Paper CC135

# Interacting with SAS® using Windows PowerShell ISE

Mayank Nautiyal
Epsilon

## ABSTRACT

The most conventional method of using SAS on a Windows environment is via a Graphic User Interface (GUI) application. There are numerous SAS users who have a UNIX background and can definitely take advantage of the Windows PowerShell Integrated Scripting Environment (ISE) to gain job efficiency in Windows. The Windows PowerShell ISE is a host application for Windows PowerShell. One can run commands, write, test, and debug scripts in a single Windows-based graphic user interface with multiline editing. This paper will demonstrate how frequently used SAS procedures can be scripted and submitted at the PowerShell command prompt. Job scheduling and submission for batch processing will also be illustrated.

## INTRODUCTION

Often, we write SAS programs which are repetitive and many a times need simple tweaks like in data set or variable names. Scripting can eliminate the need to write these repetitive SAS programs and replace them with one word user-defined aliases. This not only saves time in writing the same program multiple times but also increases efficiency by running the job in background. Such tasks have been implemented on UNIX platform but not much has been explored on Windows platform. Like UNIX, Windows PowerShell offers the similar power of scripting which can be utilized to run SAS programs. A programmer with minimal scripting and BASE SAS background could take advantage of these techniques to simplify repetitive tasks. This paper will present methods which can eliminate the need of writing repetitious SAS programs and how SAS jobs can be scheduled and submitted for batch processing in Windows environment.

## ABOUT WINDOWS POWERSHELL

Windows PowerShell is a command shell by Microsoft which provides command prompt and scripting environment at the same time. With PowerShell one can not only enter commands, but can write scripts from the PowerShell command line. A user can write, test, and debug scripts in a single Windows-based graphic user interface with multiline editing. It also provides easy features of copy, paste, editing, syntax highlighting etc. in one interface. PowerShell commands are referred to as **Cmdlets**. PowerShell gives users the privilege to create their own Cmdlets, and assign aliases to the exiting or user defined Cmdlets.Table1 shows Powershell equivalents of UNIX commands used in this paper:

| UNIX Command | PowerShell Command |
|---|---|
| Read | read-host |
| Print | write-output |
| Date | get-date |
| More | get-content |
| Export | $env |
| Alias | set-alias |

**Table 1. Some Powershell equivalents of UNIX commands used in this paper**

## USING POWERSHELL SCRIPT TO WRITE, RUN SAS PROGRAM AND EXPORT THE OUTPUT ON POWERSHELL SCREEN

Multiple times we write SAS programs which are repetitive in nature and the only change needed to get the desired output is either change in name of data set or variables. `PROC PRINT`, `PROC CONTENTS`, `PROC FREQ`, AND `PROC DATASETS` are few such examples which fall in this category. Following PowerShell Script (saved as content.ps1) and SAS program (saved as content.sas) can be used to check the content of a SAS data set with minimal effort:

```
#ASSIGNING SAS EXECUTION FILE TO VARIABLE SASCMD
$SASCMD="C:\Program Files\SASHome_94\SASFoundation\9.4\sas.exe"

#ASSIGNING SAS CONFIGURATION FILE TO VARIABLE SASCFG
$SASCFG="C:\Program Files\SASHome_94\SASFoundation\9.4\nls\en\sasv9.cfg"

#ASSIGNING SAS PROGRAM CONTENT.SAS TO VARIABLE SASPRGM
$SASPRGM=" C:\Users\MNautiyal\Documents\sas_easy\pgm\content.sas"

#ASSIGNING SAS OUTPUT CONTENT.LST TO VARIABLE SASPRGM
$SASLSTS=" C:\Users\MNautiyal\Documents\sas_easy\pgm\content.lst"

#ASSIGNING SAS LOG CONTENT.LOG TO VARIABLE SASPRGM
$SASLOG=" C:\Users\MNautiyal\Documents\sas_easy\pgm\content.log"

#VARIABLE DSNAME CAPTURES THE FIRST INPUT PARAMETER (i.e. NAME OF DATA SET) AT
COMMAND LINE
$DSNAME=$args[0]

#VARIABLE DSNAME IS DECLARED AS AN ENVIRONMENT VARIABLE SO THAT IT CAN BE EXPORTED
TO THE SAS PROGRAM
$ENV:DSNAME=$DSNAME

#CMD COMMAND CALLS FOR SAS PROGRAM CONTENT.SAS
cmd /c " ""$SASCMD"" -nosplash -config ""$SASCFG"" -sysin ""$SASPRGM"" -log
""$SASLOG"" -print ""$SASLST"""

#GET-CONTENT COMMAND PRINTS THE LOG and LST ON SCREEN
GET-CONTENT -Encoding UTF8 $SASLOG
GET-CONTENT -Encoding UTF8 $SASLST
```

