Paper 58-2018

Dependency Macro: Ensuring data reliability in healthcare analytics

Lissa Bayang, The Permanente Medical Group

Wayne L. Leonetti, The Permanente Medical Group

ABSTRACT

Obtaining the latest data in healthcare analytics is of utmost importance. Providers and management rely on the newest data obtained from medical records to help drive their daily goals and progress. In many healthcare organizations, SAS® programs are run daily to produce the latest data. However, various issues can prevent the upload of the most current data. Some of these issues include delays from the database, high session volumes due to many users accessing the database, and other program failures. It can be problematic for SAS programs to run without any way of checking if the current data set is available to be consumed. This produces unreliable data for stakeholders who need it. To mitigate this issue, a dependency macro is put in place to check if a data set is present with the current date's timestamp. If the available file does not meet the requirement, SAS processing is delayed until a data set with the current date's timestamp is present. In addition to checking the current timestamp, other options are included in this macro to make it more useful such as the ability to check intermittently up to a certain time and the ability to notify the user via email if the SAS program successfully ran or failed. Options in this macro are flexible and easy to adjust which allows it to be used in different situations. This paper is intended for intermediate and advanced SAS users and the program is used in SAS 9.4.

INTRODUCTION

In today's fast-paced work environment, having the latest data available is critical. Healthcare organizations need the newest information available to inform providers and management regarding the current status of the patients' conditions, state of various clinics and facilities, and what gaps need to be addressed to ensure optimum workflows

Since many reports update daily, it is important to ensure that daily reports contain the latest information. However, big organizations like many hospitals and medical facilities, rely on a medical record database that can be prone to delays due to upgrades, outages, high session volumes, and other factors.

Most reports are being produced during a particular time frame in the morning when many routinized programs from various departments are also being run. In addition, there are integrated reports that rely on various data sets and files before the final report is produced. In this situation, there is a need to ensure that the most current file or data set is available so that the data reflects the current status of the clinic or hospital. A way to resolve this issue is to include a macro that checks for the data set with today's timestamp. In addition to checking data sets with this macro, it is flexible enough that it can check for any type of file including logs, other file formats such as Excel, etc. A great advantage of this macro is its flexibility to meet your business needs.

The techniques shown in this paper can be applied to most SAS versions. However, this was tested on SAS 9.4 and SAS Enterprise Guide® 6.1 software using the local server. Intermediate knowledge of using macros is useful in interpreting the code used in this paper.

THE DEPENDENCY MACRO

First, you need to decide what essential macro variables should be included for your routine SAS programs to run. For this example, the main variables include the file name, current timestamp, time limit, delay, and email notification. However, other options are included which may be helpful depending on your needs and setup. This macro has two parts that need to run: the part where all the data steps are set up and the call portion that allows this macro to run.

SETTING UP THE MACRO VARIABLES AND OPTIONS

In the first part of the macro, you set up the file name (FYLNAME), time limit (TLIMIT), and the time delay (DELAY). You can also hard code the rest of the variables: EMAIL, SUBJ, and MESG if you prefer. However, if you plan to use this macro globally in the work place, it is best to leave the last 3 variables blank and modify it when you call your macro:

```
%macro Depend(FylName,tLimit='12:00:00't,Delay=300,
             eMail=,
             Subj=,
             Mesg=);
```

The FYLNAME macro variable corresponds to the data set you want to check. The TLIMIT macro variable, which is in datetime format, is the maximum time of day you want the macro to keep checking until it decides to terminate the program. In this instance, the macro is set up to check until 12:00 PM. If the data set is still not updated by that time, you can put an option to either abend or just run the program with the most current data available. In this example, the macro will abend the processing if today's data set is not available after noon. DELAY is the macro variable where you set up the time in between to check the file. In this example, DELAY has a default of 300 seconds or five minutes. This macro uses seconds as the default unit and can be adjusted to your preferences. For the EMAIL macro variable, you can put your own email address or add another person who needs to know if the program fails. The SUBJ macro variable will populate the subject line of the email and the MESG macro variable will populate the body of the email. If the process fails and the EMAIL variable is left blank in both the macro itself and the call portion, no email will be sent.

