Paper 229-2013

Here Comes Your File!

File-Watcher Tool with Automated SAS® Program Trigger

Rajbir Chadha, Cognizant Technology Solutions

ABSTRACT

In a typical warehousing environment, users are notified about file availability by system generated e-mails. Users then run their SAS processes. There was a need for a file-watcher tool to trigger the user processes as soon as input files become available.

This paper talks about a file-watcher tool that searches for files and looks to see when they were last updated. The tool uses UNIX shell scripting to search for the file. The parameters to the file-watcher tool are supplied using a 'wrapper' script that can be modified by the user as and when required. It can be programmed to look for the file within a specified time period. The script is scheduled using a 'CRON' scheduler in UNIX.

The file-watcher searches for the updated file based on the specified parameters. Once the updated version is found, the SAS program execution begins. If the file is not found or not updated within the specified time period, the script terminates. The tool sends out e-mails when the files are available and when the SAS program completes execution or the script terminates. In case of errors, users can refer to the file-watcher logs at a location specified in the CRON file. Errors in the SAS program are tracked in the SAS log at a location specified in the wrapper script.

The file-watcher tool reduces average wait time and manual effort for users by automating most of the process, allowing them to focus on other pressing tasks.

INTRODUCTION

Users working on SAS datasets and generating reports, rely on files and datasets published in the Data Warehouse. Often system generated e-mails are sent out to notify them about the availability of files. Users then run their programs to process data and generate reports. Such notifications are often sent in batches and may not be sent for every file.

This paper describes an easy-to-use UNIX File-Watcher script that scans for the file and notifies the user as soon as it becomes available. The File-Watcher uses a wrapper script to define parameters. Users have access to the wrapper script and can modify or update the search parameters as required. Wrapper script (described later in this paper) is scheduled in UNIX using CRON scheduler and can trigger relevant SAS programs when the files become available.

BACKGROUND AND ASSUMPTIONS

This paper assumes that users are familiar with the UNIX environment (i.e. file structure, environment variables and basic UNIX commands). Users need to be able to modify the parameters in the script for the file-watcher to function.

This paper uses ksh shell for all UNIX scripting. Some commands might be different from the other flavors of UNIX such as Bash, C shell, etc.

FILE-WATCHER SCRIPT

A UNIX script intended to look for specific files on the file system is typically called a file-watcher. Such scripts monitor one or more folders for new or updated files at regular intervals. File-watcher can be kept running at all times with inbuilt sleep cycles, or scheduled in a CRON like scheduler. The script can send out e-mails to users and also trigger custom actions like kicking off a downstream process. Such scripts can be used for pre-processing files as soon as they arrive, manage incoming FTP files, load files in databases etc.

File-Watcher script uses the following five parameters that define the file search criteria.

1. NO_OF_DAYS: Defines the maximum duration that a file can exist on the server and still be categorized as new
2. DAYS: Is the duration for which the file will be searched
3. INTERVAL: Is the sleep time in hours when the File-Watcher stays dormant
4. FNAME: Is the file name to be searched
5. PATH: Is the location of the file

