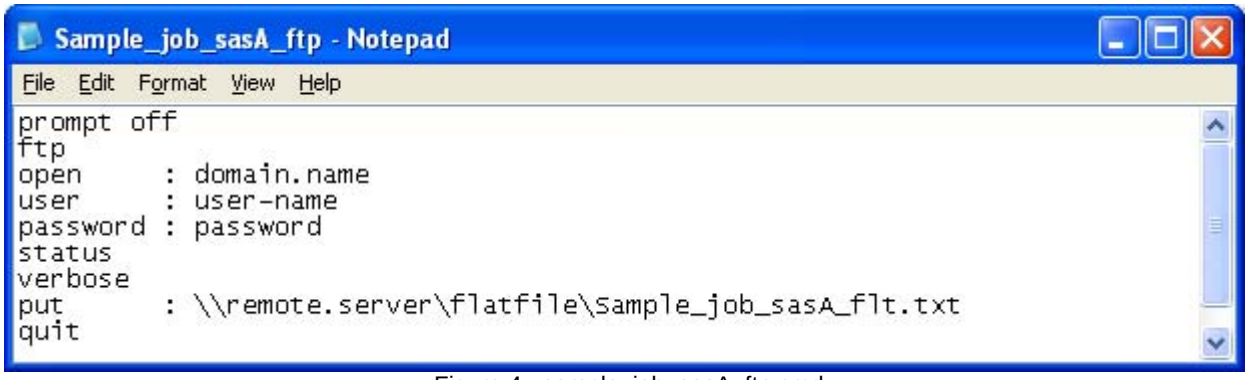

Figure 4 : sample_job_sasA_ftp.cmd

In case your flat file has no observations due to business rules, then the sample_job_sasA_ftp.cmd file will be dynamically written within SAS® job to have only the 'BYE' parameter, which will end the FTP session and will exit the FTP client. Figure 5 below shows the contents of sample_job_sasA_ftp.cmd when the flat file created by the SAS® job contains no observation.

Figure 5 : sample_job_sasA_ftp.cmd

Up to this point the user has produced the output file sample_job_sasA_flt.txt by a SAS® program in the flat file directory, a log file which will be directed into a log directory, and a CMD file sample_job_sasA_ftp.cmd generated by the sample_jobA.sas job which will be directed into the FTPBAT directory. It is important to note that the standards and naming conventions for your SAS® jobs, logs, output and FTP scripts should be consistent and clear. To run the sample_job_sasA_ftp.cmd, I will create another BAT file sample_job_sasA_ftp.bat which will generate the FTP log associated with sample_job_sasA_ftp.cmd file producing more information about the FTP transmission. Figure 6 shows the contents of the sample_job_sasA_ftp.bat file.

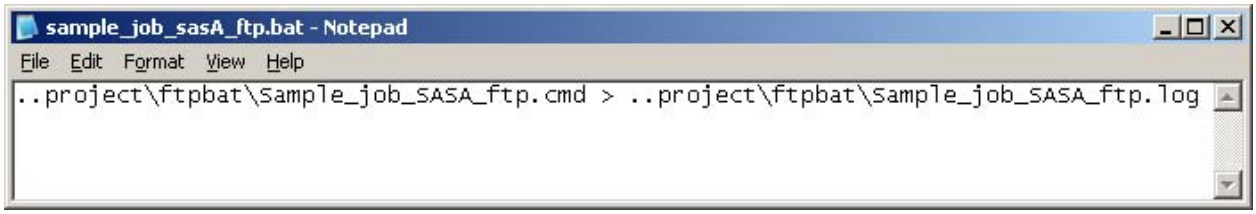

Figure 6 : sample_job_sasA_ftp.bat

The BAT file sample_job_sasA_ftp.bat will be executed using the X command in a SAS® program called run_sample_job_sasA_ftp.sas. Figure 7 below displays a SAS® editor with instructions on how to call sample_job_sasA_ftp.bat which will dynamically run sample_job_sasA_ftp.cmd and create an FTP log with details in the FTPBAT folder.

