Paper AD-149

You've Got Mail®: Automating SAS® from an Email

Peter P. Davis, U.S. Census Bureau; Mark E. Asiala, U.S. Census Bureau

ABSTRACT

The American Community Survey is an ongoing population and housing survey that provides data every year – giving communities the current information they need to plan investments and services. As such, repetitive processing is necessary and must be completed in a timely manner. Automation, where appropriate, is an essential component for operational efficiency.

As an example of where automation is implemented, we receive an email each month that serves as notification that one operation has completed and the next operation may begin. Instead of waiting for this email to manually submit our SAS programs, what if the delivery of the email initiated our SAS programs?

This paper demonstrates a KornShell (ksh93) script which parses through an email delivered to a user's UNIX email account. The script "reads" the email. As long as the deliverer and the subject of the email meet certain requirements, the appropriate SAS programs are submitted. If not, an email is sent to the user stating that an email was received but no further action occurred.

DISCLAIMER

This report is released to inform interested parties of ongoing research and to encourage discussion of work in progress. Any views expressed are those of the authors and not necessarily those of the Census Bureau.

INTRODUCTION

In this paper, we demonstrate how we use the Unix operating system to automate our SAS programs. Prior to understanding and having a greater appreciation of Unix, we waited for the arrival of an email which served as notification we could begin processing. What if the email we received automatically set the parameters for our SAS programs, initiated the SAS programs, verified the successful completion of our SAS programs, and then sent an email when it was done? An hour of processing is reduced to less than 10 minutes, assuming there are no errors. This paper explains how to integrate SAS with a KornShell program (script) by taking a step-by-step look at how to initiate your programs from an email.

This paper delves heavily into KornShell programming and is light on SAS. Once the two are combined, your SAS programs will be executed, maintained, and monitored by the scripts allowing the user time for other tasks. To understand the KornShell scripts in this paper, some basic knowledge of KornShell programming and the Unix operating system commands is useful. See Robbins (1999) and Rosenberg (1999) for more details on these topics. We demonstrate the key features of the scripts by taking a step-by-step approach, walking you through the entire system. Throughout the paper, KornShell variables are CAPITALIZED, commands are in *italics*, and comments in the code start with the pound sign (#).

EMAIL SYSTEM OVERVIEW

Figure 1 displays the process flow. An email (Box 1), delivered to a user's Unix account, starts the email system. In Box 2, the ~/.forward file delivers the email to the email script in Box 3. The email script "reads" the email and under certain conditions executes the boot script (Box 4). The boot script manages and kick starts SAS by defining SAS parameters, executing multiple SAS programs, examining the SAS lst and log files for errors, and then delivering an email notifying interested parties that the SAS programs are complete (Box 5 – Box 8). An additional email is delivered (Box 9) which provides the results of the boot script (Box 4) and notifies users that the process is complete. Let's take the boxes in Figure 1 one at a time.

