Paper 017-2008

Quick Windows Batches to Control SAS®Programs Running Under Windows and UNIX

Stanislaw Furdal, Bristol Myers Squibb Co., Plainsboro, NJ

ABSTRACT

Very often programmers use Windows operating system and UNIX system on a server to submit their programs. For such programmers it is convenient sometimes (for example because of Windows Microsoft Office usage in ad-hoc reporting departments) to build a Windows batch job and control submitted programs in Windows environment. The paper presents how to build quickly Windows batches to execute programs under Windows and/or UNIX with the focus how to control execution errors in only Windows environment. The **SAS/CONNECT®** is applied to submit programs running under UNIX. Both an included code technique and an approach with **SYSTASK** will be presented. The advantage of using **SYSTASK** against the included code will be discussed. Some interesting facts about **SAS®** program errors will be presented. Also the differences between calling Windows commands from inside a program and from a batch file will be analyzed. In addition, invoking FTP procedures, email system, zipping software and calling other executables related to SAS programming also will be presented. All solutions will be analyzed by providing ready-to-apply code. Batch examples presented in this paper are related to **Windows XP** and **UNIX 64HP** operating systems. All SAS code was executed using the **SAS 9.1.3** version.

INTRODUCTION

It is obvious that building and running batch jobs can save a lot of time and resources. In the SAS environment there are typically two ways to build batch jobs. One way is by using a wrapper SAS program (Andrews, Cogswell 2005) which calls other SAS programs. The other way is by using a higher generation language like Pearl, Phyton, REXX (Furdal 2006), Ruby, UNIX script language, VBScript, WSH, Windows commands etc. having capability to invoke programs written in other languages. This paper brings to attention writing quickly batches using Windows operating system commands. Very often programmers use Windows and UNIX to submit their programs. Using both Windows and UNIX makes programming more flexible, extends capabilities of data processing. Sometimes for a programmer building a Windows batch job on a PC is more convenient than on a server.

If we have to submit sometimes dozens of programs in a batch it is very important to know at least a SAS error code after one program execution in order to decide to continue or stop next program executions. The paper will show how to build a Windows batch and fully control errors using Windows **ERRORLEVEL** system variable. The **ERRORLEVEL** variable can be used also to control return codes of programs executed under UNIX. Such batch jobs do not require any manual intervention and can be very easy scheduled to run automatically.

It is known that after every SAS program execution in a batch, the Windows operating system assigns a return code (integer number) to the Windows **ERRORLEVEL** system variable. But how exactly the Windows system assigns a SAS program return code to this Windows **ERRORLEVEL** it is not known. The SAS system after execution shows return codes in different macro variables as for example: **SYSCC, SYSERR, SYSINFO, SYSRC, SYSFILRC**, **SYSLIBRC** and **SYSLCKRC**. However according with the SAS documentation only the **SYSCC** can be treated as a job program condition code, other macro variables are related only to some parts of SAS code, and simulation studies showed that probably only **SYSCC** decides about the **ERRORLEVEL** value. **SYSCC** values are host dependent. Under Windows XP and UNIX 64HP the **SYSCC**=0 means normal, good execution, 4 – means execution with warnings, **SYSCC** > 4 means errors happened during execution. After calling a SAS program from a batch and run under Windows, the Windows system assigns a return code to the Windows system variable **ERRORLEVEL** and the **ERRORLEVEL** = 0 means good execution, **ERRORLEVEL** = 1 means execution with warnings, **ERRORLEVEL** >= 2 means execution with errors.

CALLING SAS PROGRAM TO RUN UNDER WINDOWS

Submitting a SAS program from a Windows script has two phases, one is to call first a SAS exe program to invoke SAS system and the next part is to call the SAS program with options. Here is the example of the Windows batch statement to execute the SAS program called MyPgm.sas:

```
"C:\Program Files\Sas\Sas 9.1\sas.exe" -SYSIN C:\MyPgm.sas  -ICON  -NOSPLASH
     -LOG     C:\Logs\MyPgm.log        -PRINT  C:\Outputs\MyPgm.lst        -SYSPARM  "03182008"
```