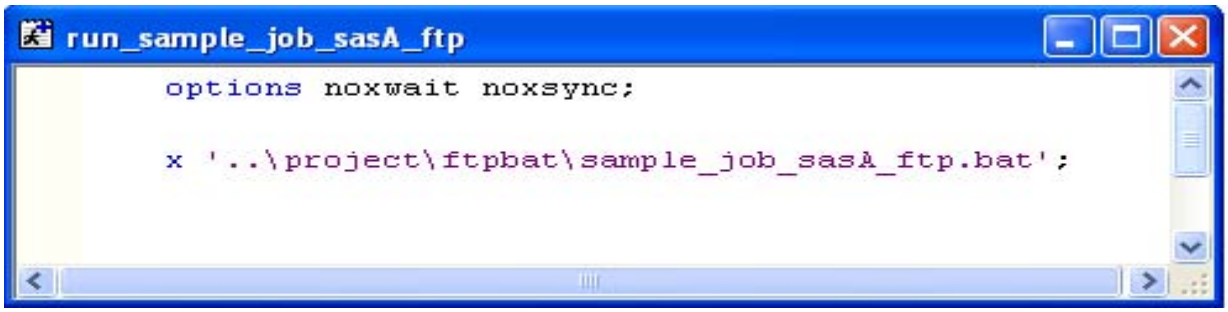

Figure 7 : run_sample_job_sasA_ftp.sas

Creating an FTP LOG

Once the FTP process for sample_job_sasA_ftp.cmd is called to upload the flat file sample_job_sasA_flt.txt, it will generate a log file called sample_job_sasA_ftp.log which, if successful, will resemble the text below in Figure 8. The SAS® utility explained from this point in the paper will focus on reading the FTP log in Figure 8 and checking if the FTP transmission was successful.

```
200 PORT command successful.
150 Opening BINARY mode data connection for  sample_job_sasA_flt.txt .
226 Transfer complete
90870 bytes sent in 12.31 seconds <7.38 Kbytes/sec>
```

Figure 8 : sample_job_sasA_ftp.log

Quick Summary

Before proceeding, the following summarizes everything covered so far. There are three basic steps required to submit your SAS® Batch Job:

1. The scheduler will submit your SAS® job sample_jobA.sas through a batch file sample_jobA.bat. A separate program run_sample_jobA.sas will execute sample_jobA.sas.
2. Another SAS® job run_sample_job_sasA_ftp.sas will execute to dynamically call sample_job_sasA_ftp.cmd causing its own execution.
3. SAS® job run_sample_job_sasA_ftp.sas is related to sample_jobA.sas. SAS® job run_sample_job_sasA_ftp.sas should always be dependent on run_sample_jobA.sas. For example
    a. The scheduler will run a batch file sample_jobA.bat first.
    b. After SAS® job sample_jobA.sas successful execution the scheduler will then run the BAT file sample_job_sasA_ftp.bat

Directory Structure

It is also important to understand the file structure of your environment. Figure 9 shows the layout of a typical directory structure utilized for multiple projects. The SAS® utility will read the directories which are organized in a hierarchical structure, the top most directory therein containing the project name. The project directory can contain many subdirectories such as output, logs, ftpbat, reports, datasets, etc. among many more. In this case, we are only interested in the ftpbat directory which will have sample_job_sasA_ftp.log which contains details about the FTP transmission.