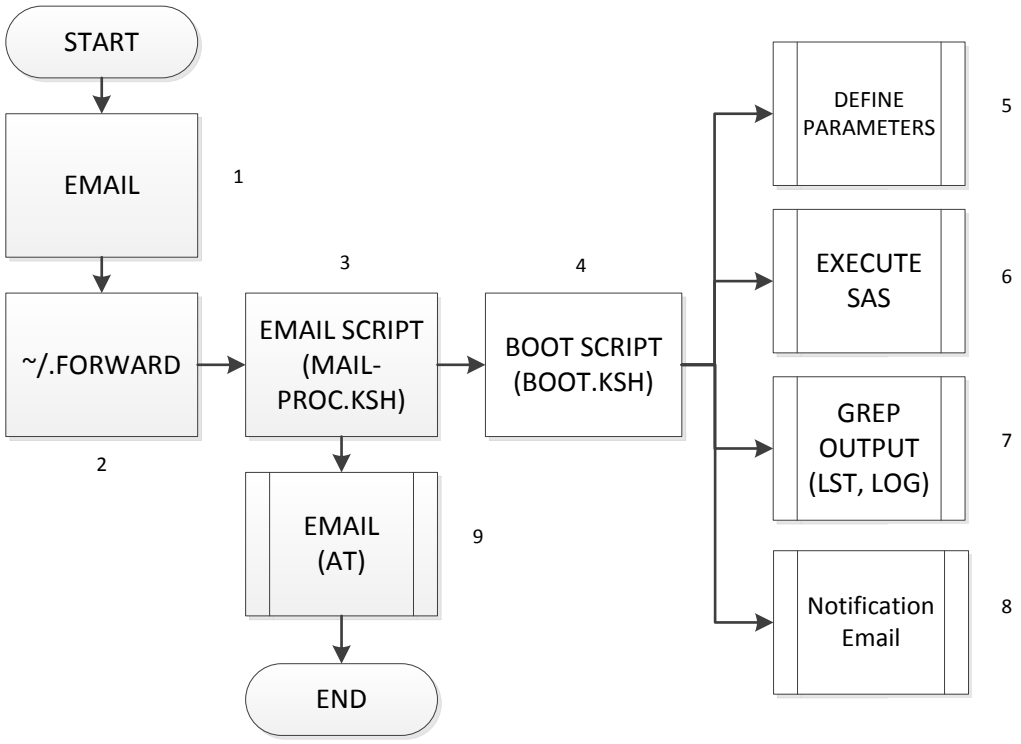

Figure 1: Email System Overview

~/.FORWARD

BOX 2

A key to the whole system running successfully is the understanding of sendmail and the ~/.forward file, commonly referred to as the "dot forward file". Sendmail is a Unix-based implementation of the Simple Mail Transfer Protocol (SMTP) for transmitting email. When delivering an email to a user (Box 1), sendmail checks to see if the user has a ~/.forward file in the user's home directory. If the file is present, the contents are read and the actions listed in the ~/.forward file are performed. Below are the contents of our ~/.forward file.

```
~/mail/inmail.tmp
"|~/mail/mail-proc.ksh"
```

The first line of the ~/.forward file takes the contents of the email as standard input and creates a temporary copy of the email, stored as ~/mail/inmail.tmp. See Appendix A to view the email. The second line executes a KornShell program called mail-proc.ksh again using the same email as standard input. The next section describes the purpose of the temporary file and the KornShell program.

The ~/.forward file can be used in many different ways. We could have developed an alternate method that did not require the temporary file. We chose to use the ~/.forward file in the method described above to learn KornShell programming and for debugging purposes.

EMAIL SCRIPT (MAIL-PROC.KSH)

BOX 3

The KornShell program, mail-proc.ksh, "reads" the copy of the email, performs the specified commands, and then deletes the temporary copy of the email. The goal of the email script is to verify that the subject and sender of the email are valid. If both the subject and sender meet certain conditions, then our SAS programs are run via another script (Box 4). If the values of either the subject or sender are invalid, then no programs are run and a new email is delivered to the project manager notifying them that an email was delivered to their Unix account and no programs were executed. The entire email script is in Appendix B.

To verify the subject and sender of the email are valid, we parse through the email and capture the subject and sender. To do this, we begin the email script by defining some KornShell variables that use three commands or processes – *grep*, pipe (|), and *sed*. The file name of the temporary copy of the email is stored as TMPFILE. To capture the subject of the email we begin with the grep command.

```
TMPFILE=~/mail/inmail.tmp
SUBJECT=`grep '^Subject:' $TMPFILE | sed -e 's/^Subject: *//'`
```

Grep searches a file for the line that matches a pattern and prints the line. For example, *grep* searches $TMPFILE[1] for the line that starts with "Subject: ". The caret (^) means "begins a line with". Using the email in Appendix A, the result of the *grep* command is "Subject: REVIEW SAS FILES". The pipe operator (|) instructs the KornShell to take the output from the first command (*grep*) and use it as the input to the next process (*sed*).