Apart from comments, the basic functionality of this script is that it calls program content.sas and passes one parameter DSNAME (data set name) to it. Once the program completes execution, the script prints out content.log and content.lst on the screen. The SAS program content.sas shown below uses `PROC CONTENTS` procedure and `%SYSGET` macro function to capture the argument passed at the command prompt.

```
LIBNAME OUT '.' ;
OPTIONS MLOGIC SYMBOLGEN ;
PROC CONTENTS DATA = OUT.%SYSGET(DSNAME) ;
RUN ;
```

By declaring OUT library at the location where data set is present eliminates the need for SAS program “content.sas” to be available in the same directory as the data set. This script can be assigned an alias and executed from any directory.

For example if the script is saved in “`C:\Users\MNautiyal\Documents\sas_easy\scripts`” as `content.ps1`, it can be assigned an alias (`content` in this case) by using `set-alias` command at the command prompt as follows:

```
> set-alias content "C:\Users\Mnautiyal\Documents\sas_easy\scripts\content.ps1”
```

In order to use this alias, submit following command at the command prompt:

```
> content dsname
```

Submitting the above command will print log and output of the content.sas program on PowerShell screen. It is important to note here that the alias should be executed from the directory containing data set of interest. Table OUTPUT1 shows output of the above submission.

Few more such examples of executing pre-defined SAS programs are shown in Appendix A and B. Appendix A demonstrates how we can print desired number of observations in a data set by submitting `alias print`. While, Appendix B demonstrates, how we can check frequency of a variable or cross-tabulation between variables by submitting `alias freq` at the PowerShell terminal.

Another similar application of PowerShell scripting is, giving the user power to submit a code at the command prompt and obtaining a quick result at the PowerShell terminal itself. The following script demonstrates how we can achieve this task:

```
 #ASSIGNING SAS EXECUTION FILE TO VARIABLE SASCMD
$SASCMD="C:\Program Files\SASHome_94\SASFoundation\9.4\sas.exe"

#ASSIGNING SAS CONFIGURATION FILE TO VARIABLE SASCFG
$SASCFG="C:\Program Files\SASHome_94\SASFoundation\9.4\nls\en\sasv9.cfg"

#ASSIGNING SAS PROGRAM QUERY.SAS TO VARIABLE SASPRGM
$SASPRGM="C:\Users\MNautiyal\Documents\sas_easy\pgm\query.sas"

#ASSIGNING SAS OUTPUT QUERY.LST TO VARIABLE SASPRGM
$SASLSTS=" C:\Users\MNautiyal\Documents\sas_easy\pgm\query.lst"

#ASSIGNING SAS LOG QUERY.LOG TO VARIABLE SASPRGM
$SASLOG=" C:\Users\MNautiyal\Documents\sas_easy\pgm\query.log"


#VARIABLE QUERY READS THE INPUT PARAMETER (EXAMPLE: A SMALL PIECE OF SAS CODE) AT
COMMAND LINE
$QUERY=READ-HOST

#WRITE LIBNAME STATEMENT IN QUERY.SAS PROGRAM
WRITE-OUTPUT "LIBNAME OUT '.' ;" >
C:\Users\MNautiyal\Documents\sas_easy\pgm\query.sas

#APPEND THE ARGUMENT $QUERY TO QUERY.SAS PROGRAM
WRITE-OUTPUT $QUERY >> C:\Users\MNautiyal\Documents\sas_easy\pgm\query.sas

#CMD COMMAND CALLS FOR SAS PROGRAM CONTENT.SAS
cmd /c " ""$SASCMD"" -nosplash -config ""$SASCFG"" -sysin ""$SASPRGM"" -log
""$SASLOG"" -print ""$SASLST"""

#GET-CONTENT COMMAND PRINTS THE LOG and LST ON SCREEN
GET-CONTENT -Encoding UTF8 $SASLOG
GET-CONTENT -Encoding UTF8 $SASLST
```