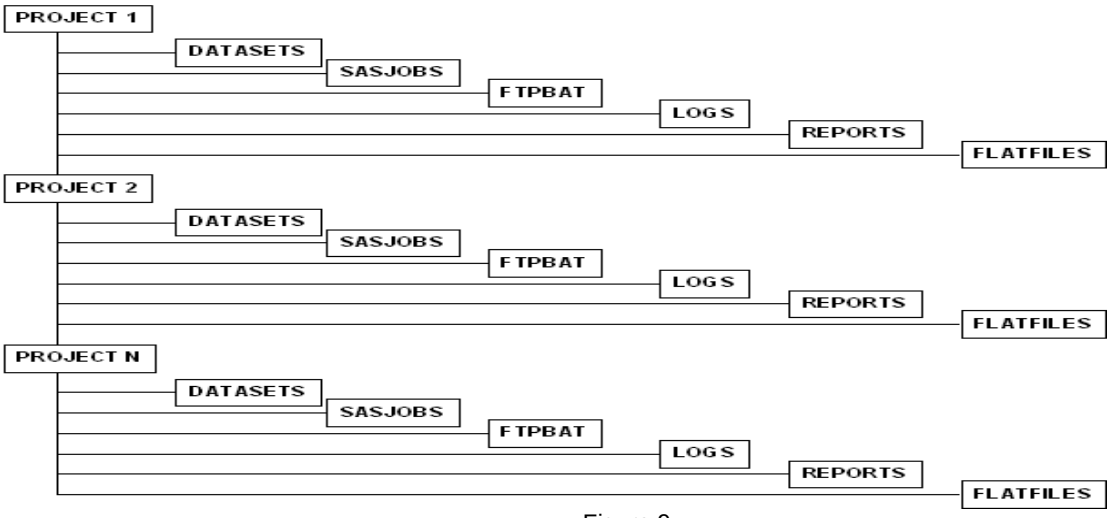

Figure 9

SAS® Process to Search for Failed FTP Transmissions

Step 1: Read Directory Structure

Assume we have CLINPROJ as the parent directory for Project1, Project2 …. Project N. In the SAS® editor use the filename pipe option to read in the output of the windows operating system command into a SAS® dataset. The option /AD /S /B will list information about all the sub-directories with absolute paths.

```
filename shr pipe "dir \\windows.cbt\clinical /AD /S /B ";
```

The next section uses a data step to read the contents of the executed dir DOS command into a SAS® dataset and to extract only the path names of each FTPBAT directory.

```
Data shr1;
        length fbat_path $2000;
        infile shr length=reclen;
        input fbat_path $varying2000. reclen;

                if substr(fbat_path,length(fbat_path)-5)='ftpbat';
 run;
```

Another data step is used to obtain the number of observation in dataset shr1 for further processing.

```
Data _null_;
        If 0 then set shr1 nobs=count;
        Call symputx('Count',Count);
STOP;
RUN;
```

Step 2: Storing Absolute Path Names As Macro Variables

This step inputs all FTPBAT absolute pathnames into macro variables for further processing

```
proc sql noprint;
        select fbat_path
        into :fbat_abs_path1-:fbat_abs_path%left(&count)
        from shr1;
quit;
```

Step 3: Read the FTP log from the FTPBAT Directory

In this section the pipe dir command is employed again to list all the FTPLOG files ending with *ftp.log. Once located, the information from the dir command will be stored in a number of datasets which later will be stacked together within the final staging dataset called dircmd.

```
%macro read_ftp_log;

        %let      readflog =\*ftp.log;
        %do i=1 %to &count;
              /*********WINDOWS PATH TO DIRECTORY THE FILES OF WHICH TO SEARCH*******/
                        filename dir&i  pipe "dir /a-d /-c &&fbat_abs_path&i&readflog" ;
              /*************************************************************************************************/
        %end;/*i=1 %to &count;*/

%mend read_ftp_log;

            /***********CREATE DATASETS TO STORE CONTENTS OF DIRECTORY***************/

%macro store_and_combine_ftp_log;

        %do i=1 %to &count;
                data file&i;
                length line $2000;
                infile dir&i length=len truncover;
                input @;
                input line $varying2000. len;
                run;
        %end;/*i=1 %to &count;*/

        data dircmd;
                set
                %do i = 1 %to &count;
                        file&i
                %end;/*i=1 %to &count;*/
                ;
        run;

%mend store_and_combine_ftp_log;
```

The contents of dataset dircmd will look something like the following text (Figure 10):

```
 Volume Serial Number is 8293-91B2
 Directory of ..\project1\ftpbat\
 09/27/2009  07:14 PM               5 sample_job_sasA_ftp.log
 09/27/2009  07:14 PM               5 sample_job_sasB_ftp.log

          2 File(s)   246  bytes
            0 Dir(s)     32619 bytes free
```