Sed is a stream-oriented editor. Here we use *sed* with the substitution command. The syntax of the *sed* substitution is "s/*pattern*/*replacement*/". We search the results of the *grep* command to substitute "Subject: " with blank space, effectively deleting "Subject: ". Using the result of the *grep* command as input, the result of *sed* defines the variable SUBJECT as "REVIEW SAS FILES". The variable ADDRESS is assigned in a similar manner.

To capture the sender, we are interested in the prefix of the email address, that is, all the information about the email address prior to @. This is assigned to the variable FROM as follows.

```
FROM="${ADDRESS%@*}"
```

The last three variables are easier to assign. YEAR is defined by capturing the system four-digit year. ENVR is the environment we are working in, "prod" which stands for "production", and SENDER is from whom we expect to receive the official email.

```
YEAR=$(date +"%Y")
ENVR=prod
SENDER=johndoe
```

The next section of the email script determines whether or not the subject and sender of the email are valid. This is accomplished through a conditional statement.

```
if [[ $FROM == $SENDER ]] && [[ $SUBJECT == "REVIEW SAS FILES" ]]
then

     at -f /data/$YEAR/$ENVR/pgms/boot.ksh now
```

If the value of FROM matches the value of SENDER and the value of SUBJECT matches "REVIEW SAS FILES", then the boot script (boot.ksh) is submitted using the *at* command. Other commands are available to submit a script. However, we use the *at* command because output from the specified commands in the script, in this instance, from the commands in boot.ksh, are emailed to the user. We will see this in more detail when we discuss Box 9.

If the value of FROM does not match the value of SENDER or the value of SUBJECT does not match "REVIEW SAS FILES", then the original email stored in TMPFILE is returned to the specified user $EMAIL.

```
else
     mailx -s "Received Email - $SUBJECT" $EMAIL < $TMPFILE
fi
```

The command *mailx* sends internet mail. The option, -s, specifies the subject of the email. The email is delivered to $EMAIL and < inserts TMPFILE in to the body of the email. The conditional *if* statement closes with if spelled backwards, *fi*. The program finishes by deleting the original email using the remove command, *rm*.

```
rm $TMPFILE
```

BOOT SCRIPT (BOOT.KSH)

BOX 4

This is where the SAS automation really begins. Everything up until this point has been checking to see if the email is appropriate and that our SAS programs should be initiated based on the email we received. Let's assume the conditional statements for the email are met, that is, the $SENDER and $SUBJECT of the email match the valid values. Then as we showed, the boot script (boot.ksh) is run using the *at* command. This brings us to Box 4 in Figure 1, the boot script. The boot program in Appendix C accomplishes four things.

1. It defines the parameters used in the SAS programs, Box 5.
2. It executes four SAS programs, sequentially, Box 6.
3. It runs a diagnostic check of the SAS log and SAS lst files, Box 7.
4. It sends a notification email that the above tasks are completed, Box 8.

[1] The following syntax represents the value of a variable: $*variable*

BOX 5

The code in the boot script that pertains to defining the parameters is below.

```
# Set environment variable
ENVR=prod
LOCO=data
printf "\nProgram started $(date). Environment is $ENVR.\n\n"

# Capture current system year and month
PYEAR=$(date +"%Y")
PMONTH=$(date +"%m")

# Assign Panel
PANEL="$PYEAR$PMONTH"

printf "\nMAIN Programs\n"
```

The above code is similar to what we have already discussed. There are two new items we need to mention. First, we assign the variable PANEL by concatenating the variables PYEAR and PMONTH. And second, we introduce the *printf* command. *Printf* prints an argument in a specified format. The result of the two *printf* commands above is seen in Output 1. The \n represents a new line.

```
Program started Sun Jul 12 16:01:11 EDT 2015. Environment is prod.


MAIN Programs
```

Output 1: *Printf* Output from Box 5

BOX 6

The next section of the code in the boot script executes four SAS programs, sequentially. Each program can not begin unless the previous program has run without error. The architecture of our Unix environment is set up so that programs, log files, lst files, and SAS data sets reside in separate directories. In this way, we maintain a structured and manageable SAS system in a Unix environment.