Figure 1. File-Watcher Parameters

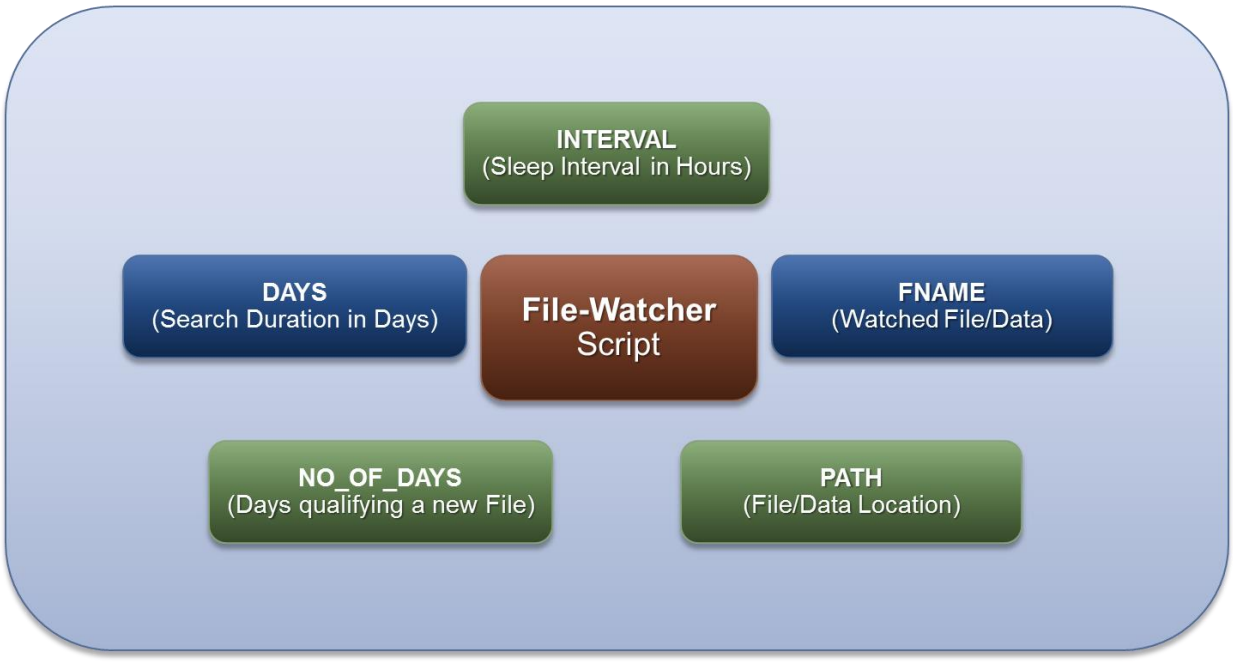


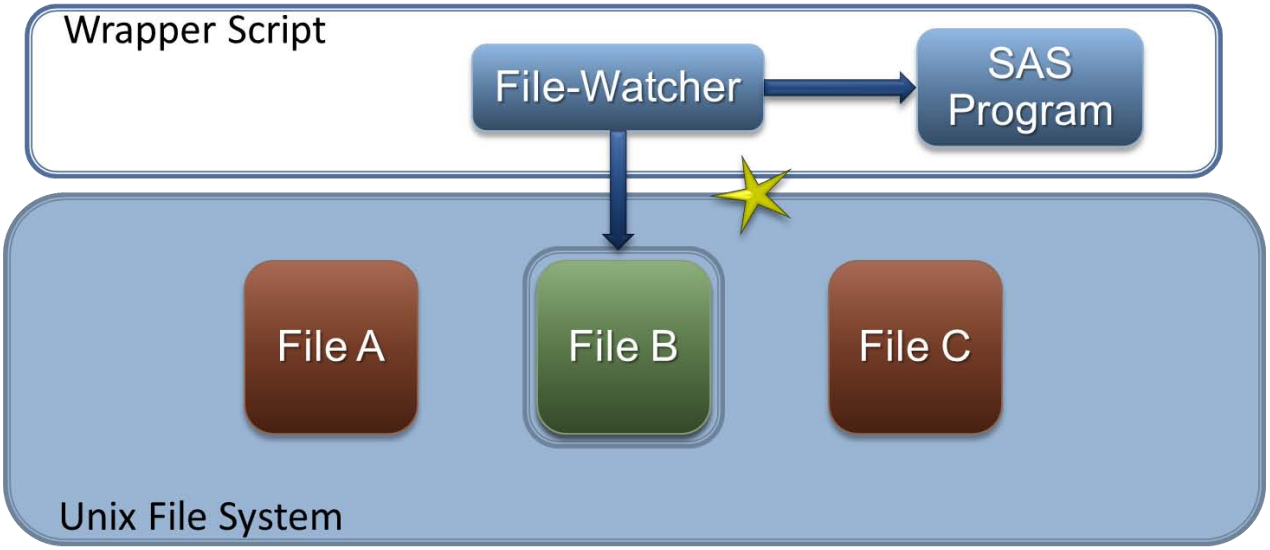
Table 1. File-Watcher Full Code

```
#!/bin/ksh
################################################################################
## UNIX File-watcher script
################################################################################


##########################################
## Check the number of parameters passed #

if [[ $# -lt 3 ]]
 then
    echo "Insufficient Number of Parameters passed to the script"
    echo "Usage: fwatch <days to scan> <interval in hours> <file name> <optional:
directory>"
    echo "example:"
    echo "fwatch 1 1 foo.dat /home/user"
    exit 0
 else
    NO_OF_DAYS=2
    days=$1
    interval=$2
    fname=$3
    path=${4-.}
fi

##########################################
## Check if the directory exists #

if [ -d ${path} ]
  then
    echo "Directory Exists ...."
  else
```

```
    echo "Directory does not Exist .... File-watcher Exiting ....."
    mailx -s "Specified Directory does not Exist" -r "SAS-FileWatcher@XYZ.com" $umail
<< EOF
Specified Directory does not Exist ${path}.

Please check and re-run the file-watcher.

*******Message generated by the file-watcher script*******

****This is an automated mail from the SAS UNIX server****

EOF
    exit 4
fi

########################################
## Calculate the sleep time and number of cycles #


x=0
night=$( awk 'BEGIN{ print int ('$interval' * 60 * 60) }' )
cycle=$( awk 'BEGIN{ print int ('$days'/'$interval' * 24) }' )
echo "delay=" $night "second(s)"
echo "cycle=" $cycle
found=0
echo "File-watcher starting....."


########################################
## Check if the file already exists #

if [ -f ${path}/${fname} ]
  then
    echo "File found... Checking to see if it has been recently updated"
  else
    echo "File not present... File-watched will keep looking for the file"
fi

########################################
## Commence looking for file #

while [ $x -lt $cycle ]
do
 if [ -f ${path}/${fname} ]
  then
     file_found=$(find ${path} -name "*$fname*" -type f -mtime -${NO_OF_DAYS} | wc -l
)
    if [[ ${file_found} -gt 0 ]]
      then
          echo "File has been created/updated recently. File-watcher cycled" $x
"times"
          found=1
          mailx -s "File Found: $fname" -r "SAS-FileWatcher@XYZ.com" $umail << EOF

*******Message generated by the file-watcher script*******

****This is an automated mail from the SAS UNIX server****

EOF
          exit 0
      else
          # echo "File Not Found:" $inpfile
          # echo "Attempt:" ${x}  "Entering Sleep Mode"
          sleep $night
```

```
    fi
   else
     #echo "File does not exist at this time"
     sleep $night
 fi
x=$((x+1))
done

if [ $found -le 0 ]
        then
                echo "File NOT found or has not been updated recently. Please check
with the source and re-initialize the File-watcher script "
                echo "File-watcher exiting....."
                mailx -s "File NOT Found: $fname" -r "SAS-FileWatcher@XYZ.com" $umail
<< EOF

*******Message generated by the file-watcher script*******

****This is an automated mail from the SAS UNIX server****

EOF
               exit 4
fi
```

Figure 2. File-Watcher Operation



The script verifies the directory specified and then commences its scan for the file. If the file already exists, File-Watcher displays an appropriate message and then checks the timestamp on the file. If the file is less than 2 days old, the file-watcher returns a positive message and triggers an e-mail to the user.

If the file does not exist, file-watcher sleeps for a predefined time and then resumes the search. 'Interval' and 'Days' parameters are used to calculate the sleep interval for the script.

The 'umail' parameter contains the list of e-mail address where the notification will be sent. This parameter is set in the wrapper script that is discussed in the next section.

The figures below (**Figure 3, 4**) illustrate the e-mail message and the file-watcher log when the file is found. The log also reports the number of cycles that the file-watcher iterated before triggering the subsequent SAS process. This example uses a delay of 7 seconds and runs for a maximum of 30 cycles.