Figure 10

Step 4: Eliminate Extraneous Data

The next step will be cleaning and parsing out extraneous data to isolate the absolute path for the ftp log files. This SAS® process will be interested in log files that are older than 2 days from current date. In the production run, we will only search for FTP logs which were aborted within past two days. However, in your environment, you have the option to extend your search period for more than two days

```
            /***************CREATE DATASET TO PARSE OUT EXTRANEOUS DATA****************/
                data filein ;
                  set dircmd;

               /****************KEEP LINES WITH FILES AND DIRECTORY NAME *****************/

                if line ne '' and not
                index(line, 'Volume')  and not
                index(line, 'File(s)') and not
                index(line, 'Total Files Listed:') and not
                index(line, 'bytes free');
```

```
        /**OBTAIN DIRECTORY NAME AND RETAIN IT FOR ALL FILES IN THAT DIRECTORY******/

                if index(line, 'Directory of') then do;
                    directory = scan(upcase(line),3,'');
                            retain directory ;
                end;/*index(line, 'Directory of')/*

                 /****************** KEEP FILE NAMES, DATE AND TIME ************************/

                if not index(upcase(line), 'DIRECTORY OF');

                    /*******************FILE PATH, DATE, TIME, FILESIZE**********************/

                file_name   = upcase(scan(line,4,''));
                file_ftp_log = compress(directory||'\'||file_name);
                 date       = input(substr(line, 1, 10), mmddyy10.);

                   /************************FILES ONLY TWO DAYS OLD *************************/

                diff_date  = today()- date;
                if diff_date =< 2;
        run;
```

Step 5: Search through FTP Logs Using Perl Regular Expressions

The next step uses Perl regular expressions to read through all the FTP log files searching for failed jobs. The SAS® code below shows a regular expression compiled on the first iteration of the data step (initialized when _n_ = 1). Be sure to add a retain statement to the SAS® code which will ensure that the regular expression is carried through the dataset to all proceeding observations. The text 'ftp\>\ bye|bye|226 transfer complete|226 Binary Transfer complete' within the log files indicates successful FTP transmission.

Skip this text within your search ('ftp\>\ bye|bye|226 transfer complete|226 Binary Transfer complete') to focus only on text which indicates an unsuccessful FTP transmission. The final dataset will contain a list of FTP logs which were not successful or not submitted properly.

```
 /******CREATE DATASET TO SEARCH STRINGS "FTP>BYE, BYE, 226 AND 250 TRANSFER COMPLETE*******/

        data dsout;
                length file2read $256 line $10000 ;
                set filein(keep= file_ftp_log);

/*      READ FROM FILE2READ, MATCH REGULAR EXPRESSION TO CHARS IN LINE READ FROM FILE.
         IF MATCH, OUTPUT*/

                file2read = file_ftp_log ;
                infile dummy filevar=file2read end=lastobs length=reclen;

        /*************
                METACHARACTERS:
                THE i INDICATES A CASE INSENSITIVE SEARCH
                THE ^ INDICATES FIRST SEARCH STRING MATCH
                THE | INDICATES AN EITHER/OR BEHAVIOR BY SEPARATING TWO OR MORE POSSIBLE
                CHOICES
        **********/

        /*********
                PRXPARSE FUNCTION COMPILES A PERL REGULAR EXPRESSION (PRX) THAT
                CAN BE USED FOR PATTERN MATCHING OF A CHARACTER VALUE
        ***********/

                if _N_ = 1 then do;
                        retain     perlrx;

                perlrx = prxparse("/^(ftp\>\ bye|bye|226 transfer complete|226 Binary Transfer complete)/i");
```

```
        end;/*_N_ = 1 */

        do while(not lastobs) ;
        input line $varying10000. reclen ;

/* THE CALL PRXSUBSTR ROUTINE SEARCHES THE VARIABLE SOURCE WITH THE
   PATTERN FROM PRXPARSE, RETURNS THE POSITION OF THE START OF THE
   STRING, AND OPTIONALLY RETURNS THE LENGTH OF THE STRING THAT IS
   MATCHED. */

          call prxsubstr(perlrx, line, position, length);

/*      POSITION IS A NUMERIC VALUE THAT SPECIFIES THE POSITION IN SOURCE WHERE THE
        PATTERN BEGINS. IF NO MATCH IS FOUND, CALL PRXSUBSTR RETURNS ZERO.*/

        if position ^= 0 then
                match = substr(line, position, length);
         end ;/*not lastobs*/

        bat_file   = substr(file_ftp_log,1,index(file_ftp_log,".LOG"))||"BAT";";

        sas_x_bat_file =  "X "'|| bat_file;

        /* EXTRACT FILES HAVING NO MATCH */

        if match ^= ‘’  then  delete;
        run;
```