Let's examine how the first SAS program, sampling.sas, is submitted. Before we execute any of the SAS programs, we define the path and file name of the SAS log and SAS lst files. This section of the boot script concludes with a *printf* command to verify we have accurately defined all variables.

```
# SAMPLING
LOG_SMP=/data/$PYEAR/$ENVR/logs/sampling-$PMONTH$PYEAR.log
OUT_SMP=/data/$PYEAR/$ENVR/output/sampling-$PMONTH$PYEAR.lst

printf "\nYEAR=$PYEAR MONTH=$PMONTH PANEL=$PANEL \n  LOG=$LOG_SMP \n  OUT=$OUT_SMP"
```

All of the KornShell variables we have defined in the boot script are exported to the SAS program using the *export* command. In the SAS program, we define these KornShell variables as macro variables which are used throughout the SAS program. The SAS program is executed with the defined log, output paths, and file names.

```
export PYEAR
export PMONTH
export PANEL
export LOCO
export ENVR

sas -LOG $LOG_SMP -PRINT $OUT_SMP /data/$PYEAR/$ENVR/pgms/sampling.sas
```

Finally, every KornShell statement, script, command, and program returns an error status (or exit status). The KornShell automatically assigns the error status to the variable $?. The error status is an integer that symbolizes the success or failure of the statement, script, command, or program. An error status of 0 indicates success, and a nonzero error status indicates failure. In the next section of the boot script, we assign the exit status ($?) to a variable and check if it is nonzero. If it is nonzero, the SAS program failed and the boot script exits (stops); there is no point in having the boot script continue if any one of the SAS program fails.

```
# SAS Error Check
ERR2=$?
if [[ $ERR2 > 0 ]]
then
     printf "\nError $ERR2 returned by SAS.  Check $LOG_SMP.  Exiting (Error=2)"
     exit 2
fi
```

Each of the four SAS programs is executed in the same manner as seen in the boot script in Appendix C.

BOX 7

After the boot script runs the SAS programs and no errors are returned from the exit status ($?), we perform an additional check of all the log files and all the lst files. Below is the code that examines the log files.

```
# Check LOG files.
if [[  -e $LOG_SMP ]] && [[ -e $LOG_VER ]] && [[ -e $LOG_VAL ]] && [[ -e $LOG_SUM
]]
then
     printf "\n\n\nLOG FILE Validation"
     printf "\n\nThese log files contain the word(s) ERROR, error, WARNING, or
      warning.  The number of issues is displayed for each file.\n"
     egrep -c 'ERROR|error|WARNING|warning' $LOG_SMP $LOG_VER $LOG_VAL $LOG_SUM
     printf "\nImportant ERROR or WARNING messages from the log files, if any.\n"
     egrep -n '^ERROR' $LOG_SMP $LOG_VER $LOG_VAL $LOG_SUM
     egrep -n '^"WARNING' $LOG_SMP $LOG_VER $LOG_VAL $LOG_SUM
else
     printf "\n\nAt least one of the LOG FILES does not exist."
fi
```

Beginning with an *if* statement, we check to see if each of the four log files exists. If at least one does not exist, the *printf* command let's us know. If all four log files exist, we check to see if there are any statements in the log files containing "ERROR", "error", "WARNING", or "warning". We accomplish this by using the *egrep* command. *Egrep* is very similar to the *grep* command discussed earlier. Here, we use *egrep* to count (*egrep –c*) the number of times any of the four log files contain "ERROR", "error", "WARNING", or "warning". (Hopefully, this is zero.) Then, using *egrep* again, we display the line number (egrep –n) of the log file that contains either an "ERROR" statement or a "WARNING" statement, if any exists. Output from our log files without any errors is displayed in Output 2.

```
These log files contain the word(s) ERROR, error, WARNING, or warning.  The number
of issues is displayed for each file.
/data/2015/prod/logs/sampling-072015.log:0
/data/2015/prod/logs/verification-072015.log:0
/data/2015/prod/logs/validation-072015.log:0
/data/2015/prod/logs/summary-072015.log:0
```