All the above pieces of the statement to invoke the MyPgm.sas have to be written in one line of the batch file. The **ICON** and **NOSPLASH** SAS options disable displaying the pop-up panels with info about running program and a startup panel. The **LOG** and **PRINT** allow to locate the log and listing files, the **SYSPARM** allows to pass parameters to the executed program. The full path for the SAS system sas.exe can be omitted if we define it in the **PATH** Windows system variable. The extension exe can also be omitted from an executable file name if exe is defined in the Windows **PATHEXT** environment variable. Usually **.com, .exe, .bat, .cmd** are defaults defined in the **PATHEXT**. As it was mentioned in the introduction, after calling a SAS program from a batch using the above code and the

program execution, the Windows system assigns a return code to the Windows system variable **ERRORLEVEL**. The **ERRORLEVEL** = 0 means good execution, **ERRORLEVEL** = 1 means execution with warnings, **ERRORLEVEL** = 2 means execution with errors.

CALLING SAS PROGRAM FROM WINDOWS TO RUN UNDER UNIX

We can do that using **SAS/CONNECT**. However we want to submit a UNIX SAS program from Windows environment and we would like to get, for control purposes, at least a return code after this execution back in the Windows batch file. There are following two ways to submit and execute a SAS program under UNIX using **SAS/CONNECT** and get a return code back in Windows:

a) Submitting a program using INCLUDE

A simple way to submit a Unix SAS program is just to include its code (we can use also %**INCLUDE**) between **RSUBMIT** and **ENDRSUBMIT** in the wrapper Windows SAS program and use **%SYSRPUT** macro statement to get a UNIX SAS return code, from **SYSCC**, back in Windows. The Windows SAS using **SAS/CONNET** invokes SAS system on UNIX defined in the **filename rlink** script and executes the included SAS statements. Here is a simple example how to execute SAS statements under UNIX and get the UNIX SAS return code back in Windows:

```
RSUBMIT;
   /* SAS statements */
  %SYSRPUT Usyscc=&SYSCC;   /* SYSCC on UNIX */
ENDRSUBMIT;
signoff;
%let SYSCC=&Usyscc;                /* SYSCC on Windows*/
```

Here is another approach to submit the MyPgm.sas using **%INCLUDE** and get back UNIX **SYSCC** in the Windows **SYSCC**:

```
RSUBMIT;
  PROC UPLOAD INFILE=" C:\MyPgm.sas"
                OUTFILE="/home/mypgms/MyPgm.sas"; RUN;
  %INCLUDE "/home/mypgms/MyPgm.sas";
  %SYSRPUT Usyscc=&SYSCC;     /* SYSCC on UNIX */
ENDRSUBMIT;
signoff;
%let SYSCC=&Usyscc;                /* SYSCC on Windows*/
```

The last statement in both cases **%let SYSCC**=&Usyscc assigns the UNIX **SYSCC** value to the Windows **SYSCC**. We can update the Windows **SYSCC** because **SYSCC** is a read / write variable and can be reset. However here is some problem using the above include approaches. There are some errors (like missing apostrophes) which can cause that the following **%SYSRPUT** statement will not be executed and the information about this error will not be carried out by **%SYSRPUT** to the Windows environment. Testing revealed that besides of this problem mostly the Windows system reacts well to this above code and sets the **ERRORLEVEL** respectively to the Windows **SYSCC** value. If **SYSCC** = 4 (warnings) then **ERRORLEVEL** = 1, if **SYSCC** > 4 (errors) then **ERRORLEVEL**=2. The log file in Windows contains also the UNIX log messages after executing the included statements.

b) Submitting a program using SYSTASK

The other way (which seems to be better one, more reliable to get a return code), is to use the **SYSTASK COMMAND** statement to call and execute a SAS program under UNIX. Instead of including the code between **RSUBMIT** and **ENDRSUBMIT** we can use **PROC UPLOAD** to upload the program from Windows to UNIX, then to call and execute this program under UNIX using the **SYSTASK** command. The **SYSTASK** status value reflects the correctness of the program execution under UNIX in a similar way to **ERRORLEVE**L. It is equal 0 if the program ran successfully, is equal 1 when the run was with warnings and is equal 2 when there were serious errors. Here is an example how to submit MyPgm.sas from the Windows SAS wrapper program and let it run under UNIX:

```
RSUBMIT;
  PROC UPLOAD INFILE=" C:\MyPgm.sas"  OUTFILE="/home/mypgms/MyPgm.sas"; RUN;
  SYSTASK COMMAND "sas  MyPgm.sas"  TASKNAME=MyPgm  WAIT  STATUS=Uretcode;
 %SYSRPUT Wretcode=&Uretcode;
   PROC DOWNLOAD infile="/home/mypgms/MyPgm.log"  outfile="C:\MyPgm.log"; RUN;
   x 'rm MyPgm.sas  MyPgm.log';
ENDRSUBMIT;
```