Step 6: Send Email and Resubmit FTP Scripts for Failed FTP Transmissions

At this point, the information about FTP transmissions which were not transmitted properly will be contained in dataset dsout. Furthermore, a data step will be used to determine total number of observations in dataset dsout. If there are any observations in dataset dsout, the SAS® process will write out failed FTP transmission BAT files into a SAS® output file for later execution. If there are no observations, then SAS® program will be output which will have only SAS® option statements. SYSPARM will allow information to be passed from the operating environment to SAS® program steps. DLY and REQ parameters will be defined in their corresponding BAT files.

Note: In this example, consider that sample_job_sasA_ftp.log is missing search text ‘ftp\>\ bye|bye|226 transfer complete|226 Binary Transfer complete‘

```
Data _null_;
        If 0 then set dsout nobs=count;
        Call symputx('dsout_obs',Count);
STOP;
RUN;

Data _null_;
set dsout;
        mv="bat_file"||(left(_n_);
        call symputx(mv, file_ftp_log);
run;

%if &sysparm=DLY %then %do;

%if &dsout_obs > 0 %then %do;

filename mail email ‘sac2001@gmail.com’;
data _null_;
        file mail
        subject= “SAS FTP Transmission Failed during production run”
        to=(‘sac2001@gmail.com’);
        put " OPERATIONS ";
        put;
        put "  Please note the following ";
        put;
        put "  The corresponding BAT/CMD files for the following LOG file(s) have been submitted for re-execution";
```

```
            put "  -After 15 minutes from receiving this email, please submit the following";
            put"  -Bat file chkftpreq.bat to determine if any are still in error";
            put ;

            %do  i=1 %to &dsout_obs;
            put "&&bat_file&i";
            %end;

Run;

            data _null_;
                  set dsout(keep=sas_x_bat_file);
                  file "..\project1\sasjobs\rerunftp.sas" print pagesize=1000 header=h notitles ;
                  put  @5  sas_x_bat_file ;
            return;

            H:
                  put  "options  noxwait noxsync;" //;
                  put;
                  return;
run;
```

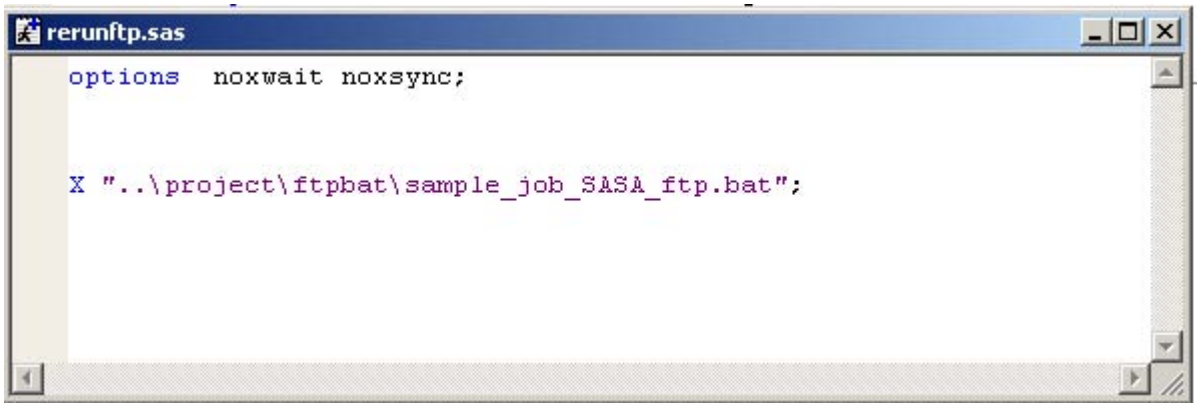

Figure 11: Showing the output of SAS® job 'rerunftp.sas' in SASJOBS directory when the SAS® process encounters a failed FTP transmission.

```
%end;/*&dsout_obs > 0*/
%else %do;
            data _null_;
            file "..\project1\sasjobs\rerunftp.sas " ;
            put  "options  noxwait noxsync;";
            run;
```