Output 2: *Egrep* Output from Box 7

In a similar fashion, we perform simple checks of the SAS lst file. In one of the four SAS programs, there exists seven PROC COMPARE statements. We know that a successful PROC COMPARE produces the following statement: "NOTE: No unequal values were found. All values compared are exactly equal." With that knowledge, we use the *egrep* command to print any instance of a line that begins with "NOTE:" from the lst file that contains the PROC COMPARE statements. We accomplish this as follows.

```
egrep -n '^NOTE:' $OUT_VER
```

By using the *egrep* command, you may request to view any line(s) from any SAS lst file(s). For example, we added an additional check to view any differences in the SAS variable RN.

```
printf "\n\nRANDOM NUMBER Validation"
printf "\n\nExpect differences in the random number (rn). The maximum difference
should be really small.\n"
egrep -B 2 "^rn" $OUT_VER
```

The command *egrep –B 2* prints two lines of leading context before the matching line that begins with "rn". Therefore, we are printing three lines of output from the SAS lst file which includes the line that starts with "rn".

BOX 8

A notification email (Box 8) concludes the boot script. We use the print command (*printf*) to create a formatted email containing tab indents (\t). The formatted email is stored in the KornShell variable, MFILE. MFILE is then mailed to interested parties using *mailx*. See Appendix C for more details.

EMAIL (AT)

BOX 9

At a quick glance, it appears the email system is finished. However, this is not the case. Recall in Figure 1 – Box 3, the email script, we use the *at* command to run the boot script (Box 4). The *at* command has a nice feature – the user is mailed all the output invoked from the *at* command. In this instance, the user is emailed all the commands from the boot script in Box 4. Therefore, the results of all the *printf* and *egrep* commands, along with all the diagnostic checks from the boot script that control the running of the SAS programs are delivered to the user via email. See Appendix D for an example of the delivered email. The email from the *at* command provides a nice summary of the SAS programs and their performance.

CONCLUSION

For future research, we plan to investigate sendmail and the ~/.forward file a little more. In developing this automated email system, we discovered that the sendmail utility assigns environment variables, for example, it assigns the subject of an incoming email to a variable. Leveraging existing environment variables from sendmail may eliminate the need to parse through the email as we did in the email script (Figure 1 – Box 3). Our current email system provided an excellent learning experience and aided in debugging during development.

While we realize this paper relies heavily on KornShell programming and is light on SAS, we hope you find this paper useful. Combining the features of SAS and the KornShell has improved our operational efficiency. We constantly look for ways to integrate SAS and the KornShell to automate our processes and to make our lives easier.

REFERENCES

Robbins, Arnold, 1999, "Unix in a Nutshell", 3rd Edition, Sebastopol, CA: O'Reilly & Associates, Inc.

Rosenberg, Barry, 1999, "Hands-On KornShell93 Programming", Boston, MA: Addison-Wesley.

ACKNOWLEDGMENTS

Thanks to my coauthor, Mark Asiala, for providing guidance. What started as a simple question to Mark, "Can an email start our SAS programs?" turned in to a how-to-automate-SAS project using KornShell programming.