This script creates a SAS program query.sas by outputting the `libname` statement to a blank SAS program and then appending the SAS code submitted at the command prompt to it. We can assign an alias to the script query.ps1 as follows:

```
> set-alias query "C:\Users\Mnautiyal\Documents\sas_easy\scripts\query.ps1”
```

When submitted at the command prompt, it waits for the user to enter a SAS code. The SAS code should be entered in one line (i.e. without inserting an <enter> in the code). In order to get a better picture of this application we can examine a sample data set shoes.sas7bdat provided in sashelp library. Below is the ouput of submitting `content shoes` at the command prompt. It shows PROC CONTENTS (log has been suppressed here) of the shoes data set.

```
                          The CONTENTS Procedure

Data Set Name        OUT.SHOES                     Observations          395
Member Type          DATA                          Variables             7
Engine               V9                            Indexes               0
Created              06/19/2013 23:29:58           Observation Length    88
Last Modified        06/19/2013 23:29:58           Deleted Observations  0
Protection                                         Compressed            NO
Data Set Type                                      Sorted                NO
Label                Fictitious Shoe Company Data
Data Representation  WINDOWS_64
```

```
Encoding             us-ascii  ASCII (ANSI)


Engine/Host Dependent Information

Data Set Page Size          65536
Number of Data Set Pages    1
First Data Page             1
Max Obs per Page            743
Obs in First Data Page      395
Number of Data Set Repairs  0
ExtendObsCounter            YES
Filename                    C:\Users\MNautiyal\Documents\data\shoes.sas7bdat
Release Created             9.0401B0
Host Created                X64_7PRO


Alphabetic List of Variables and Attributes

#    Variable      Type    Len    Format       Informat     Label

6    Inventory     Num       8    DOLLAR12.    DOLLAR12.    Total Inventory
2    Product       Char     14
1    Region        Char     25
7    Returns       Num       8    DOLLAR12.    DOLLAR12.    Total Returns
5    Sales         Num       8    DOLLAR12.    DOLLAR12.    Total Sales
4    Stores        Num       8                              Number of Stores
3    Subsidiary    Char     12
```

**Output 1. Contents of shoes.sas7bdat data set. Log has been suppressed.**

If we want to look at sum of sales by region we can submit our query at the command prompt as follows:

```
   > query
PROC SQL ; SELECT REGION , SUM(SALES) AS SUM_SALES FROM OUT.SHOES FORMAT COMMA8.0 GROUP
BY REGION ; QUIT ;
```

```
NOTE: Libref OUT was successfully assigned as follows:
      Engine:        V9
      Physical Name: C:\Users\MNautiyal\Documents\data
2          PROC SQL ;
2        !            SELECT REGION, SUM(SALES) AS SUM_SALES FORMAT COMMA8.0 FROM
OUT.SHOES GROUP BY REGION ;
2        !
QUIT ;
NOTE: The PROCEDURE SQL printed page 1.
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.04 seconds
      cpu time            0.04 seconds



NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
NOTE: The SAS System used:


                          The SAS System            21:09 Sunday, August 3, 2014   1

               Region                      SUM_SALES
               ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ
               Africa                        2342588
               Asia                           460231
               Canada                        4255712
```

```
Central America/Caribbean    3657753
Eastern Europe               2394940
Middle East                  5631779
Pacific                      2296794
South America                2434783
United States                5503986
Western Europe               4873000
```

**Output 2. Output of the proc sql query submitted at the command prompt**

You might notice strange character "ƒ" in the output which can be avoided by enabling `formchar` options in the SAS program. It is also important to keep in mind that user-defined aliases are lost when you open a new PowerShell window, so in order to preserve these aliases one should use `EXPORT-ALIAS` and `IMPORT-ALIAS` command to make the aliases permanent.

