Paper 167-30

Using Perl As A Front-End For SAS® Database Updates

John Shingler, CSIS Project, Penn State University, Pennsylvania
Mollie Van Loon, CSIS Project, Penn State University, Pennsylvania
Kenneth Rosenberry, Center for Academic Computing, Penn State University

ABSTRACT

This paper presents an example of using Perl as a front-end "wrapper" for SAS code designed to process newly received data and add it to an established SAS database. This database is updated frequently, usually on a weekly basis. Cases can arrive in various stages of completeness. New information for existing cases can arrive at any time. The use of Perl as a wrapper allows for the manipulation of the various files used in the database updating process. Further, because the database structure and variables are often revised or altered, it is important to be able to run database updates under routine conditions without actually writing the new data to the database. Using Perl as a front-end provides a powerful wrapper for efficiently running such "Test" updates before going live with any programming revisions.

INTRODUCTION

The Consumer Services Information System (CSIS) is a SAS database which collects, error-checks, and stores information on consumer contacts with the Pennsylvania Public Utility Commission. Thousands of new cases are added to the system on a weekly basis, while thousands more are updated. Data files containing these new data are automatically transferred to the CSIS on a weekly basis from the case-intake system. Cases can arrive in various stages of completion and new or updated pieces of information for existing cases can arrive at any time. Every case must be error-checked before being added to the database. SAS code is used to run error checks on all newly received data strings and to then add new cases or update existing cases in the database. This database is then used to run reports for case tracking, performance evaluation, and policy making on the state-level. All of the database updating takes place on a UNIX server, while reports are usually run from a personal computer. (Note that the main body of SAS code is named FIDO for File Integration and Data Organization. The name "FIDO" will appear often in the sample code presented later in this paper.)

Because the utility regulatory environment is constantly changing, the system must be as flexible as possible. New variables are added on a regular basis and existing variables are often modified to capture new aspects of the regulatory process or to answer new questions that arise among regulators with political and policy changes. Each time new variables are added or existing variables are modified, there is the opportunity for error to be introduced into the SAS code. Occasionally, it is also necessary to make major changes to the format of the database and individual data strings. These conditions make it important to have a method of running the update and error-checking SAS program with real data without making any actual changes to the existing database. Such "Test" updates will reveal any errors that have been introduced into the code without sacrificing the integrity of the database. If an Update is run in the Test mode without producing any errors, it is safe to run the Update in the "Live" mode.

It is important that such Test updates are run with actual data and in conditions as close as possible to a real update. Perl provides a method for doing this in an efficient and flexible manner. Perl not only manages all the files and directories involved in processing routine updates, but it also oversees the Test updates. It provides flexible options for modifying and debugging programs as needed. In a Test Update, Perl allows everything to run as it normally would run but does not execute the step of writing the new data to the main SAS database. Instead, it reports on the Update as if it were a real update, producing information on how many and what kinds of errors were found in the data, and then sets everything back to the way it was before the update was run. After examining the output from the error report, it is possible to determine if more adjustments must be made to the revised programming or if the program is ready to process new data "for real."

PERL FUNCTIONS

In summary, there are four primary functions served by the use of a Perl wrapper:

1. The Perl wrapper checks to see if the prior database update ran without any errors and if all the necessary files are present and in the proper condition.

2. The Perl wrapper checks newly arrived data files to make sure that all data is in its proper format. If data is not in the proper format, or has been transferred improperly, it will not run the update. In this way, flawed data is not added to the database.

3. The Perl wrapper automatically runs back-ups after each database update is run. Multiple back-ups allow the system to be re-set to a past date if it is discovered that a systematic error exists in the data that is not detected in other error-checking procedures.

4. The Perl wrapper allows for the database updates to be run in a Test mode. As discussed above, this allows the programmer to test many programming changes without actually writing the processed data to the database. Thus, if an error is detected in the programming, the database has not been compromised.

SAMPLE CODE

Examples for each of these four functions are presented below:

Sample Code 1: Example code to illustrate using a Perl wrapper around a SAS program. This example will check for the existence of a file before calling SAS.

Note: A more complicated Perl program might check to make sure that many files exist or that some files do NOT exist. Further, we could also check the contents of files to verify that they contain what we expect them to. We don't show such code here. But, the production version of our runfido.pl program available as a handout does show such examples.

Some Perl information for this example:

- '#' is the comment character. On each line, everything after a '#' is ignored.
- '-e' is a Perl file test operator which returns true if a file exists, false otherwise.
- 'die' is a Perl statement that prints a message and exits the Perl program.
- Backticks (`) are one way to run external programs (like SAS).

```
#!/usr/local/Perl5.005_03/bin/Perl -w

#-----------------------------------------------------------------
# Program   : SampleCode1.pl
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# if the 'runfido.OK' file does NOT exist,
# print an error message and exit the program.
#-----------------------------------------------------------------
if (not -e 'runfido.OK')
   {
   die "Error: runfido.OK file not found.  Stopped";
   }

   #-----------------------------------------------------------------
   # We got this far, so our 'runfido.OK' file must exist.
   # Print the results of running the program through SAS.
   #-----------------------------------------------------------------
   print `sas fido.sas`;
```

Sample Code 2: Add code to verify the existence of a data file and code confirming that the data file has the correct logical record length (LRECL).