CONTACT INFORMATION

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

Peter P. Davis
U.S. Census Bureau
4600 Silver Hill Rd
Washington, D.C. 20233
301-763-4291
peter.p.davis@census.gov
www.census.gov

Mark E. Asiala
U.S. Census Bureau
4600 Silver Hill Rd
Washington, D.C. 20233
301-763-3605
mark.e.asiala@census.gov
www.census.gov

APPENDIX A – EMAIL

```
[[…Internal sendmail information edited for security purposes…]
Subject: REVIEW SAS FILES
[[…Internal sendmail information edited for security purposes…]
From: johndoe@census.gov (User Account)

Please review the SAS files /data/sample.sas7bdat, and /data/weights.sas7bdat and let
us know if there are any problems.  If nothing is reported, the file will
automatically be delivered in one hour.
```

APPENDIX B – EMAIL SCRIPT (MAIL-PROC.KSH)

```
#!/bin/ksh
#      File:         MAIL-PROC.KSH
#      Description:  This script manages email.
#
#                    Email delivered to a user follows two steps.
#                    1. Contents of the ~/.forward file are read and implemented.
#                    2. ~/mail/MAIL-PROC.KSH is submitted.
#
#                    The SUBJECT, FROM, and ADDRESS are captured from each email.
#                    Depending on the SUBJECT, FROM, and ADDRESS, each email is
#                    handled appropriately.
#
#================================================================================
umask 003

#  PARSE EMAIL
#
#  The original email is temporarily stored (TMPFILE).
#  Extract full email address (ADDRESS) and email subject (SUBJECT) from the email
#  (TMPFILE).
#  Extract the sender prefix (FROM) from the full email address (ADDRESS).  For
#  example, extract "johndoe" from johndoe@census.gov.

TMPFILE=~/mail/inmail.tmp
EMAIL='peter.p.davis@census.gov'
SUBJECT=`grep '^Subject:' $TMPFILE | sed -e 's/^Subject: *//'`
ADDRESS=`grep '^From:' $TMPFILE | sed -e 's/^From: *//'`
FROM="${ADDRESS%@*}"
YEAR=$(date +"%Y")
ENVR=prod
SENDER=johndoe

#  Operation:   SAMPLING
#
#  Action:      Runs the boot script
#
#  Occurrence:  Monthly, at the end of the month

if [[ $FROM == $SENDER ]] && [[ $SUBJECT == "REVIEW SAS FILES"]]
then

        at -f /data/$YEAR/$ENVR/pgms/boot.ksh now

# Operation:    No Processing
#
# Action:       If no script has been identified to run, the original email is
#               returned.
#
# Occurrence:   This happens only when all other email conditions fail.

else
        mailx -s "Received Email - $SUBJECT" $EMAIL < $TMPFILE
fi

#  Action:      Delete the temporary email that was stored for processing.
#
#  Occurrence:  Any time an email is delivered to the user's account.

rm $TMPFILE
```