Figure 3. File-Watcher File Found E-mail

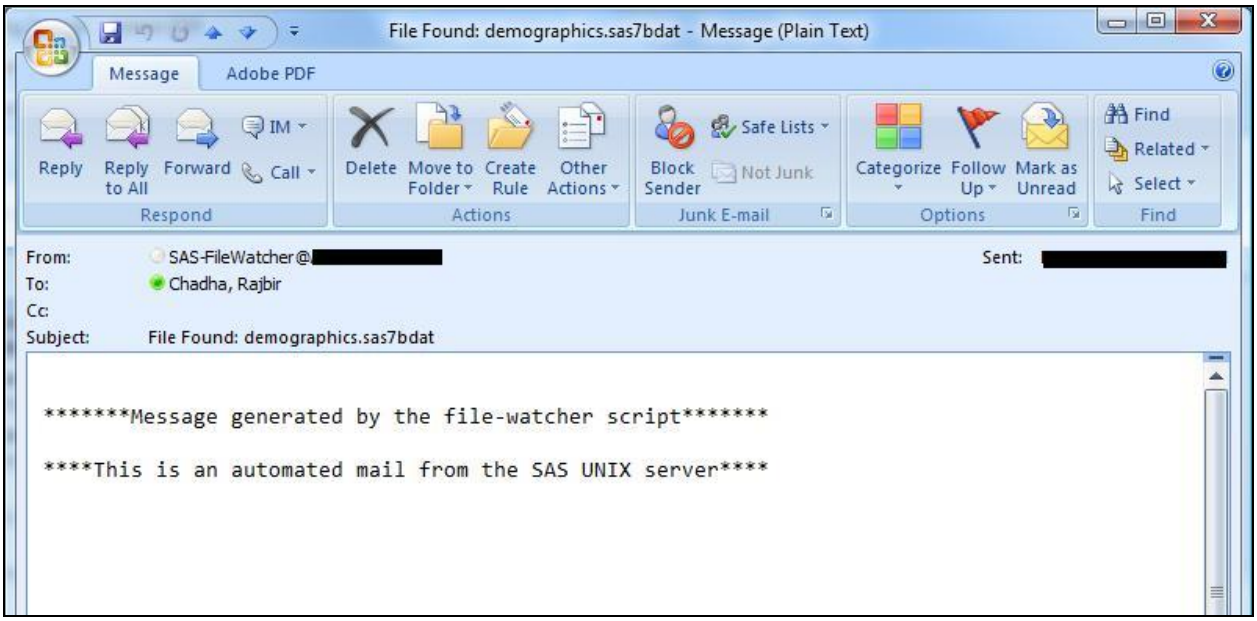
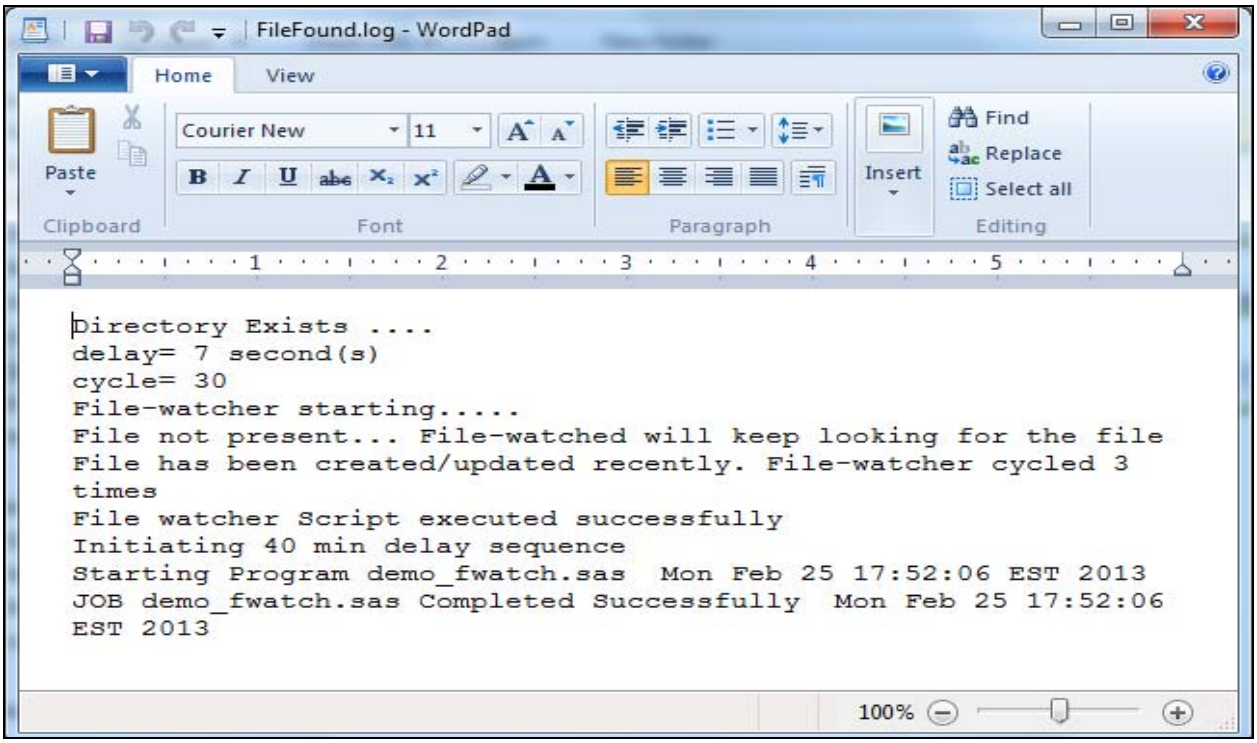


Figure 4. File-Watcher Log when file is found.



The figures below (**Figure 5, 6**) illustrate the e-mail message and the file-watcher log when the file is not found. The file-watcher aborts with an error message, prompting the user to reinitialize the script after checking with the source. This example uses a delay of 7 seconds and runs for a maximum of 30 cycles.

