Paper SAS258-2014

Useful Tips When Deploying SAS® Code in a Production Environment

Elena Shtern, SAS Institute Inc., Arlington, VA

ABSTRACT

When deploying SAS® code into a production environment, a programmer should ensure that the code satisfies the following key criteria:

1. The code runs without errors.
2. The code performs operations consistent with the agreed upon business logic.
3. The code is not dependent on manual human intervention.
4. The code performs necessary checks to provide sufficient quality control of the deployment process.

Base SAS® programming offers a wide range of techniques to support the last two aforementioned criteria. The presentation demonstrates the use of SAS® macro variables in combination with simple macro programs to perform a number of routine automated tasks that are often part of the production-ready code. Some of the examples to be demonstrated include the following topics:

- How to check that required key parameters for successful program run are populated in the parameters file.
- How to automatically copy the content of the permanent folder to the newly created backup folder.
- How to automatically update the log file with new run information.
- How to check whether dataset already exists in the library.

INTRODUCTION

This presentation is a collection of snippets of the code that SAS® programmers can use when coding routine business processes. The scripts leverage SAS® procedures, data steps, and SAS® macro language.

DRIVER PROGRAM

There are many best practices that a code developer would benefit from in his or her work. One of the simplest ones is setting up a driver program to manage code complexity and to organize code in an easy-to-follow and structured way. Driver programs can be thought of as central hubs for setting up macro parameters, performing completeness checks, executing programs in a particular order, updating a log file with new information, and other relevant tasks.

The following are parts of the driver program that contain coding techniques that may be used for production code or in support of a repetitive business process.

CLEANING UP THE WORK LIBRARY

If a program is not run in the batch mode, it is recommended to clean the SAS work library to avoid the risk of accidentally using older versions of datasets prior to the start of a new run. This is easily accomplished through the use of the DATASETS procedure.

```
proc datasets lib=work kill;
run;
```

The KILL option removes all datasets in the library.

SETTING UP PARAMETERS FOR SUBSEQUENT USE

A driver program is the best place to set up macro parameters for subsequent use in the process. The snippet of the code below imports a parameter file that is based on Excel and uses PROC SQL to create macro variables. The file used for this demo has two parameters: Input File1 and Report Month.

```
/** IMPORT EXCEL-BASED PARAMETERS FILE */

proc import datafile="&path\Parameters_Demo.xlsx"
   out=Parameters
   dbms=xlsx replace;
run;


/** SET UP PARAMETERS VALUES TO BE USED THROUGHOUT THE PROCESS **/

proc sql noprint;
 select value into: InputFile1
 from Parameters
 where Parameter="Input File1";

 select value into: ReportMonth
 from Parameters
 where Parameter="Report Month";
quit;
```

CHECKING COMPLETENESS OF THE PARAMETERS FILE

An important quality control check is to ensure that all parameters required for successful processing have valid non-missing values. There are different ways to accomplish this. One approach is through the use of a macro program shown below.

```
/** MACRO PROGRAM CHECKS WHETHER ALL KEY PARAMETERS HAVE NON-MISSING VALUES **/

%macro check_parameters;

%if (%str(&InputFile1) eq %str()) %then %do;
    %put ERROR: Missing Input information for Input File1. The program stopped
running.;
    %abort cancel;
%end;


%if (%str(&ReportMonth) eq %str()) %then %do;
    %put ERROR: Missing Input information for Report Month. The program stopped
running.;
    %abort cancel;
%end;
%mend;
%check_parameters;
```

The statement %ABORT CANCEL stops submitted SAS program and generates error message in the SAS log.

CREATING A BACKUP FOLDER

It is important to create a backup folder of key datasets before making changes to them. This makes it easy to restore the prior version if needed and eliminates dependency on IT personnel. The code below creates a subfolder with the date/time of a new run and copies key datasets to this folder. The code works only in a Windows environment.

```
/** SET UP MACRO VARIABLES TO STORE DATE AND TIME OF EACH RUN **/

%let
timenow=%sysfunc(compress(%sysfunc(scan(%sysfunc(time(),time.),1,':'))%sysfunc(scan(
%sysfunc(time(),time.),2,':'))));

 %let datenow=%sysfunc(date(), date7.);

/** SET UP A MACRO VARIABLE TO DEFINE PATH TO FOLDERS **/

 %let path=C:\SGUF2014;

 /** ASSIGN SAS LIBRARIES **/

 libname perm   "&path\Permanent Data";
 libname backup "&path\Backup Data\Permanent_Folder_&DATENOW._&TIMENOW";

 /** COPY THE CONTENT OF THE PERMANENT FOLDER TO THE BACKUP FOLDER **/

 x md "&path\Backup Data\Permanent_Folder_&DATENOW._&TIMENOW.\";


 proc copy in=perm out=backup;
  run;
```

INVOKING OTHER PROGRAMS

A driver program is often used for invoking other programs that support repeatable business processes. The code below shows how easily programs can be called from the driver program through the use of the %INCLUDE statement.

```
%include "&path\ImportDemographics.sas";
```