In the above example the **PROC UPLOAD** uploads the MyPgm.sas program from a Windows folder to a UNIX folder. Instead of **PROC UPLOAD** we could use **filename FTP** and transfer the program to the UNIX folder. The **SYSTASK** command first calls SAS system on UNIX (**sas** in the command) and then executes the MyPgm.sas giving

a return code in the Uretcode macro variable. The **SYSRPUT** assigns the Uretcode value to the Windows macro variable Wretcode. The **PROC DOWNLOAD** in this example is used to download the program UNIX log file to Windows. The UNIX SAS **X** command executes UNIX **rm** command to remove MyPgm.sas and MyPgm.log from UNIX. And this is some disadvantage of this **SYSTASK** approach. After execution of MyPgm by using the **SYSTASK** command the log file (also the sas pgm code and the lst file) will be only under UNIX and we have to download it to Windows if we want to have all log information under Windows and we have to delete it from UNIX to avoid keeping them both on Windows and UNIX. On the other side, after using the Include approach, the log messages are included in the Windows wrapper program log file. Catching and carrying UNIX **SYSCC** to Windows just after **SYSTASK** command is not a good idea in this case, because this UNIX **SYSCC** reflects only the correctness of **SYSTASK** command (syntax error or lack of resources), not the correctness of the MyPgm.sas execution. So the UNIX program can execute with errors but the UNIX **SYSCC** just after **SYSTASK** can be 0. The UNIX program invoked by **SYSTASK** is treated like a separate entity to which this **SYSCC** does not relate to. We could catch the Unix **SYSCC** inside the MyPgm code. That would be the same idea like with the Include code where sometimes **SYSRPUT** inside the MyPgm would not work.

After getting back the UNIX return code in the Windows macro variable Wretcode, now we can write a simple macro at least to abend the wrapper Windows SAS program if &Wretcode > 1, which will give the **ERRORLEVEL** value greater than 2 (will be 5 under Windows XP):

```
%macro checkRC;
  %if &Wretcode > 1 %then %do;
      data _null_;
        Abort Abend;
      run;
  %end;
%mend;
%checkRC;
```

The simulation study revealed that the **SYSTASK** statement brings a reliable return code about correctness of a UNIX program execution. Any SAS error or warning is reflected in the **STATUS** macro variable. Additionally, using the **SYSTASK** command, we can submit different tasks from the wrapper Windows SAS program and we can run them asynchronously.

CONTROLLING EXECUTION ERRORS

As it was mentioned before, after each executed program or command in a Windows batch file, a certain return code number is assigned automatically to the **ERRORLEVEL** Windows system variable.

Here is the list of possible **ERRORLEVEL** values and their meanings related to a SAS program execution:

0 - Program ended normally, no warnings, no errors
1 - Program ended with warnings
2 - Program ended with error statements, no abort
3-5 - Program terminated by **ABORT** statements
6 - SAS system internal errors

By using **IF** and **GOTO** statements in a batch against **ERRORLEVEL** values, we can make simple decisions to execute or not other following programs and even send emails about program executions. In some cases **ERRORLEVEL** can be negative, as for instance after a manual termination of the SAS session.