Figure 5. File-Watcher File Not Found E-Mail

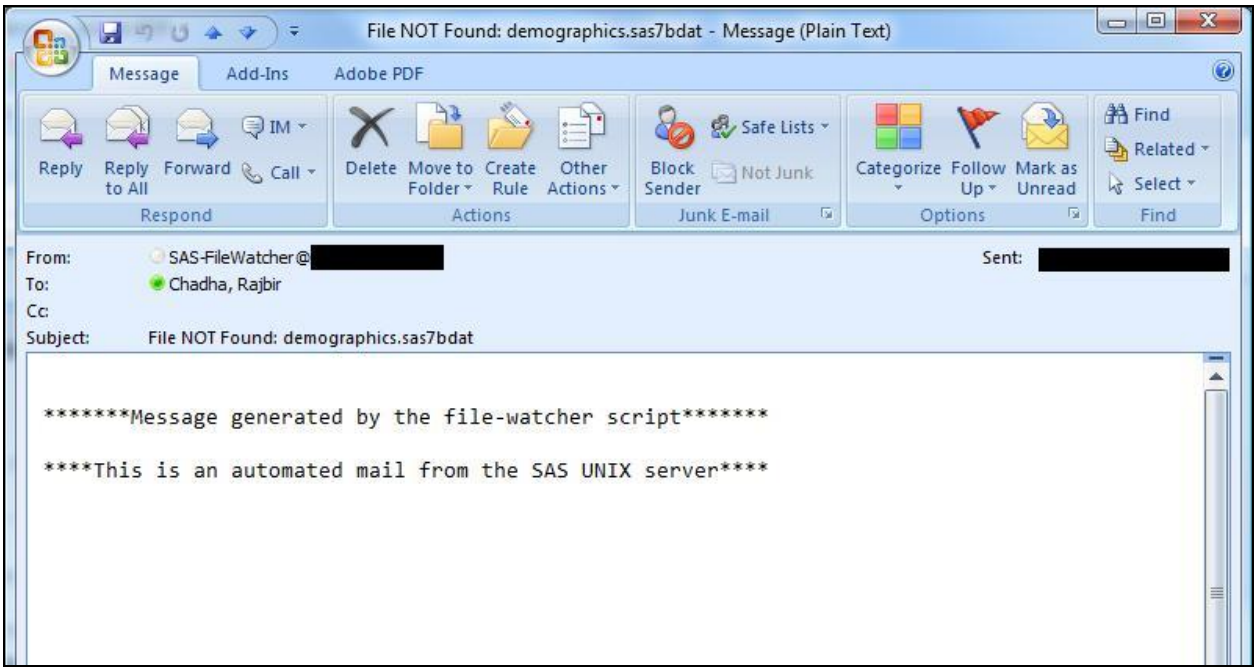
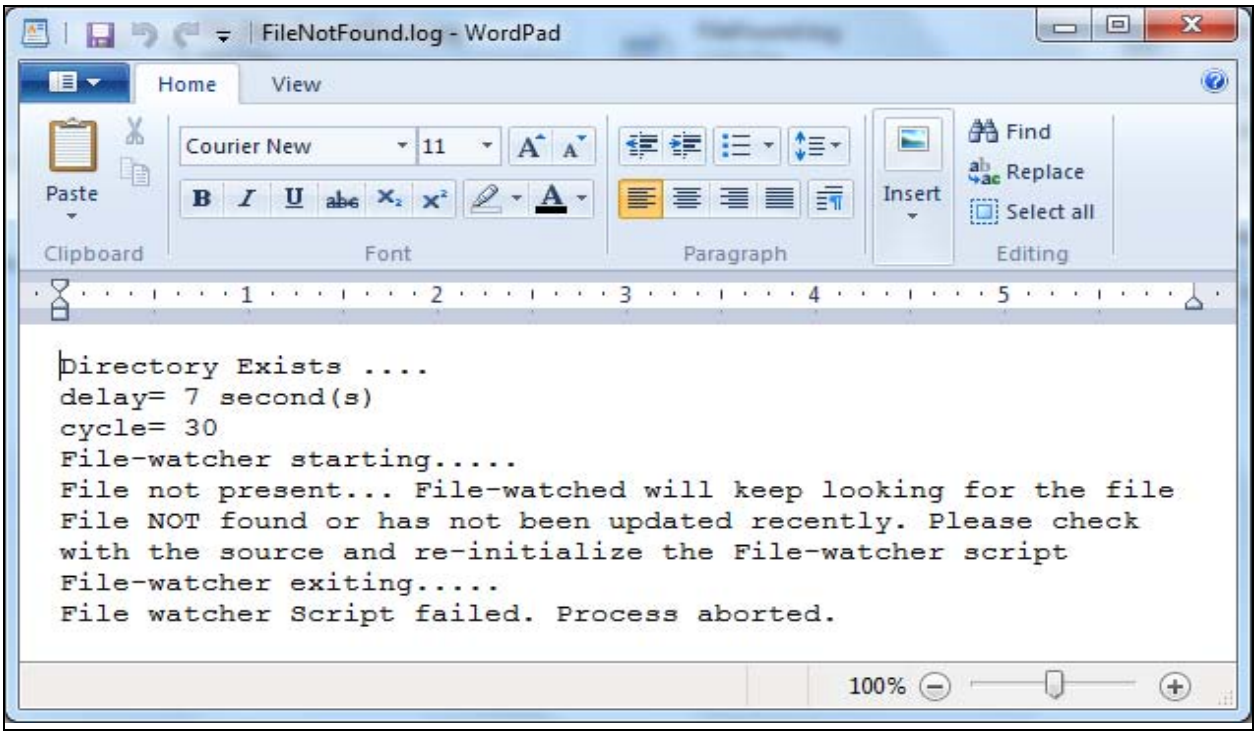


Figure 6. File-Watcher Log when file is Not Found



In order to prevent false triggering of the SAS process, the File-Watcher analyzes the time-stamp on the file if it already exists. If the time-stamp is more than 2 days old (this value can be modified), it does not trigger the file-watcher. The script continues to monitor the file, scanning it based on the specified parameters. If the file gets updated during that time, the script triggers the 'File Found' e-mail and subsequently the SAS process. If the file is not

updated, the file-watcher throws an exception and triggers the 'File Not Found' e-mail. The figures below (**Figure 7, 8**) illustrate the File-Watcher log for these conditions.