## JOB SCHEDULING AND BATCH PROCESSING OF SAS JOBS IN WINDOWS

Automating execution of SAS programs allows the user to run jobs unattended. PowerShell scripts can be used to perform job scheduling and batch processing in Windows. Such processing runs SAS programs in background and writes the results to log and lst files which can be viewed later. Let's look at an example of a PowerShell script which can run SAS jobs in batch mode.

```
 #ASSIGNING SAS EXECUTION FILE TO VARIABLE SASCMD
$SASCMD="C:\Program Files\SASHome_94\SASFoundation\9.4\sas.exe"

#ASSIGNING SAS CONFIGURATION FILE TO VARIABLE SASCFG
$SASCFG="C:\Program Files\SASHome_94\SASFoundation\9.4\nls\en\sasv9.cfg"

FUCNTION BATCH
{
#ASSIGNING SAS PROGRAM TO VARIABLE SASPRGM
$SASPRGM="C:\Users\MNautiyal\Documents\sas_easy\pgm\$args.sas"

#ASSIGNING SAS OUTPUT QUERY.LST TO VARIABLE SASPRGM
$SASLSTS=" C:\Users\MNautiyal\Documents\sas_easy\pgm\$args.lst"

#ASSIGNING SAS LOG QUERY.LOG TO VARIABLE SASPRGM
$SASLOG=" C:\Users\MNautiyal\Documents\sas_easy\pgm\$args.log"

#CMD COMMAND CALLS FOR SAS PROGRAM CONTENT.SAS
cmd /c " ""$SASCMD"" -nosplash -config ""$SASCFG"" -sysin ""$SASPRGM"" -log
""$SASLOG"" -print ""$SASLST"""

#GET-CONTENT COMMAND PRINTS THE LOG and LST ON SCREEN
GET-CONTENT -Encoding UTF8 $SASLOG
GET-CONTENT -Encoding UTF8 $SASLST

}

BATCH sasjob1
BATCH sasjob2
BATCH sasjob3
```

The above script (saved as sasbatchjobs.ps1) has pretty much similar structure as the scripts discussed in previous section. Only variation is that SAS program(s) to be called are defined in a function called BATCH. This function accepts only one argument i.e. name of SAS program to be executed. So, if there are three SAS jobs (sasjob1.sas, sasjob2.sas and sasjob3.sas) which are to be executed sequentially, they can be passed as an argument to the function BATCH in order of execution.

