Paper CC-03

Show Me the Folder

Brandon Barrett, Human Genome Sciences, Inc., Rockville, MD
Binoy Varghese, Cybrid, Inc., Wormleysburg, PA

ABSTRACT

Within Base SAS®, there are a lot of tools and readily accessible information that can assist programmers in creating flexible and robust programs. In many industries, SAS programmers work on similar tasks across multiple studies or projects. The less changes SAS programs require from one study to the next, the more efficiently programmers can get their work done. Also, the less input a program needs, the less sources of human error there are to be introduced. This paper shows how to convert multiple MS Excel spreadsheets into SAS datasets without specifying the folder location or a single file name. We describe the SAS code needed for this task, and show how a user can move the program to a different directory location and convert a different set of files to SAS datasets without making any changes to the code.

INTRODUCTION

In clinical studies, data is typically separated into many datasets associated with the different case report forms (CRFs) required for the study. In clinical data management, a single study might also have many different reports intended to check the data for potential problems or inconsistencies. For these reasons, clinical database programmers may need to read multiple spreadsheets into SAS that contain data review comments, raw form data from an Electronic Data Capture (EDC) system, or other information.

If multiple files are stored in the same network folder, a program can be written to read all the files into SAS without explicitly specifying the folder location or any of the file names.

STEPS USED

In general terms, here are the three main steps used in the approach discussed in this paper.

1. Get the folder location from information that is already available to the user within the SAS session.

2. Store a list of the data files to be converted by reading the file contents of the folder into SAS.

3. Read each data file into SAS using generic code that doesn't require explicit specification of the file name.

There are multiple ways to accomplish each of the steps above, but the code provided uses the techniques below.

1. Get the folder location from the XPATH variable in the SASHELP.VEXTFL view, which contains path information about external files identified by the SAS session.

2. Open the folder within a data step using the FILENAME and DOPEN functions, and assign each file within the folder to a row in a temporary SAS dataset using the DNUM and DREAD functions.

3. Within a data _NULL_ step, use CALL EXECUTE command with a PROC IMPORT statement to import each file into SAS.

GETTING THE FOLDER PATH

The VEXTFL view is one of many data views in the SASHELP library available to users. Among the rows in this view, there is one record for each SAS program contained in the folder from where the SAS program is being run. The XPATH variable contains the full path and file name of each SAS program. Using PROC SQL, the code below extracts the folder portion of XPATH variable, excluding the SAS program name, and assigns this text to a macro variable called &PATH.

```
proc sql noprint;
       select tranwrd(xpath,"\" || scan(xpath,-1,'\'),'') into :path from
       sashelp.vextfl
       where upcase(xpath) like '%.SAS';
quit;
```

GETTING A LIST OF THE DATA FILES

Once the folder location is stored in a macro variable, the folder can be opened and read within a data step. In the data step below, the folder path is assigned to a fileref using the FILENAME function, and opened with the DOPEN function. Then "n" variable is assigned the number of files within the folder, and each file is read into the data step using the DREAD function within a DO loop. In this example, we only want to read the spreadsheets with ".XLS" file extension, so we restrict which files are output to the temporary dataset. Here, the code can be modified to store all file names or a different file type found in the folder.

The variable FILE_NAME is created to contain only the name of each spreadsheet without the file extension. This variable is then used to name the SAS datasets created in the third step.

```
data dirxls(keep=file_name);
       length fullname file_name $200;
       rc=filename("dir","&path");
       d=dopen("dir");
       n=dnum(d);

       do i=1 to n;
              fullname=dread(d,i);
              if index(UPCASE(fullname),".XLS") then do;
                     file_name=scan(fullname,1,".");
                     output;
              end;
       end;

       rc=dclose(d);
run;
```

READ EACH FILE INTO SAS

The previous step created a list of the data files to be read into SAS, and saved this list in a temporary SAS dataset called "dirxls". Next, we run the IMPORT procedure on each XLS file and save the output files as a permanent SAS datasets within the same folder. For this purpose, we use the CALL EXECUTE routine in a data step to create a SAS statement from a combination of the FILE_NAME variable and text strings containing the PROC IMPORT code. The macro variable &PATH from the first step above is used to specify the location of the DATAFILE.

```
data _null_;
       set dirxls;
       call execute('PROC IMPORT OUT=curr._'||strip(file_name)||
       ' DATAFILE= "%trim(&path)\'||strip(file_name)||'.xls" DBMS=EXCEL REPLACE;'||
       ' GETNAMES=YES; MIXED=YES; SCANTEXT=YES; USEDATE=YES; SCANTIME=YES; RUN;');
run;
```

When this step has completed, the user should have one SAS dataset created for every XLS file in the same folder from where the program is running. If files of different format need to be read into SAS, this step can be customized with different code in the CALL EXECUTE routine.

FULL CODE

```
**  1) Create a macro variable &PATH with the folder location of this program and XLS
files;

proc sql noprint;
       select tranwrd(xpath,"\" || scan(xpath,-1,'\'),'') into :path from
       sashelp.vextfl
       where upcase(xpath) like '%.SAS';
quit;

%put &path;

**  2)  Create a file with one row per XLS file to be converted;

data dirxls(keep=file_name);
       length fullname file_name $200;
       rc=filename("dir","&path");
       d=dopen("dir");
       n=dnum(d);

       do i=1 to n;
              fullname=dread(d,i);
              if index(UPCASE(fullname),".XLS") then do;
                     file_name=scan(fullname,1,".");
                     output;
              end;
       end;

       rc=dclose(d);
run;

**  3)  Read each XLS file into SAS with PROC IMPORT and save permanently to curr
library;

libname curr "%trim(&path)";

data _null_;
       set dirxls;
       call execute('PROC IMPORT OUT=curr._'||strip(file_name)||
       ' DATAFILE= "%trim(&path)\'||strip(file_name)||'.xls" DBMS=EXCEL REPLACE;'||
       ' GETNAMES=YES; MIXED=YES; SCANTEXT=YES; USEDATE=YES; SCANTIME=YES; RUN;');
run;
```

CONCLUSION

SAS provides a lot of tools that can be combined and utilized to create generic code and automate processes. In this paper, we discuss a technique that shows how a program can be reused for a different task and require minimal to no changes. Removing the need for explicit file references reduces the likelihood of errors.

This technique has been successfully tested using SAS version 9.2 on Windows Operating System.

CONTACT INFORMATION

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

Brandon Barrett
Human Genome Sciences, Inc.
14200 Shady Grove Road
Rockville, MD 20850
brandon_barrett@hgsi.com

Binoy Varghese
Cybrid, Inc.
1017 Mumma Road, Suite 105
Wormleysburg, PA 17043
mailme@binoyvarghese.com