Figure 7. File Updated log message.

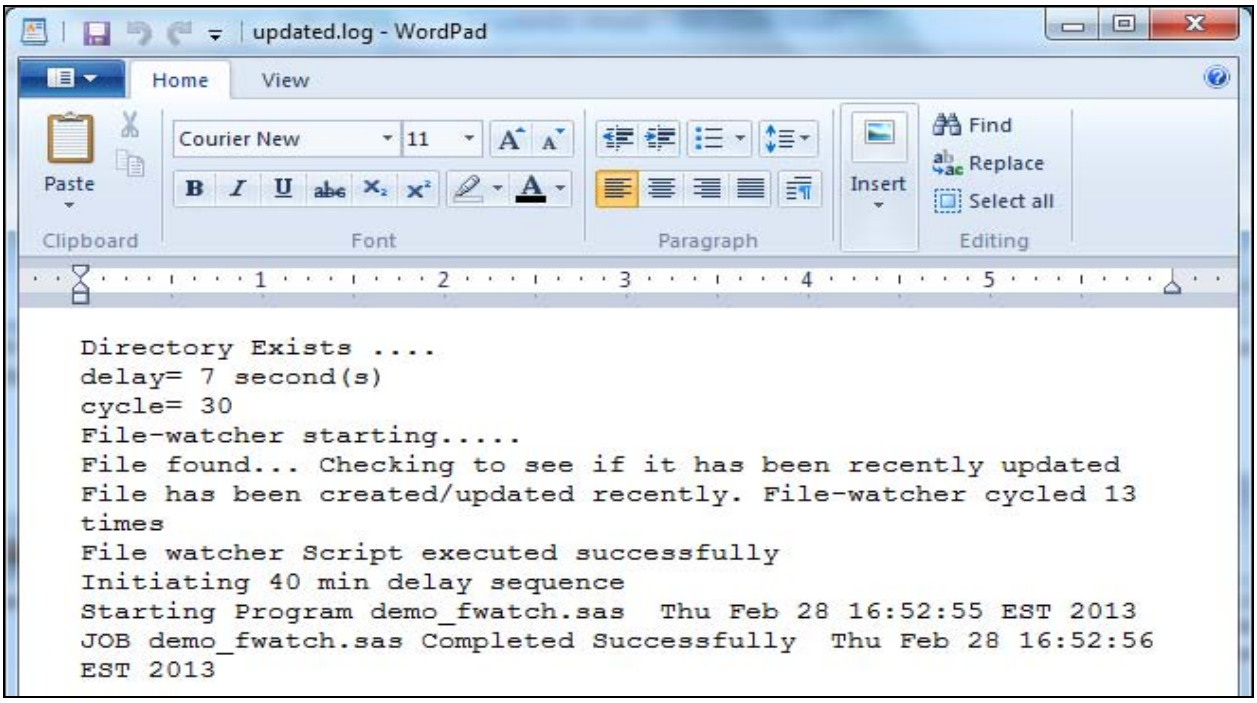
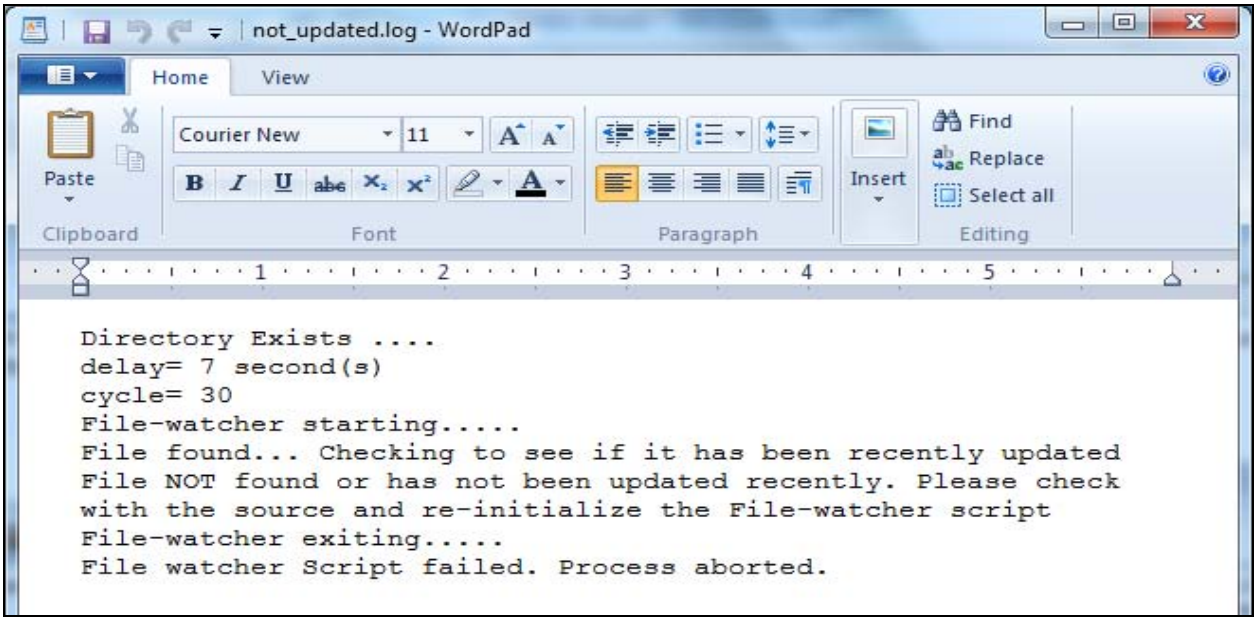


Figure 8. File not updated log message



WRAPPER SCRIPT

The Wrapper script, as the name suggests, simply "wraps" around an existing script, in this case, the file-watcher. It is used to shield the users from a more complicated script and to prevent tampering of the inner script.