The next step in the macro is setting up the variables that drive the process of checking the file and sending notifications to the email recipient. First, you create global and local variables with default settings to detect the file and to notify the user in case of failure:

```
%global Fail;
%local rc fid fidc ModifyDT Done;

%let Done = N;
%let Fail = N;

options nonotes;
```

A global macro variable called FAIL is created to enable programs outside of this macro to recognize it in case it fails. However, if the macro goes through and a new data set is found, it will then continue to run your subsequent program to finish off your process. The local macro variables RC, FID, FIDC, MODIFYDT, and DONE are also created in the first portion of the dependency macro. The macro variables DONE and FAIL default to 'N'. It is also recommended to include an option to suppress excessive note production in the beginning to prevent overloading the log by adding the statement OPTIONS NONOTES.

For this macro to work, you need to set up the following local macro variables referenced above:

- RC: a return code that lets you know if the file needed is present
- FID/FIDC: macro variables that use a return value generated by a macro process that allow you to open and close the file you need during the looping process
- MODIFYDT: the current date you need your file to have, usually today's date
- DONE: a variable that notifies you if the process completed

The local macro variable DONE is used with the global macro FAIL to indicate if the process was successful. This macro will loop through until the needed file is found or the macro exceeds the time limit (TLIMIT) that was set in the beginning. The following syntax shows this process:

```
	%do %until(&Done=Y or &Fail=Y);
		%let rc=%sysfunc(filename(onefile,&fylname));
		%let fid=%sysfunc(fopen(&onefile));

		%if &fid ne 0 %then %do;
			%let ModifyDT=%qsysfunc(finfo(&fid,Last Modified));
			%let fidc=%sysfunc(fclose(&fid));
			%let rc=%sysfunc(filename(onefile));
```

Further down the macro, you will see that the date being used in the file is a string variable. You need to convert this into a date format by using the input function and feed it into the MODIFYDT macro variable. During the looping process, the macro will check if the date matches the date that you set earlier. If the date matches, then the process continues and allows your subsequent program to run. If an email is set up to notify you that it is successful, then you will also get a "success" notification. If the current time after processing exceeds the TLIMIT set previously, it will convert the FAIL macro variable into a "Y" and the email recipient will get notified. However, if the file is not found and it is still before the time limit, the loop starts all over again. The macro will wait for the designated amount of time you set in the SLEEPTIME variable:

```
			data _null_;
			tdt = datepart(input("&ModifyDT",datetime20.));
			if tdt = date() then call symput("Done","Y");
			else if time() gt &tLimit then call symput("Fail","Y");
			else SleepTime = sleep(&delay,1);
			run;
		%end;
```

There is a possibility that this macro may run concurrently with the process that creates the file of interest which can cause a conflict in determining if the file is available to access. To ensure that the proper return code is recognized during the macro processing, it is advisable to clear out the MODIFYDT, FIDC, and the RC macro variables before the process starts looping again. This will prevent errors that might hinder your macro from looping:

```
		%else %do;
			%let ModifyDT=;
			%let fidc=;
			%let rc=;

			data _null_;
			SleepTime = sleep(&delay,1);
			run;
		%end;
	%end;

	options notes;
```

Once the macro recognizes that the DONE or FAIL macro variables are set to Y, it is recommended re-enabling the note logging. To do this, simply add the statement OPTIONS NOTES back in your program. This will allow you to audit any possible errors in your subsequent programs.

If the process fails, meaning that the time limit has passed without a new file updated, an email will be generated to the recipient notifying them that the process failed, which leads to the last part of the code:

```
%if &Fail=Y %then %do;
      %if %length(&eMail) gt 0 %then %do;
            filename eml email
                  to="&eMail" from="&eMail" subject="&Subj";

            data _null_;
            file eml;
            put "&Mesg";
            put;
            run;
      %end;
         %abort abend;
     %end;
%mend Depend;
```

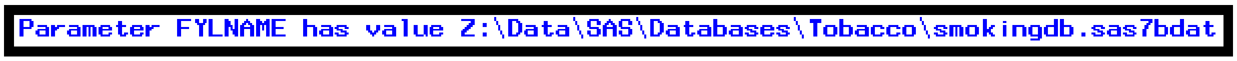
In this option, the macro will abend or terminate the processing and an email will be sent to you or another person if an email address is specified in the EMAIL macro variable. It is important to note that including the ABEND statement will terminate subsequent processing. If you have several programs that rely on this macro to run, you may want to consider omitting the abend statement so that the rest of your programs can still run. The decision to abend the program in case of failure depends on your situation. If this is not your intention, it is highly suggested to omit the abend statement or run the program in batch mode.