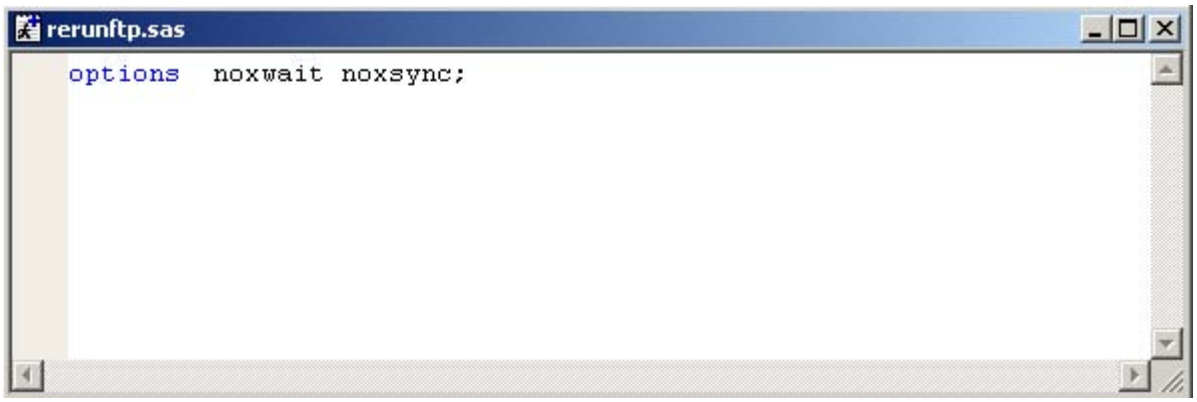

Figure 12: Showing the output of SAS® job 'rerunftp.sas' in SASJOBS directory when the SAS® process finds no failed FTP transmissions.

```
%end;
%end; /*%if &sysparm=DLY %then %do;*/
```

```
%if &sysparm=REQ %then %do;
        %if &dsout_obs > 0 %then %do;

        filename mail email 'sac2001@gmail.com';
        data _null_;
        file mail
        subject= "The following SAS FTP Transmission indicates FTP are still in error"
        to=('sac2001@gmail.com');
        put " OPERATIONS ";
        put;
        put "  This message is from scheduler running BAT file chkftpreq.bat   ";
        put;
        put "  The FTP logs are still showing errors   ";
        put "  The corresponding BAT/CMD files for the following LOG file(s) have been resubmitted and executed";
        put ;

            %do  i=1 %to &dsout_obs;
            put "&&bat_file&i";
            %end;

        Run;
        %end;/*&dsout_obs > 0*/
%end; /*%if &sysparm=DLY %then %do;*/
```

Step 7: Schedule the Jobs

The scheduler will submit the bat file rerunftp.bat for newly created SAS® job rerunftp.sas. The SAS® job rerunftp.sas should always be dependent on the SAS® process 'chkftp.sas' mentioned previously in this paper. There will be two BAT files for SAS® process CHKFTP.sas, one chkftpdly.bat to run the daily section of the SAS® utility using the sysparm "DLY" parameter, second chkftpreq.bat to run the request section of the SAS® utility using the sysparm "REQ" parameter. Sysparm is used to pass variables to the SAS® session from the command line.

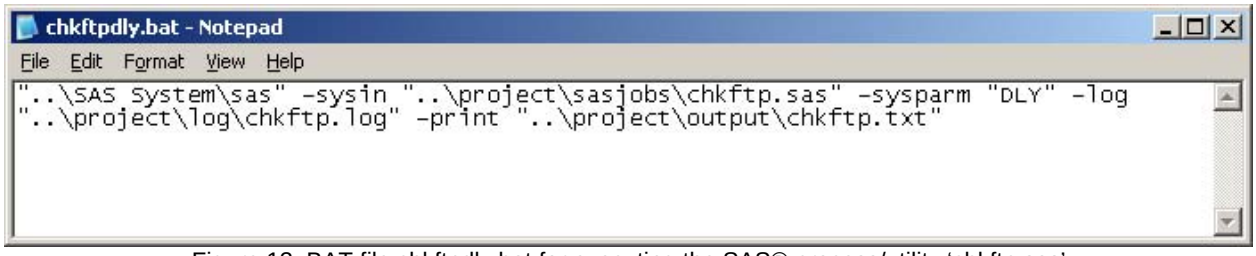

Figure 13: BAT file chkftpdly.bat for executing the SAS® process/utility 'chkftp.sas'