The wrapper contains user defined parameters, required by the file-watcher and is scheduled using the CRON scheduler in UNIX. Users can modify the wrapper to search for different files and to execute various SAS programs once those files are found.

The Wrapper script parameters are listed below:

1. UMAIL: Is the destination e-mail address.
2. DD: Defines the number of days to search for the file.
3. INT: Is the time in hours that the script sleeps between searches.
4. DSET: Is the file/dataset being watched.
5. DLOC: Defines the location of the file/dataset being watched.
6. PGM_NAME: Is the name of the SAS program to be triggered after file is found.
7. PGM_PATH: Is the location of the SAS program.
8. LOG_PATH: Is the location of the Log file.

Figure 9. Wrapper Script Parameters

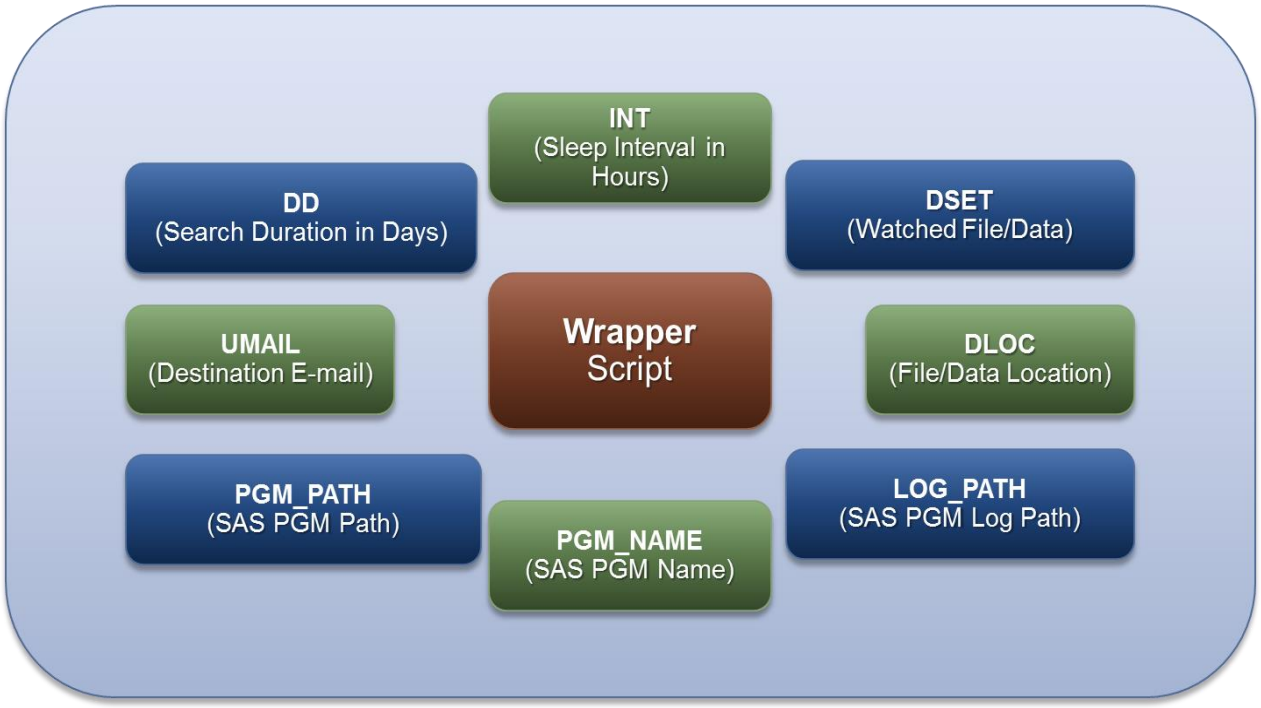


Table 2. Wrapper Script Sample

```
#!/bin/ksh
#############################################################################
## UNIX File-watcher wrapper script
#############################################################################


#*** Crontab setup Instructions ***
#** At the command prompt type 'crontab -e' **
#** Type the following statement and save:
# 30 07 * * 2 /sas/apps/sas_prod/user/demo_user/wrapper_fwatch.sh >
```

```
/sas/apps/sas_prod/user/demo_user/wrapper_fwatch.log 2>&1


#********* Parameters *********
export umail="john.doe@XYZ.com"
#*Enter the number of days the script needs to run*
dd=4
#*Enter the interval (in hours), after which the script should search for the
file*
int=6
#*Enter the file name*
dset=sas_data.sas7bdat
#*Enter the file location*
dloc=/sas/datamart_data/production/sas
#*Enter the User SAS Program details under 'SAS job 1' below*

#******* File-watcher Script - DO NOT MODIFY THIS SECTION *******
/sas/bin/fwatch $dd $int $dset $dloc
RC=$?
 if [[ $RC -ne 0 ]] ; then
     print "File-watcher Script failed. Process aborted."
     exit 1;
 fi
print "File-watcher Script executed successfully"
print "Initiating 40 min delay sequence"
sleep 2400
#******* File-watcher Script - Do NOT MODIFY THIS SECTION *******

#*** SAS job 1 - SAS program name and location ***
pgm_name=user_program;
pgm_path=/sas/apps/sas_prod/user/demo_user/;
log_path=/sas/apps/sas_prod/user/demo_user/;

print "Starting Program ${pgm_name}.sas " `date`
/sas/sas92/SASFoundation/9.2/sas ${pgm_path}/${pgm_name}.sas -log
${log_path}/${pgm_name}.log -print ${log_path}/${pgm_name}.lst
RC=$?
if [[ $RC -ge 2 ]] ; then
   print "FATAL ERROR JOB ${pgm_name}.sas Failed, check log file
${pgm_name}.log "`date`
   print "Return Code: ${RC}"
   exit 1;
else
   print "JOB ${pgm_name}.sas Completed Successfully " `date`
fi
```

The wrapper script has a 40 minute delay sequence built in to allow FTP or QC processes to complete before executing user programs. This is entirely optional and may be removed depending upon your site. The wrapper script can contain multiple SAS programs which can either be run in parallel or in sequence.