Now, if we want to schedule execution of these SAS jobs at a particular time of a day or we want them to be scheduled to run at regular intervals, then we can schedule execution of sasbatchjobs.ps1 script accordingly. The following script (saved as scheduler.ps1) requires the user to modify the initial 9 variables, which are used to schedule a job at desired time and interval. Illustrative comments have been incorporated to describe each step.

```
$ONETIMEJOB="Y"       #1) IS THIS A ONE TIME JOB ?
$RUNJOBDAILY="N"      #2) IS THIS A DAILY JOB ?
$RUNJOBWEEKLY="N"     #3) IS THIS A WEEKLY JOB ?
$RUNJOBHOURLY="N"     #4) IS THIS AN HOURLY JOB ?
$RUNJOBMONTHLY="N"    #5) IS THIS A MONTHLY JOB ?
$BEGINDAY="02"        #6) DAY OF JOB ?
$BEGINMONTH="08"      #7) MONTH OF JOB ?
$BEGINYEAR="2014"     #8) YEAR OF JOB ?
$BEGINTIME="15:03"    #9) TIME OF JOB IN 24-HOUR FORMAT ?

#CONCATENATE MONTH, DAY, YEAR AND BEGINTIME TO CREATE A FORMAT SIMILAR TO GET-DATE
$BEGIN="$BEGINMONTH/$BEGINDAY/$BEGINYEAR $BEGINTIME"

#CREATE A VARIABLE TO PROCESS ONETIME JOBS
$ONETIME=1

#CREATE A FUNCTION WAIT_TIME WHICH CALCULATES WAIT TIME BETWEEN REPETITIVE JOBS
FUNCTION WAIT_TIME
{
#READ FLAG VALUE TO MAKE SURE SCHEDULAR EXECUTES NEXT RUN
$FLAG=(GET-CONTENT C:\Users\Mnautiyal\Documents\sas_easy\scripts\flag.txt)[0]

#IF FLAG VALUE IS 0 THEN SCHEDULE NEXT RUN OF JOB ELSE BREAK THE SCHEDULAR
IF ($FLAG -EQ 0)
{
#CURRENT DATE
$CDATE= GET-DATE
#TIMESPAN BETWEEN CURRENT AND NEXT JOB
$WTIME=NEW-TIMESPAN –START $CDATE –END $BEGIN
#GET THE ABSOLUTE VALUE OF TIMESPAN
$ABSWTIME=$WTIME.NEGATE()
WRITE-HOST $WTIME
WRITE-HOST $ABSWTIME
#CALCULATE TOTAL SLEEP SECONDS BASED ON HOURLY (ARGS[0]), DAILY(ARGS[1]), WEEKLY
(ARGS[2]) OR MONTHLY (ARGS[3])JOBS
$SLEEPSECONDS=$ARGS[0]*$ARGS[1]*$ARGS[2]*$ARGS[3] - $ABSWTIME.TOTALSECONDS
WRITE-HOST $SLEEPSECONDS

#PUT THE SCRIPT ON SLEEP
START-SLEEP -S $SLEEPSECONDS
```

```
#RUN THE SAS JOB WHEN WAIT TIME ENDS
START-JOB C:\Users\Mnautiyal\Documents\sas_easy\scripts\sasbatchjobs.ps1
}
ELSE
{
BREAK
}
}

FUNCTION SCHEDULE
{
#THIS CONDITION RUNS ONLY FOR THE FIRST RUN. IT CALCULATES TIME BETWEEN EXECUTION TIME
AND CURRENT TIME. PUTS THE SCRIPT ON SLEEP FOR THAT PERIOD AND THEN EXECUTES IT
IF ($ONETIME -EQ 1)
{
$CDATE= GET-DATE
$WTIME=NEW-TIMESPAN –START $CDATE –END $BEGIN
WRITE-HOST $WTIME
$SLEEPSECONDS=$WTIME.TOTALSECONDS
WRITE-HOST $SLEEPSECONDS
START-SLEEP -S $SLEEPSECONDS
START-JOB C:\Users\Mnautiyal\Documents\sas_easy\scripts\sasbatchjobs.ps1
}
$ONETIME=$ONETIME + 1
#IF IT’S A ONETIME JOB THEN BREAK
IF ($ONETIMEJOB -EQ "Y")
{
BREAK
}

#IF IT’S AN HOURLY JOB THEN CALL AND PASS 3600 SECONDS TO THE WAIT_TIME FUNCTION
ELSEIF ($RUNJOBHOURLY -EQ "Y")
{
WAIT_TIME 3600 1 1 1
SCHEDULE
}

#IF IT’S A DAILY JOB THEN CALL AND PASS 3600*24 SECONDS TO THE WAIT_TIME FUNCTION
ELSEIF ($RUNJOBDAILY -EQ "Y")
{
```

```
WAIT_TIME 3600 24 1 1
SCHEDULE
}
#IF IT'S A WEEKLY JOB THEN CALL AND PASS 3600*24*7 SECONDS TO THE WAIT_TIME FUNCTION
ELSEIF ($RUNJOBWEEKLY -EQ "Y")
{
WAIT_TIME 3600 24 7 1
SCHEDULE
}
#IF IT'S A MONTHLY JOB THEN CALL AND PASS 3600*24*7*(TOTAL NUMBER OF DAYS IN THAT
MONTH) SECONDS TO THE WAIT_TIME FUNCTION
ELSEIF ($RUNJOBMONTHLY -EQ "Y")
{
$MONTH=(GET-DATE).MONTH
$YEAR=(GET-DATE).YEAR
$DAYSINMONTH=[DateTime]::DaysInMonth($YEAR, $MONTH)
WAIT_TIME 3600 24 1 $DAYSINMONTH
SCHEDULE
}
SCHEDULE
```