APPENDIX C – BOOT SCRIPT (BOOT.KSH)

```
#!/bin/ksh
#      File:         BOOT.KSH
#
#      Description:  BOOT.KSH kick starts and manages the SAS programs.
#                    BOOT.KSH runs based off the delivery of an email. The email
#                    must meet certain conditions in order to run this script.
================================================================================
umask 003

# Set environment variable
ENVR=prod
LOCO=data
printf "\nProgram started $(date). Environment is $ENVR.\n\n"

# Capture current system year and month
PYEAR=$(date +"%Y")
PMONTH=$(date +"%m")

# Assign Panel
PANEL="$PYEAR$PMONTH"

printf "\nMAIN Programs\n"

# SAMPLING
LOG_SMP=/data/$PYEAR/$ENVR/logs/sampling-$PMONTH$PYEAR.log
OUT_SMP=/data/$PYEAR/$ENVR/output/sampling-$PMONTH$PYEAR.lst

printf "\nYEAR=$PYEAR MONTH=$PMONTH PANEL=$PANEL \n  LOG=$LOG_SMP \n  OUT=$OUT_SMP"

export PYEAR
export PMONTH
export PANEL
export LOCO
export ENVR

sas -LOG $LOG_SMP -PRINT $OUT_SMP /data/$PYEAR/$ENVR/pgms/sampling.sas

# SAS Error Check
ERR2=$?
if [[ $ERR2 > 0 ]]
then
        printf "\nError $ERR2 returned by SAS.  Check $LOG_SMP.  Exiting (Error=2)"
        exit 2
fi

# VERIFICATION
LOG_VER=/data/$PYEAR/$ENVR/logs/verification-$PMONTH$PYEAR.log
OUT_VER=/data/$PYEAR/$ENVR/output/verification-$PMONTH$PYEAR.lst

printf "\nYEAR=$PYEAR MONTH=$PMONTH PANEL=$PANEL \n  LOG=$LOG_VER \n  OUT=$OUT_VER"

export PYEAR
export PMONTH
export PANEL
export LOCO
export ENVR

sas -LOG $LOG_VER -PRINT $OUT_VER /data/$PYEAR/$ENVR/pgms/verification.sas

# SAS Error Check
ERR3=$?
if [[ $ERR3 > 0 ]]
then
```

```
        printf "\nError $ERR3 returned by SAS.  Check $LOG_VER.  Exiting (Error=3)"
        exit 3
fi

# VALIDATION
LOG_VAL=/data/$PYEAR/$ENVR/logs/validation-$PMONTH$PYEAR.log
OUT_VAL=/data/$PYEAR/$ENVR/output/validation-$PMONTH$PYEAR.lst

printf "\nYEAR=$PYEAR MONTH=$PMONTH PANEL=$PANEL \n  LOG=$LOG_VAL \n  OUT=$OUT_VAL"

export PYEAR
export PMONTH
export PANEL
export LOCO
export ENVR

sas -LOG $LOG_VAL -PRINT $OUT_VAL /data/$PYEAR/$ENVR/pgms/validation.sas

# SAS Error Check
ERR4=$?
if [[ $ERR4 > 0 ]]
then
        printf "\nError $ERR4 returned by SAS.  Check $LOG_VAL.  Exiting (Error=4)"
        exit 4
fi

# SUMMARY
LOG_SUM=/data/$PYEAR/$ENVR/logs/summary-$PMONTH$PYEAR.log
OUT_SUM=/data/$PYEAR/$ENVR/output/summary-$PMONTH$PYEAR.lst

printf "\nYEAR=$PYEAR MONTH=$PMONTH PANEL=$PANEL \n  LOG=$LOG_SUM \n  OUT=$OUT_SUM"

export PYEAR
export PMONTH
export PANEL
export LOCO
export ENVR

sas -LOG $LOG_SUM -PRINT $OUT_SUM /data/$PYEAR/$ENVR/pgms/summary.sas

# SAS Error Check
ERR5=$?
if [[ $ERR5 > 0 ]]
then
        printf "\nError $ERR5 returned by SAS.  Check $LOG_SUM.  Exiting (Error=5)"
        exit 5
fi

# Check LOG files.
if [[  -e $LOG_SMP ]] && [[ -e $LOG_VER ]] && [[ -e $LOG_VAL ]] && [[ -e $LOG_SUM ]]
then
        printf "\n\n\nLOG FILE Validation"
        printf "\n\nThese log files contain the word(s) ERROR, error, WARNING, or
         warning.  The number of issues is displayed for each file.\n"
        egrep -c 'ERROR|error|WARNING|warning' $LOG_SMP $LOG_VER $LOG_VAL $LOG_SUM
        printf "\nImportant ERROR or WARNING messages from the log files, if any.\n"
        egrep -n '^ERROR' $LOG_SMP $LOG_VER $LOG_VAL $LOG_SUM
        egrep -n '^"WARNING' $LOG_SMP $LOG_VER $LOG_VAL $LOG_SUM
else
        printf "\n\nAt least one of the LOG FILES does not exist."
fi

# Check LST files.
if [[  -e $OUT_SMP ]] && [[ -e $OUT_VER ]] && [[ -e $OUT_VAL ]] && [[ -e $OUT_SUM ]]
then
```

```
        printf "\n\nPROC COMPARE Validation"
        printf "\n\nSeven (7) PROC COMPAREs are evaluated in $OUT_VER."
        printf "\nThe line number of each NOTE: statement is displayed.\n"
        egrep -n '^NOTE:' $OUT_VER

        printf "\n\nRANDOM NUMBER Validation"
        printf "\n\nExpect differences in the random number (rn). The maximum
         difference should be really small.\n"
        egrep -B 2 "^rn" $OUT_VER
        printf " "
else
        printf "\nAt least one of the LST FILES does not exist."
fi

# Notification email
MFILE=/data/$PYEAR/$ENVR/output/notice.txt
PEMAIL='peter.p.davis@census.gov'
CEMAIL='mark.e.asiala@census.gov'

printf "\n\nSampling is complete.
\nThe following programs have been run:
\t/data/$PYEAR/$ENVR/pgms/
\t1. boot.ksh
\t2. sampling.sas
\t3. verification.sas
\t4. validation.sas
\t5. summary.sas
\n\n\nReview the sampling, verification and validation output below.
\nOutput is available at:
\t/data/$PYEAR/$ENVR/output/
\t1. sampling-<MM><YYYY>.lst
\t2. verification-<MM><YYYY>.lst
\t3. validation-<MM><YYYY>.lst
\t4. summary-<MM><YYYY>.lst
\n\nSAS Log information is available at:
\t/data/$PYEAR/$ENVR/logs/
\t1. sampling-<MM><YYYY>.log
\t2. verification-<MM><YYYY>.log
\t3. validation-<MM><YYYY>.log
\t4. summary-<MM><YYYY>.log
\n\n\nAdditional information is available at Z:/Sampling Results/." > $MFILE

mailx -s "Sampling is Complete" -c $CEMAIL $PEMAIL < $MFILE

printf "\n\nProgram finished $(date)."
```