ERROR HANDLING AND EMAIL NOTIFICATION

The wrapper script is used to call user's SAS program in batch mode. The script also checks the SAS program for errors based on the return code (RC). E-mails alerts may be sent out to users informing them about the status of the SAS program.

The figure below (**Figure 10**) illustrates the mechanism used to check the Return-Code. UNIX return code can be obtained by the command '$?' in Korn shell. This value is stored in a variable 'RC' for comparison. Return codes

range from 0-256, where 0 indicates a successful operation. SAS typically returns 0 for success, 1 for warnings and 2 for errors. You can influence the exit status by using the ABORT statement. ABORT statement lets you assign the exit status from the range 0-256. When using the ERRORABEND SAS system option, the return code is set to 5 for errors.

Figure 10. Error Handling and Return Code Check

```
RC=$?
if [[ $RC -ge 2 ]] ; then
   print "FATAL ERROR JOB ${pgm_name}.sas Failed, check log file
${pgm_name}.log "`date`
   print "Return Code: ${RC}"
   exit 1;
else
   print "JOB ${pgm_name}.sas Completed Successfully " `date`
fi
```

You can use the code in (**Figure 11, 12**) to add e-mail notification for the SAS program. The mailx command is used to read incoming e-mail interactively. It can also be used to send e-mail messages to the specified recipients. The '-s' flag is used to specify a subject for the e-mail. If the subject includes spaces or special characters, it should be enclosed in quotation marks. The '-r' flag is used to specify the return address. This address appears in the FROM field on the e-mail at the receiver's end. The recipient's e-mail address is also supplied to the mailx command.

The uuencode command converts binary file to ASCII data which can be included in the email.
The uudecode command may be used to convert ASCII data back into binary form.

The uuencode command takes the file name on the standard input and produces an encoded file at the standard output. The encoding uses printable ASCII characters.

Figure 11. E-Mail Notification Code

```
mailx -s "SAS Program Completed: ${pgm_name}.sas " -r "SAS-
FileWatcher@XYZ.com" $umail << EOT
****This is an automated mail from the SAS UNIX server****
EOT
```

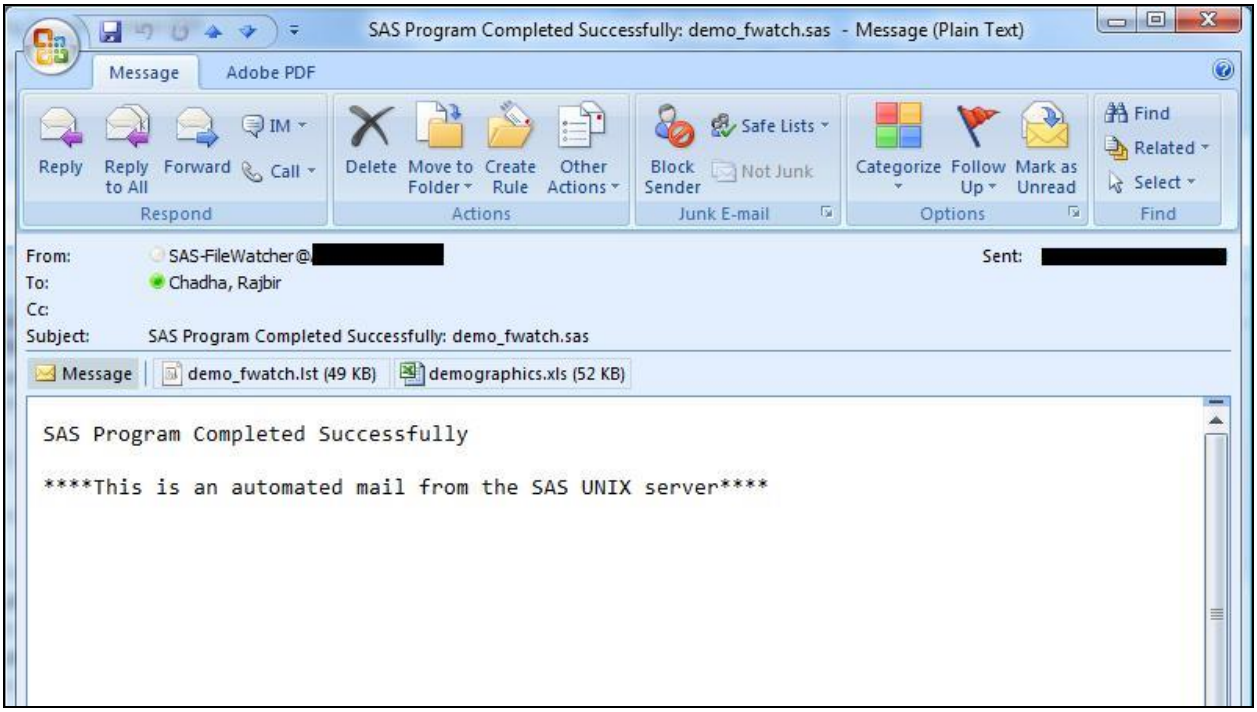
Figure 12. E-Mail Notification Code with Attachments

```
uuencode /sas/apps/sas_prod/user/demo_user/demographics.xls demographics.xls
> attachment
uuencode ${log_path}/${pgm_name}.lst ${pgm_name}.lst >> attachment

mailx -s "SAS Program Completed: ${pgm_name}.sas " -r "SAS-
FileWatcher@XYZ.com" $umail< attachment
```

You figure below (**Figure 13**) illustrates a sample e-mail that contains attachments. This e-mail is generated by the mailx command. The uuencode command is used to encode the excel file 'demographics.xls', and directed to an attachment file. The list file generated by the SAS program is also encoded using uuencode and included in the same attachment file.