Finally, to invoke the macro, you will need to write the syntax that calls this program:

```
%Depend(Z:\Data\SAS\Databases\Tobacco\smokingdb.sas7bdat,delay=15,

      tLimit='09:00:00't,
      email=yourname@anywhere.com,
      subj=Your program,
      mesg=process failed)
```

For this portion, the file name that needs to be updated must be included. The file name is the only required value for this macro to run. The FYLNAME resolves to:



```
Parameter FYLNAME has value Z:\Data\SAS\Databases\Tobacco\smokingdb.sas7bdat
```

In addition, you also have the option to modify the time delay between checking the files while it is being updated and the maximum time that you want the checking to be done. If you decide to populate the call portion of the macro with the corresponding values for DELAY and TLIMIT, it will override the default values in the actual macro. Also, to make this macro program more useful, it is recommended to include an email address, a subject, and a message to yourself or whoever needs to be notified.

ADDITIONAL OPTIONS

This macro, most especially the ABEND option, was initially designed for single-process use. However, if your business needs require subsequent programs to run regardless of the current file being there, simply remove the line containing %ABORT to allow unrelated process that follow to run.

However, please note that simply removing %ABORT will allow the program to rebuild “old” data.   This is where the global macro FAIL comes in. An additional macro to “check for fail” can be included to run if the value of FAIL is ‘N’:

```
%macro TestForFail;
```

```
      %if &Fail = N %then %do;

            <insert your process here>
      %end;
%mend TestForFail;

%TestForFail
```

For the %TESTFORFAIL macro, you would include the subsequent program that processes the file you need in the `<insert your process here>` portion. With this macro, the process that depended on data being updated would not run if the update failed in time, and any unrelated processes that followed would still have the opportunity to run. If you feel that this part is not necessary, you can always convert the global macro FAIL into a local one.

CONCLUSION

In an ever-changing environment such as the healthcare industry where current data is not only critical but necessary, ways of ensuring reliable data need to be in place. This dependency macro can be one of many solutions to help fill that gap. In addition, this code is useful for any type of setting besides healthcare where programs need to rely on current data. Its flexibility to include various options allows one or more programmers to share this macro program within a department. Also, being able to check for other files besides a SAS data set is helpful. Overall, this macro is fairly simple and can be used by programmers with intermediate experience.

CONTACT INFORMATION

Questions and feedback are highly encouraged. Contact the author at:

Lissa Bayang
Kaiser Permanente
Lissa.Bayang@kp.org

APPENDIX A: THE DEPENDENCY MACRO

```
%macro Depend(FylName,tLimit='12:00:00't,Delay=300,
              eMail=,
              Subj=,
              Mesg=);


/* initialize macro variables */
%global Fail;
%local rc fid fidc ModifyDT Done;
%let Done = N;    /* process completed normally */
%let Fail = N;    /* process incomplete at tLimit */


/* suppress excessive note production */
options nonotes;

/* start loop, exit loop when “done” or “fail” */
%do %until(&Done=Y or &Fail=Y);
      /* get last modified value from file */
      %let rc=%sysfunc(filename(onefile,&fylname));
      %let fid=%sysfunc(fopen(&onefile));

      %if &fid ne 0 %then %do;
            %let ModifyDT=%qsysfunc(finfo(&fid,Last Modified));
            %let fidc=%sysfunc(fclose(&fid));
            %let rc=%sysfunc(filename(onefile));

            /* test for today’s date, adjust flags */
            data _null_;
            tdt = datepart(input("&ModifyDT",datetime20.));

            /* file updated today? “done” */
            if tdt = date() then call symput("Done","Y");

            /* file updated today? “done” */
            else if time() gt &tLimit then call symput("Fail","Y");

            /* not updated? wait... */
            else SleepTime = sleep(&delay,1);
            run;
      %end;

      %else %do;
            %let ModifyDT=;
            %let fidc=;
            %let rc=;

            /* not updated? wait... */
            data _null_;
            SleepTime = sleep(&delay,1);
            run;
      %end;
%end;

/* re-enable note production */
```

```
options notes;

/* handle time limit failure */
%if &Fail=Y %then %do;
      %if %length(&eMail) gt 0 %then %do;

            filename eml email
                  to="&eMail" from="&eMail" subject="&Subj";

            data _null_;
            file eml;
            put "&Mesg";
            put;
            run;

      %end;

      %abort abend;

%end; /* end of loop */

%mend Depend;
```