In case of one time run, the function SCHEDULE runs only once, but if a job is supposed to run multiple times then it calls function WAIT_TIME. WAIT_TIME calculates the sleep time between the runs and puts the script on sleep accordingly. This script assumes that total time of execution of all the SAS jobs is less than the time between repetitions of a cycle. It does not take day light savings factor into account. So, daylight saving might also cause issues during execution. In order to break a scheduled task which runs on regular intervals, one can change the value of FLAG variable from 0 to a non-zero value in the flag.txt file.

## CONCLUSION

UNIX scripting knowledge can be leveraged to write scripts in Windows PowerShell. By running SAS jobs using PowerShell scripts can definitely help in eliminating the need to write repetitive SAS programs. Jobs scheduled to run in the background also contribute in gaining efficiency in job execution, since such processing runs and writes the results to log and lst files without the need to open Editor, LOG or Output windows.

## REFERENCES

Hemedinger, Chris. "Running Windows PowerShell Scripts." Accessed on 07/27/2014. Available at http://blogs.sas.com/content/sasdummy/2011/09/12/running-windows-powershell-scripts/

Levy, Shay. "#PSTip Handling negative TimeSpan objects." PowerShell Magazine. Accessed on 07/27/2014. Available at http://www.powershellmagazine.com/2013/02/18/pstip-handling-negative-timespan-objects/.

## ACKNOWLEDGMENTS

I would like to thank the Southeast SAS Users Group for accepting my abstract and paper. I would also like to thank Laureano Gomez at Aspen Marketing Services – a division of Epsilon, Ambuj Nautiyal at Microsoft and Jessica Kneedler for providing feedback.

## CONTACT INFORMATION

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

Mayank Nautiyal
Epsilon
6 Concourse Parkway
Suite 2500,
Atlanta, GA 30328
mnautiyal@epsilon.com



## APPENDIX A

### PowerShell Script : print.ps1

```
#ASSIGNING SAS EXECUTION FILE TO VARIABLE SASCMD
$SASCMD="C:\Program Files\SASHome_94\SASFoundation\9.4\sas.exe"

#ASSIGNING SAS CONFIGURATION FILE TO VARIABLE SASCFG
$SASCFG="C:\Program Files\SASHome_94\SASFoundation\9.4\nls\en\sasv9.cfg"

#ASSIGNING SAS PROGRAM PRINT.SAS TO VARIABLE SASPRGM
$SASPRGM=" C:\Users\MNautiyal\Documents\sas_easy\pgm\print.sas"

#ASSIGNING SAS OUTPUT PRINT.LST TO VARIABLE SASPRGM
$SASLSTS=" C:\Users\MNautiyal\Documents\sas_easy\pgm\print.lst"

#ASSIGNING SAS LOG PRINT.LOG TO VARIABLE SASPRGM
$SASLOG=" C:\Users\MNautiyal\Documents\sas_easy\pgm\print.log"

#VARIABLE DSNAME CAPTURES THE FIRST INPUT PARAMETER (i.e. NAME OF DATA SET) AT
COMMAND LINE
$DSNAME=$args[0]

#VARIABLE OBS CAPTURES THE FIRST INPUT PARAMETER (i.e. NUMBER OF OBSERVATIONS) AT
COMMAND LINE
$OBS=$args[1]

#VARIABLE DSNAME AND OBS ARE DECLARED AS AN ENVIRONMENT VARIABLE SO THAT THEY CAN
BE EXPORTED TO THE SAS PROGRAM
$ENV:DSNAME=$DSNAME
$ENV:OBS=$OBS

#CMD COMMAND CALLS FOR SAS PROGRAM CONTENT.SAS
cmd /c " ""$SASCMD"" -nosplash -config ""$SASCFG"" -sysin ""$SASPRGM"" -log
""$SASLOG"" -print ""$SASLST"""

#GET-CONTENT COMMAND PRINTS THE LOG and LST ON SCREEN
GET-CONTENT -Encoding UTF8 $SASLOG
GET-CONTENT -Encoding UTF8 $SASLST
```