The mailx command accepts the encoded input from the uuencode command and attaches them to the message. The recipient can open the attachments from the e-mail like any other e-mail attachment. This is really useful when sharing reports or analysis from SAS programs with other users.

Figure 13. Sample Notification E-Mail with Attachments



CRONTAB UTILITY

CRON is a UNIX based scheduler that enables users to schedule processes and run them at a pre-set time and date. CRONTAB is a file used to maintain jobs in CRON. Each line in the CRONTAB file represents a CRON job.

Note: The 'EDITOR' environment variable in UNIX must be set in order to access the CRONTAB file. It can be set using the following command in ksh –

```
export EDITOR = vi
```

CRONTAB COMMANDS

CRONTAB can be accessed from the UNIX command line using some basic commands. It typically uses vi-editor commands to edit/modify the scheduled jobs. Here are some of the CRONTAB commands:

Figure 14. CRONTAB Commands

crontab –e	Edit your crontab file, or create one if it doesn't already exist.
crontab –l	Display your crontab file.
crontab –r	Remove your crontab file.
crontab –v	Display the last time you edited your crontab file. (This option is only available on a few systems.)

CRONTAB FILE AND SCHEDULING DEFINITIONS

CRONTAB file has five fields for specifying day, date and time followed by the command to be run at that interval. There are also referred to as the CRON expression.

Figure 15. CRONTAB Command Syntax

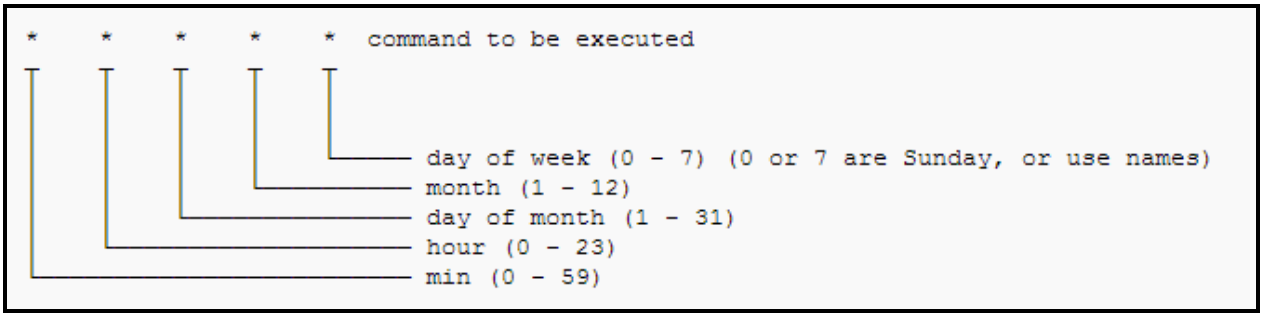


Table 3. CRON Expression

Entry	Description	Equivalent To
@yearly (or @annually)	Run once a year, midnight, Jan. 1st	0 0 1 1 *
@monthly	Run once a month, midnight, first of month	0 0 1 * *
@weekly	Run once a week, midnight on Sunday	0 0 * * 0
@daily	Run once a day, midnight	0 0 * * *
@hourly	Run once an hour, beginning of hour	0 * * * *
@reboot	Run at startup	@reboot

@reboot configures a job to run once when the daemon is started. Since CRON is typically never restarted, this typically corresponds to the machine being booted.

@reboot can be useful if there is a need to start up a server or daemon under a particular user, and the user does not have access to configure init to start the program.

CRON PERMISSIONS

The permissions to create a CRON file or schedule a CRON job are handled by a set of files in UNIX. The following two files manage these permissions:

/etc/cron.allow - If this file exists, then you must be listed therein (your username must be listed) in order to be allowed to use CRON jobs.

/etc/cron.deny - If the cron.allow file does not exist but the /etc/cron.deny file does exist, then you must not be listed in the /etc/cron.deny file in order to use CRON jobs.

Please note that if neither of these files exists, then depending on site-dependent configuration parameters, only the super user or administrator will be allowed to use CRON jobs, or all users will be able to use CRON jobs.

CONCLUSION

The file-watcher tool reduces wait times and effort on part of the users by automating the program triggers, allowing them to focus on other pressing tasks. The code is very flexible and can be modified to fit any UNIX SAS environment. E-mail triggers can be added to the script to notify users once their programs have executed and even include the log file or list report in the e-mail. Multiple programs can be triggered either simultaneously or in a sequence, without any manual intervention, thus saving time and other valuable resources.

REFERENCES

1. "CRONTAB Command". © Copyright IBM Corporation 2010, 2012. Available at http://pic.dhe.ibm.com/infocenter/aix/v7r1/index.jsp?topic=%2Fcom.ibm.aix.cmds%2Fdoc%2Faixcmds1%2Fcrontab.htm
2. "CRON". Wikipedia®, the free encyclopedia. Wikimedia Foundation, Inc. (February, 2013). Available at http://en.wikipedia.org/wiki/Cron
3. Paida, Vinodh and Pandya, Niraj J. 2011. Proceedings of the PharmaSUG 2011 Conference, Nashville, TN (May, 2011). "Let the system do the work! Automate your SAS code execution on UNIX and Windows platforms". Available at http://www.pharmasug.org/proceedings/2011/AD/PharmaSUG-2011-AD11.pdf

ACKNOWLEDGMENTS

I would like to thank my friends and colleagues at Cognizant Technology Solutions, for motivating me to write this paper, documenting my work on SAS and also inspiring me and to present at the SAS conference.

I would also like to mention the following author whose papers have been referenced here - Vinodh Paida and Niraj J. Pandya.

CONTACT INFORMATION

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

Rajbir Chadha
Cognizant Technology Solutions
Wilmington, DE
Phone: 1 (302) 290-6130
E-mail: Rajbir.Chadha@Cognizant.com / Rajbir.Chadha@Gmail.com