Here is a simple example of a batch to execute two SAS programs (Pgm1 - Windows program, Pgm2Unix - Windows wrapper program containing **SYSTASK** to call a UNIX program). By using **ECHO**, **IF** and **GOTO** Windows statements in the batch we can display a return code value, and if **ERRORLEVEL** is greater or equal 2 to redirect execution to the statements following the label End

```
@ECHO OFF
ECHO ******* Start *******

"C:\Program Files\Sas\Sas 9.1\sas.exe" -SYSIN Pgm1.sas -ICON -NOSPLASH -SYSPARM "01312006"
ECHO Pgm1       rc= %ERRORLEVEL%
IF %ERRORLEVEL% GEQ 2 GOTO End

"C:\Program Files\Sas\Sas 9.1\sas.exe" -SYSIN Pgm2Unix.sas -ICON -NOSPLASH
ECHO Pgm5Unix rc= %ERRORLEVEL%

:End
ECHO ******* End *******
```

As it was mentioned in the previous paragraph using **SAS/CONNECT**, **SYSTASK** to run UNIX program and get a return code from **SYSTASK** status in Windows is better than using **SAS/CONNECT**, **INCLUDE**. There are some errors (like missing apostrophes) that after using **SAS/CONNECT**, **INCLUDE** and **%SYSRPUT**, the **ERRORLEVE**L in Windows will be 0.

SAVING LOG EXECUTION MESSAGES

After a batch execution usually we have two kinds of log messages about the Windows batch processing. One type of the log messages is from the SAS system and the other one is from the Windows system. The log file from the SAS system we get automatically when a SAS program is executed. We can set the location of this file using -**LOG** option in a batch call statement. We can also use **PROC PRINTTO log=** inside the program it takes priority over –**LOG**. If we omit –**LOG** option and **PROC PRINTTO** the SAS system creates a log file in the same folder where the program is located. As it was mentioned before when we use **SYSTASK** the log file from UNIX SAS program can be downloaded to Windows using **PROC DOWNLOAD**.

Log messages of the Windows system display automatically on the Windows cmd.exe screen when a batch is running. We can save them manually in a text file. We can save them also automatically if we call our batch from a wrapper batch using the statement as in the following example where we use the "**>**" character to save the batch log messages in the MyBatch.log file:

CALL MyBatch**.bat > ** C:\MyBatch**.log**

This log file will not have log messages from SAS programs. It will have only messages from the Windows system, related to the executed Windows commands in the bat file. We can use "**>>**" to cumulate (append) the log messages from consecutive executions.

CALLING OTHER SOFTWARE (ZIP, FTP, email) FROM BATCH

a) Zipping or unzipping files

The software to zip/unzip files is commonly used. The Microsoft **WinZip** is probably the most often used by computer users. To zip or unzip files very often we manually use an online version of zip software with some user friendly screens but for zipping / unzippping files from a batch file without any manual intervention, we have to have a command-line executable file. The **WinZip Pro** has it, but also we can use some free-ware software available from Internet. The available zipping software is mostly based on older **PKZIP** and **PKUNZIP**. Let's say that we have a C:**zip.exe** file to zip files from a batch, then the typically batch statements to call it and execute can be like this:

"C:**zip.exe**" Test.**zip** TestFile1.txt TestFile2.txt

In this example two files TestFile1.txt and TestFile2.txt will be zipped into the Test.**zip** file. Attention! – The **ERRORLEVEL** after zipping from a batch reflects only the correctness of the zip.exe statement like syntax and correct location of files in this statement. It doesn't reflect the quality of zipping.

b) Calling File Transfer Protocol procedure

The popular **FTP** procedure is usually executed in two phases, first we call an **FTP** server using some command (usually the same name **FTP** is used) and then executing some script with **FTP** statements. Here is an example how to call the **File Transfer Protocol** procedure from a batch using **FTP** command and execute **FTP** statements in the myftpscript.txt file.

FTP -i -d -s:myftpscript.txt TCP/IPremoteAddress > C:\myftpscript.log

The **–i, -d, -s** are the **FTP** command options (**i** – turns off interactive prompting, **d** – displays **FTP** commands passed between client and server, **s** – specifies a script file), myftpscript.log is the log file in which we will get the messages after execution of the **FTP** statement in the script file. The myftpscript.txt (can have also **scr** extension) contains usually an **FTP** user id and password and following **FTP** procedure statements like **lcd, cd, put, get, status**, **ls** and so on. There is a way to write both the **FTP** command and the script statements in one batch file.