### SAS CODE: print.sas

```
LIBNAME OUT '.' ;
   OPTIONS MLOGIC SYMBOLGEN ;
   PROC PRINT DATA = OUT.%SYSGET(DSNAME) (OBS = %SYSGET(OBS));
   RUN ;
```

Set Alias as follows:

```
PS > set-alias print "C:\Users\Mnautiyal\Documents\sas_easy\scripts\print.ps1”
```

Submit by calling the alias name, name of the data set (class data set used from sashelp library) and number of observations as follows:

```
PS > print class 10
```

```
1          The SAS System                                08:03 Saturday, August 9, 2014



1          LIBNAME OUT '.' ;
NOTE: Libref OUT was successfully assigned as follows:
      Engine:        V9
      Physical Name: C:\Users\MNautiyal\Documents\data
2          OPTIONS MLOGIC SYMBOLGEN ;
3          proc print data = out.%sysget(dsname) (obs = %sysget(var));
4          run ;

NOTE: There were 10 observations read from the data set OUT.CLASS.
NOTE: The PROCEDURE PRINT printed page 1.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds


NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
NOTE: The SAS System used:
      real time           0.21 seconds
      cpu time            0.24 seconds

                    The SAS System          08:03 Saturday, August 9, 2014   1

        Obs    Name       Sex    Age    Height    Weight

          1    Alfred      M      14     69.0      112.5
          2    Alice       F      13     56.5       84.0
          3    Barbara     F      13     65.3       98.0
          4    Carol       F      14     62.8      102.5
          5    Henry       M      14     63.5      102.5
          6    James       M      12     57.3       83.0
          7    Jane        F      12     59.8       84.5
          8    Janet       F      15     62.5      112.5
          9    Jeffrey     M      13     62.5       84.0
         10    John        M      12     59.0       99.5
```

**Output 3. Output of alias print submitted at the command prompt for class data set and 10 observations**

## APPENDIX B

### PowerShell Script : freq.ps1

```
 #ASSIGNING SAS EXECUTION FILE TO VARIABLE SASCMD
$SASCMD="C:\Program Files\SASHome_94\SASFoundation\9.4\sas.exe"

#ASSIGNING SAS CONFIGURATION FILE TO VARIABLE SASCFG
$SASCFG="C:\Program Files\SASHome_94\SASFoundation\9.4\nls\en\sasv9.cfg"

#ASSIGNING SAS PROGRAM FREQ.SAS TO VARIABLE SASPRGM
$SASPRGM=" C:\Users\MNautiyal\Documents\sas_easy\pgm\freq.sas"

#ASSIGNING SAS OUTPUT FREQ.LST TO VARIABLE SASPRGM
$SASLSTS=" C:\Users\MNautiyal\Documents\sas_easy\pgm\freq.lst"

#ASSIGNING SAS LOG FREQ.LOG TO VARIABLE SASPRGM
$SASLOG=" C:\Users\MNautiyal\Documents\sas_easy\pgm\freq.log"

#VARIABLE DSNAME CAPTURES THE FIRST INPUT PARAMETER (i.e. NAME OF DATA SET) AT
COMMAND LINE
$DSNAME=$args[0]

#VARIABLE VAR CAPTURES THE FIRST INPUT PARAMETER (i.e. VARIABLE NAME) AT COMMAND
LINE
$VAR=$args[1]

#VARIABLE DSNAME AND OBS ARE DECLARED AS AN ENVIRONMENT VARIABLE SO THAT THEY CAN
BE EXPORTED TO THE SAS PROGRAM
$ENV:DSNAME=$DSNAME
$ENV:VAR=$VAR

#CMD COMMAND CALLS FOR SAS PROGRAM CONTENT.SAS
cmd /c " ""$SASCMD"" -nosplash -config ""$SASCFG"" -sysin ""$SASPRGM"" -log
""$SASLOG"" -print ""$SASLST"""

#GET-CONTENT COMMAND PRINTS THE LOG and LST ON SCREEN
GET-CONTENT -Encoding UTF8 $SASLOG
GET-CONTENT -Encoding UTF8 $SASLST
```