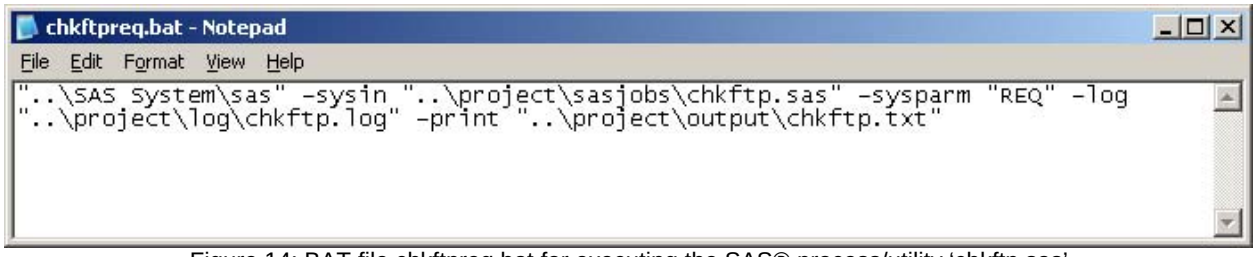

Figure 14: BAT file chkftpreq.bat for executing the SAS® process/utility 'chkftp.sas'

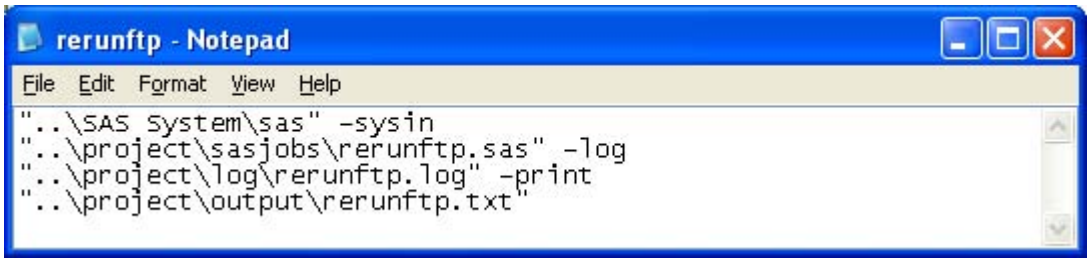

Figure 14: BAT file for executing a newly created job 'rerunftp.sas'

CONCLUSION

This paper has presented a process that can be used to improve application performance in automatically detecting, alerting and resubmitting failed FTP transfers by using the SAS® programming language. The specific code in this paper can be applied to similar SAS® programs running in a UNIX and mainframe environment with little modification. Following this process, standardized and well-implemented naming conventions of files and directories are important. The goal of this paper is to ease the burden of personnel running SAS® batch jobs in a production environment, and the responsibility of determining successful versus unsuccessful FTP transfers of SAS® program output to final destinations. Furthermore, the approach presented in this paper is a concept that can be expanded to automate a restart of failed scheduled jobs not only applied to SAS® programs and FTP transmissions, but also to other processes (e.g. SFTP, COBOL processes, PTP clients, etc.) where a log file is typically generated and which can be programmatically evaluated to determine if an automatic restart or resend is required.

Acknowledgements and Contact

The author wishes to thank Kevin Desforges of Trident Analytics, Inc. for his careful review of the paper. The author may be contacted at:

Salman Ali
Element Technologies Inc
15 Corporate Place South Suite 209
Piscataway, NJ 08854
Contact No: 646-744-4131
sac2001@gmail.com

References

SAS® Inc Functions and CALL Routines
Compiles a Perl regular expression (PRX) that can be used for pattern matching of a character value
http://post.queensu.ca:8080/SASDoc/getDoc/en/lrdict.hlp/a002295977.htm