UPDATING THE LOG FILE

To keep track of execution runs, it is prudent to maintain a log with key information relevant to each run. The example below updates the log file with information about the date and time of the run, report month, and input file.

```
/** SET UP MACRO VARIABLES TO STORE DATE AND TIME OF EACH RUN **/

%let
timenow=%sysfunc(compress(%sysfunc(scan(%sysfunc(time(),time.),1,':'))%sysfunc(scan
(%sysfunc(time(),time.),2,':'))));

%let datenow=%sysfunc(date(), date7.);

/** ADD EXECUTION RUN INFO TO THE EXECUTION LOG DATASET **/

data new_run_info;
 format Date $9.
           Time $4.
           inputData $30.
           reportMonth $7.;
 Date="&datenow";
 Time="&timenow";
 inputData="&InputFile1";
 reportMonth="&ReportMonth";
run;
```

```
proc append data=new_run_info base=perm.Execution_Log; run;

proc delete data=new_run_info; run;
```

The added record looks as follows in the dataset EXECUTION_LOG:

Obs	Date	Time	inputData	report Month
1	11NOV13	1154	Demo_InputFile1.xls	MAR2013

Display 1: Screenshot of the added record

CHECK TO SEE IF A SAS DATASET EXISTS

If a process is dependent on the presence of a particular dataset, it is prudent to have code that checks whether such a dataset exists and generates an error message when the dataset is not found. The code below checks whether dataset CITIYR exists in the SAS library SASHELP.

```
/** CHECK IF A DATASET CITIYR EXISTS IN THE SASHELP LIBRARY **/

%macro check_db;

%if %sysfunc(exist(SASHELP.Citiyr)) %then
  %goto exit;
%else
 %do;
  %put ERROR: Dataset CITIYR is not found. The program stopped running. ;
  %abort cancel;
  %end;

%exit:
%mend check_db;
%check_db;
```

The statement %ABORT CANCEL stops submitted SAS program and generates error message in the SAS log.

HOW TO AVOID APPENDING DUPLICATE RECORDS TO A MASTER DATASET

Business processes often involve updating a master dataset with new or updated records. To avoid appending duplicate records, especially when no time stamp information is present to differentiate between the older and newer records, one can leverage the code below to remove records in the master dataset that are being updated with new information. The code below uses the dataset CITIYR in the SAS library SASHELP. The dataset CITIYR has 10 records with yearly data from 1980 through 1989.

For demo purposes, the dataset CITIYR_UPD is created with two records, one for year 1989 and one for year 1990.

```
/** CREATE A MOCK UP INTERIM DATASET TO BE ADDED TO A MASTER DATASET **/

data Citiyr_Upd;
 set sashelp.Citiyr;
 if _n_<=2;
 if year(date)=1980 then date='01Jan1989'd; else
                         date='01Jan1990'd;
run;

proc sql noprint;
 select distinct year(Date) into: Year separated by ','
 from Citiyr_Upd;
quit;
```

The macro program below updates the master dataset CITIYR with updated information for 1989, since the record for

1989 already exists in dataset CITIYR, and adds a new record for the year 1990. Once the macro program is complete, dataset CITIYR has 11 records.

```
/** FIRST, CHECK IF A MASTER DATASET EXISTS. IF YES, CHECK WHETHER RECORDS FOR  **/
/** THE SAME PRIMARY KEY ALREADY EXIST IN THE MASTER DATASET. IF YES, REPLACE   **/
/** THE OLDER RECORDS WITH NEWER RECORDS.                                       **/
/** IF THIS IS THE FIRST TIME THE RECORDS ARE ADDED TO THE MASTER DATASET,      **/
/** NO CHECKS ARE REQUIRED, AND ALL RECORDS CAN BE APPENDED.                    **/

%macro append_record;
%macro a;
%mend a;

%if %sysfunc(exist(sashelp.Citiyr)) %then
 %do;
    proc sql;
     create table Citiyr_Temp as
      select *
      from sashelp.Citiyr
       where year(Date) notin (&Year);

    proc append data=Citiyr_Upd base=Citiyr_Temp;
     run;

    data Citiyr;
     set Citiyr_Temp;
    run;

    proc delete data=Citiyr_Temp;
     run;
 %end;
%else
 %do;

 data Citiyr;
  set sashelp.Citiyr;
 run;

  proc append data=Citiyr_Upd base=Citiyr;
     run;
  %end;

%mend append_record;
%append_record;
```

CONCLUSION

Base SAS® programming offers a wide range of programming techniques to perform quality control checks of the deployment process and to minimize dependency on the manual input. This presentation focused on sharing simple techniques which programmers can apply in various situations.

REFERENCES

- SAS 9.4 Product Documentation at http://support.sas.com/documentation/94/index.html
- SAS Technical Papers at http://support.sas.com/rnd/

CONTACT INFORMATION

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

Elena Shtern
1530 Wilson Blvd, Suite 800
Arlington, VA 22209
SAS Institute Inc.
Elena.Shtern@sas.com
http://www.sas.com