### SAS CODE: freq.sas

```
LIBNAME OUT '.' ;
OPTIONS MLOGIC SYMBOLGEN ;
PROC FREQ DATA = OUT.%SYSGET(DSNAME);
TABLE %SYSGET(VAR)/MISSING;
RUN ;
```

Set Alias as follows:

```
PS > set-alias freq "C:\Users\Mnautiyal\Documents\sas_easy\scripts\freq.ps1”
```

Submit by calling the alias name, name of the data set and variable name

```
> freq class sex
```

```
1          LIBNAME OUT '.' ;
NOTE: Libref OUT was successfully assigned as follows:
      Engine:        V9
      Physical Name: C:\Users\MNautiyal\Documents\data

2          OPTIONS MLOGIC SYMBOLGEN ;
3          proc freq data = out.%sysget(dsname);
4          table %sysget(var) ;
5          run ;

NOTE: There were 19 observations read from the data set OUT.CLASS.
NOTE: The PROCEDURE FREQ printed page 1.
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds


                    The SAS System              08:14 Saturday, August 9, 2014

NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
NOTE: The SAS System used:
      real time           0.20 seconds
      cpu time            0.20 seconds

                    The SAS System          08:14 Saturday, August 9, 2014   1

                                  The FREQ Procedure

                                                 Cumulative    Cumulative
           Sex    Frequency     Percent     Frequency      Percent
           ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ
           F              9       47.37              9        47.37
           M             10       52.63             19       100.00
```

**Output 4. Output of alias freq submitted at the command prompt for class data set and sex variable**

In order to get a cross tabulation between two or more variables the second argument (i.e. VAR) should look like (var1*var2):

```
> freq class male*age
```

```
1          LIBNAME OUT '.' ;
NOTE: Libref OUT was successfully assigned as follows:
      Engine:        V9
      Physical Name: C:\sas_shortcuts\scripts
2          OPTIONS MLOGIC SYMBOLGEN ;
3          proc freq data = out.%sysget(dsname);
4          table %sysget(var) ;
5          run ;

NOTE: There were 19 observations read from the data set OUT.CLASS.
NOTE: The PROCEDURE FREQ printed page 1.
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.04 seconds
      cpu time            0.04 seconds
```

```
            The SAS System              08:30 Saturday, August 9, 2014

NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
NOTE: The SAS System used:
      real time           0.17 seconds
      cpu time            0.18 seconds


            The SAS System          08:30 Saturday, August 9, 2014   1

                                The FREQ Procedure

                                Table of Sex by Age

              Sex       Age

              Frequency
              Percent
              Row Pct
              Col Pct         11      12      13      14      15      16  Total
              ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ
              F               1       2       2       2       2       0       9
                             5.26   10.53   10.53   10.53   10.53    0.00   47.37
                            11.11   22.22   22.22   22.22   22.22    0.00
                            50.00   40.00   66.67   50.00   50.00    0.00
              ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ
              M               1       3       1       2       2       1      10
                             5.26   15.79    5.26   10.53   10.53    5.26   52.63
                            10.00   30.00   10.00   20.00   20.00   10.00
                            50.00   60.00   33.33   50.00   50.00  100.00
              ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ
              Total           2       5       3       4       4       1      19
                            10.53   26.32   15.79   21.05   21.05    5.26  100.00
```

**Output 5. Output of alias freq submitted at the command prompt for class data set and sex*age variables**