However there is one problem with executing the **FTP** from a batch file. The Windows system variable **ERRORLEVEL** after **FTP** execution reflects only the correctness of the **FTP** command execution (syntax, access to remote system, etc.), not the correctness of the executed **FTP** statements in the script. The correctness of the script statement executions we can find out only in the log file. The **ERRORLEVEL** does not reflect this and very often **ERRORLEVEL** is 0 even when a log file shows that sending or receiving files was with errors. We can solve this problem applying the **FINDSTR** Windows command to the saved log file searching for error strings. Attention, if the error strings (error key words or **FTP** error message codes) are found then the **ERRORLEVEL** = 0, if strings are not found the **ERRORLEVEL** = 1. Here is the example of the **FINDSTR** command call to check if error words are in the log file:

FINDSTR /i "failed denied error found could invalid" C:\myftpscript.log
IF %**ERRORLEVEL**% **EQU** 0 **ECHO** Found error words.

c) Sending emails

We can send emails from a batch at least to inform us about correctness of a batch execution. We can use the SAS system to do that. Here is an example of the statements in a batch file to run two SAS programs to send emails about a failure or a successful completion of a SAS program:

```
IF %ERRORLEVEL% GEQ 2 (
"C:\Program Files\Sas\Sas 9.1\sas.exe" -SYSIN C:\MyFailureEmail.sas    -ICON -NOSPLASH -EMAILSYS SMTP -
EMAILHOST mailhost.xms.com -EMAILPORT 25 -LOG C:\MyFailureEmail.log
) ELSE (
"C:\Program Files\Sas\Sas 9.1\sas.exe" -SYSIN C:\MyCompletionEmail.sas -ICON -NOSPLASH -EMAILSYS SMTP
-EMAILHOST mailhost.xms.com -EMAILPORT 25 -LOG C:\MyCompletionEmail.log
)
```

These statements, with the **EMAILSYS, EMAILHOST, EMAILPORT** SAS options, allow sending emails even when the email system is turned off. The simple MyCompletionEmail.sas program to send email about successful completion could be like this:

```
filename notify email stan.furdal@bms.com  subject = "Job Completed";
data _null_;
  file notify;
  put "The reports have been created successfully."; run;
```

CALLING WINDOWS COMMANDS FROM BATCH VS FROM SAS PROGRAM

As we know the SAS system allows to execute Windows commands from inside a SAS program using **X** statement, **CALL SYSTEM** or %**SYSEXEC**. Apparently, if the logic of data processing allows, calling them from a batch file seems to be a better solution. To call Windows commands from a SAS program, a SAS programmer has to find out good settings of SAS options **xwait, xsync** and also a respective **sleep** statement to get good execution of the Windows commands. Sometimes even simple delete and copy of multiple files using **X** command can be incorrect and the SAS program does not give any error information about it. If we submit Windows commands from a Windows batch file only Windows operation system is responsible for that and **ERRORLEVEL** will tell us correctly if the execution of commands was good or not.

CONCLUSIONS

1. It is easy to build a Windows batch to execute SAS programs under Windows and/or UNIX and fully control execution errors in only Windows environment. UNIX system can be used only as a program executor. We can keep all source code and get all information about program executions in Windows environment.
2. SAS **SYSTASK** command and **%SYSRPUT** statement allow submitting successfully a program under UNIX and getting a correct execution return code back in the Windows SAS system. It seems more reliable than applying **%INCLUDE**.
3. The Windows system **ERRORLEVEL** environment variable can be used to control program executions under Windows and UNIX.
4. Executing Windows commands from a batch file is more reliable than executing them from inside a SAS program.
5. Windows batch jobs can run other executables related to SAS programming as for example to zip, FTP, send emails, etc.
6. Windows batches provide clear structure of executing programs and system commands.

REFERENCES

1. Andrews Rick, Dixon Sherry, (2005, SUGI 30): “Performance Monitoring for SAS Programs on Windows XP”.
2. Cogswell Denis L. (2005, SUGI 30): “More than Batch – A Production SAS Framework”.
3. Furdal Stanislaw (2006, SUGI 31): “Batch Job to Create and Send Mass Excel Reports Using REXX, SAS®, HTML (XML)”, Paper 022-31.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please feel free to contact the author at:

Stanislaw Furdal
Bristol Myers Squibb Co.
777 Scudders Mill Road
Plainsboro, NJ 08543
E-mail: stan.furdal@bms.com