Some additional Perl information for this example:

- 'maxwidth' is an external program that returns the length of the longest line in a file

• 'newdata.dat' is the file containing newly arrived information to be added to the database.

```
#!/usr/local/Perl5.005_03/bin/Perl -w

#-----------------------------------------------------------------
# Program   : SampleCode2.pl
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# if the 'runfido.OK' file does NOT exist,
# print an error message and exit the program.
#-----------------------------------------------------------------
if (not -e 'runfido.OK')
   {
   die "Error: runfido.OK file not found.  Stopped";
   }

   # -----------------------------------------------------------------
   # if our 'newdata.dat' file does NOT exist,
   # print an error message and exit the program.
   #-----------------------------------------------------------------
   if (not -e 'newdata.dat')
   {
   die "Error: newdata.dat file not found.  Stopped";
   }

   #-----------------------------------------------------------------
   # If the longest line of the 'newdata.dat' file is not exactly
   # 155 characters, print an error message and exit.
   #-----------------------------------------------------------------
   if (155 != `maxwidth < newdata.dat`)
   {
   die "Error: newdata.dat file has incorrect lrecl.  Stopped";
   }

   #-----------------------------------------------------------------
   # We got this far, so our 'runfido.OK' file must be ok.
   # Run the file through SAS and print the results.
   #-----------------------------------------------------------------
   print `sas fido.sas`;
```

Sample Code 3: Add code to allow us to keep copies of our old fido.sas7bdat files. To make archive versions we simply rename our old master dataset (fido.sas7bdat) file to have the current date appended. We can keep as many archive versions of our dataset as we have space.

Note: Using the date to create archive copies of our master data set assumes that we will never run this program more than once per day. If that were not the case, we could consider a different naming convention for the archived copies of our master file. For example, appending the time of day in addition to the date.

Also Note: Assume that the fido.sas program reads the newdata.dat file and merges it with our master dataset (fido.sas7bdat). Further, the fido.sas program does NOT update the master dataset directly. Rather, it creates a new SAS dataset (fido_new.sas7bdat) which is created from a PROC MERGE of the newdata.dat file and the fido.sas7bdat file.

Thus, after the fido.sas program completes, we have two datasets:

fido.sas7bdat - the old master dataset
fido_new.sas7bdat - the new master dataset

This allows us to create the archive copy of the old master dataset by renaming the old master AFTER the SAS program runs. If the SAS program updated the master dataset directly, we would have to create the backup by copying the old master dataset BEFORE the SAS program runs.

Some additional Perl information for this example:

- 'mv' is an external program that renames a file

```
#!/usr/local/Perl5.005_03/bin/Perl -w

#-------------------------------------------------------------------
# Program   : SampleCode3.pl
#-------------------------------------------------------------------

#-------------------------------------------------------------------
# if the 'runfido.OK' file does NOT exist,
# print an error message and exit the program.
#-------------------------------------------------------------------
if (not -e 'runfido.OK')
   {
   die "Error: runfido.OK file not found.  Stopped";
   }

   #-------------------------------------------------------------------
   # if our 'newdata.dat' file does NOT exist,
   # print an error message and exit the program.
   #-------------------------------------------------------------------
   if (not -e 'newdata.dat')
   {
   die "Error: newdata.dat file not found.  Stopped";
   }

   #-------------------------------------------------------------------
   # If the longest line of the 'newdata.dat' file is not exactly
   # 155 characters, print an error message and exit.
   #-------------------------------------------------------------------
   if (155 != `maxwidth < newdata.dat`)
   {
   die "Error: newdata.dat file has incorrect lrecl.  Stopped";
   }

   #-------------------------------------------------------------------
   # We got this far, so our 'runfido.OK' file must be ok.
   # Run the file through SAS and print the results.
   #-------------------------------------------------------------------
   print `sas fido.sas`;

   #-------------------------------------------------------------------
   # Get today's date as yyyy-mm-dd
   #-------------------------------------------------------------------
   $today = sprintf("%04d-%02d-%02d",
             (localtime)[5]+1900,
             (localtime)[4]+1,
             (localtime)[3]);

   #-------------------------------------------------------------------
   # Rename our old fido dataset with today's date appended
   #-------------------------------------------------------------------
   `mv fido.sas7bdat fido.sas7bdat.$today`;

   #-------------------------------------------------------------------
   # Rename our new fido dataset to be our new master dataset
   #-------------------------------------------------------------------
```

```
`mv fido_new.sas7bdat fido.sas7bdat`;
```

Sample Code 4: Add code to allow us to do a test run of our SAS program.

Recall from Sample Code 3 that the SAS program creates a new master dataset (fido_new.sas7bdat) by merging the existing master dataset (fido.sas7bdat) with a new data file (newdata.dat). The original master dataset is unmodified.