APPENDIX D – EMAIL (AT)

```
Program started Mon Jun 29 10:12:04 EST 2015.  Environment is prod.

MAIN Programs

YEAR=2015 MONTH=07 PANEL=201507
  LOG=/data/2015/prod/logs/sampling-072015.log
  OUT=/data/2015/prod/output/sampling-072015.lst
YEAR=2015 MONTH=07 PANEL=201507
  LOG=/data/2015/prod/logs/verification-072015.log
  OUT=/data/2015/prod/output/verification-072015.lst
YEAR=2015 MONTH=07 PANEL=201507
  LOG=/data/2015/prod/logs/validation-072015.log
  OUT=/data/2015/prod/output/validation-072015.lst
YEAR=2015 MONTH=07 PANEL=201507
  LOG=/data/2015/prod/logs/summary-072015.log
  OUT=/data/2015/prod/output/summary-072015.lst

LOG FILE Validation

These log files contain the word(s) ERROR, error, WARNING, or warning.  The number of
issues is displayed for each file.
/data/2015/prod/logs/sampling-072015.log:0
/data/2015/prod/logs/verification-072015.log:0
/data/2015/prod/logs/validation-072015.log:0
/data/2015/prod/logs/summary-072015.log:0


PROC COMPARE Validation

Seven (7) PROC COMPAREs are evaluated in /data/2015/prod/output/verification-
072015.lst.
The line number of each NOTE: statement is displayed.
45:NOTE: No unequal values were found. All values compared are exactly equal.
97:NOTE: No unequal values were found. All values compared are exactly equal.
297:NOTE: No unequal values were found. All values compared are exactly equal.
400:NOTE: No unequal values were found. All values compared are exactly equal.
498:NOTE: No unequal values were found. All values compared are exactly equal.
691:NOTE: The MAXPRINT=50 printing limit has been reached for the variable rn. No more
values will be printed for this comparison.
788:NOTE: No unequal values were found. All values compared are exactly equal.


RANDOM NUMBER Validation

Expect differences in the random number (rn). The maximum difference should be really
small.
Variable  Type  Len  Ndif   MaxDif

rn        NUM     8  8254  114E-15

Program finished Mon Jun 29 10:12:56 EST 2015.
```