To make a test run, instead of renaming files such that the old master becomes a backup copy, and the new master dataset becomes the current master dataset, we change the way we rename the files after the SAS program runs.

Specifically, the new master data set (fido_new.sas7bdat) is simply renamed to be fidotest.sas7bdat. We can then run whatever tests we find necessary to validate that the SAS program worked correctly.

Finally, we need a way to signal our program that we are running a 'TEST' run, versus a 'LIVE' update. Ideally, we want to read this signal in from a control file, and perhaps even allow our newdata.dat file to have a header line with this flag. But, to keep our example code simple, we hard code a variable into our program with a value to control the program logic.

```
#!/usr/local/Perl5.005_03/bin/Perl -w

#-------------------------------------------------------------------
# Program   : SampleCode4.pl
#
#-------------------------------------------------------------------

#-------------------------------------------------------------------
# Create a PERL variable (runMode) that will control whether we
# are doing a 'TEST" run, or running a 'LIVE' update.
#-------------------------------------------------------------------
$runMode = 'TEST';  # Change this to 'LIVE' if you dare.

#-------------------------------------------------------------------
# if the 'runfido.OK' file does NOT exist,
# print an error message and exit the program.
#-------------------------------------------------------------------
if (not -e 'runfido.OK')
{
die "Error: runfido.OK file not found.  Stopped";
}

#-------------------------------------------------------------------
# if our 'newdata.dat' file does NOT exist,
# print an error message and exit the program.
#-------------------------------------------------------------------
if (not -e 'newdata.dat')
{
die "Error: newdata.dat file not found.  Stopped";
  }

#-------------------------------------------------------------------
# If the longest line of the 'newdata.dat' file is not exactly
# 155 characters, print an error message and exit.
#-------------------------------------------------------------------
if (155 != `maxwidth < newdata.dat`)
{
die "Error: newdata.dat file has incorrect lrecl.  Stopped";
}

#-------------------------------------------------------------------
```

```
# We got this far, so our 'runfido.OK' file must be ok.
# Run the file through SAS and print the results.
#-----------------------------------------------------------------
print `sas fido.sas`;

#-----------------------------------------------------------------
# Get today's date as yyyy-mm-dd
#-----------------------------------------------------------------
$today = sprintf("%04d-%02d-%02d",
          (localtime)[5]+1900,
          (localtime)[4]+1,
          (localtime)[3]);

if ($runMode eq "LIVE")
{
#-----------------------------------------------------------------
# Same rename commands as explained in Sample Code 3.
#-----------------------------------------------------------------
`mv fido.sas7bdat fido.sas7bdat.$today`;
`mv fido_new.sas7bdat fido.sas7bdat`;
}
else # We must be doing a TEST run.
{
#-----------------------------------------------------------------
# Rename our new fido dataset to indicate it's a test dataset.
# Do NOT rename our old (unmodified) master dataset.
#-----------------------------------------------------------------
`mv fido_new.sas7bdat fido_test.sas7bdat`;
}
```

DISCUSSION

The CSIS operates on an UNIX server but the concept of using a front-end wrapper to process SAS programs and databases can be implemented on any platform. Every operating system has its own languages that will allow programmers to manipulate files and perform the functions described above.

Because SAS allows external system calls, it is possible to do all of the above in SAS without the use of a wrapper. However, we chose to use PERL in order to play to the strengths of both languages. While SAS is a powerful tool for processing and managing large data sets, Perl's strength is in file manipulation. Since the programming undergoes frequent updates and revisions, using PERL allows us to determine the location of an error while the SAS code remains intact. We believe that using SAS and Perl in this combination is more efficient that using just SAS.

In fact, it is not necessary to use Perl on UNIX. Other programs that could perform the same functions include Python, Ruby and PHP. We use Perl because that is the program we are most familiar with. Additionally, Perl has a good track record for working on UNIX platforms—we have not experienced any problems that would make us want to use another program.

ADVANTAGES

Advantages of using Perl as a wrapper include:

1. Many of the steps, such as back-ups, are automated so that the programmer has less to do.
2. Updates and revisions can be made quickly.
3. Files are efficiently manipulated.
4. Updates do not run if data is not in proper format or necessary files are not present or do not contain appropriate information.
5. Tests may be run to see if programming updates have been made correctly, without impacting on the database.
6. Errors and location of errors are easily identified and corrected without compromising SAS code.

CONCLUSION

Perl provides a strong complement to SAS to maximize the flexibility of a frequently updated information system. SAS provides powerful tools for creating and updating the database and for running frequent analytical reports. Perl provides a shell that efficiently manipulates the various files utilized in updating and creating the database, and also allows for Test updates when changes are made to the data content or structure.

REFERENCES

Siever, Ellen, Stephen Spainhour and Nathan Patwardhan (1999), PERL In A Nutshell, O'Reilly & Associates.

CONTACT INFORMATION

John Shingler
Penn State University
Room 5 Armsby Building
University Park, PA 16802
Jxs10@psu.edu

Mollie Van Loon
Penn State University
Room 5, Armsby Building
University Park, PA 16802
Mev2@psu.edu