PhUSE 2012

Paper CS01

The Great Outdoors – (Using SAS External File and SAS File I/O Functions)

Lawrence Heaton-Wright, Quintiles

OUTLINE

This paper will concentrate on using the External File and SAS File I/O group of functions to access, identify and control external files and non-SAS data sources such as Excel files. These functions can be used within macro code or within DATA steps (or a combination of the two).

In general I use the macro code statements when trying to create macro variables based on external file information. When I want to manipulate this data and then utilise it (i.e. list of files in a directory) then the DATA step is used and other SAS functions can be used to modify and transform this data into the format required.

This paper will concentrate on identifying some particular uses of these functions and illustrating how these functions can be used to import operating system information into SAS data sets. Alternative coding methods will also be presented for some of these tasks to illustrate the difference in the amount of code required.

INTRODUCTION

SAS programs should not just be about accessing, manipulating, analysing and reporting data. SAS can be used to access, manipulate, analyse and report metadata. This metadata can be used to conditionally control SAS programs or operating system information can be included in SAS programs extending the task set of the SAS System hugely.

In this paper we will be examining the external file and SAS file I/O groups of functions, giving some examples of how these can be used to increase the robustness of our programming. Alternative methodologies will also be presented highlighting the versatility of SAS as well as these functions.

SAS EXTERNAL FILE FUNCTIONS

DCLOSE	DOPTNUM	FDELETE	FINFO	FPOS	FWRITE
DCREATE	DREAD	FEXIST	FNOTE	FPUT	MOPEN
DINFO	DROPNOTE	FGET	FOPEN	FREAD	PATHNAME
DNUM	FAPPEND	FILEEXIST	FOPTNAME	FREWIND	RENAME
DOPEN	FCLOSE	FILENAME	FOPTNUM	FRLEN	SYSMSG
DOPTNAME	FCOL	FILEREF	FPOINT	FSEP	SYSRC

SAS FILE I/O FUNCTIONS

ATTRC	DROPNOTE	FETCHOBS	LIBREF	RENAME	VARINFMT
ATTRN	DSNAME	GETVARC	NOTE	REWIND	VARLABEL
CEXIST	ENVLEN	GETVARN	OPEN	SYSMSG	VARLEN

CLOSE	EXIST	IORCMSG	PATHNAME	SYSRC	VARNAME
CUROBS	FETCH	LIBNAME	POINT	VARFMT	VARNUM
					VARTYPE

HOW MANY OBSERVATIONS DO I HAVE IN MY DATASET?

This is a common question that needs answering. Often when creating data listings early in the course of a study, the data required for display has not been collected yet.

For instance we may need to report the number of deaths or serious AEs on a weekly basis but there may not have been any reported yet. But we don't want to have to change the reporting program when this data is reported as you may have to schedule this program to be executed automatically. One thing we can do is to check the input data to see whether it has any data selected (i.e. 1 or more observations). If there are zero observations selected then a null report can be produced (there are several ways of doing this but we shall be concentrating on the checking of zero observations).

WHAT METHODS CAN BE USED?

External File Functions

```
%LET dsid=%SYSFUNC(OPEN(WORK.input));
```
Uses the OPEN function to open the specified SAS data set

```
%LET nobs=%SYSFUNC(ATTRN(&dsid.,NOBS));
```
Use the ATTRN function to return information about the specified (numeric) attribute of the open data set. In this case we want to identify the number of observations [NOBS]

```
%LET rc=%SYSFUNC(CLOSE(&dsid.));
```
Uses the CLOSE function to close the specified SAS data set

DATA steps or PROC SQL

```
DATA _NULL_;
  SET WORK.input NOBS=nobs;
```
Essentially this data step opens the input data set, sends the NOBS attribute information to a variable (in this case NOBS)

```
  CALL SYMPUTX ('nobs',nobs);
```
CALL SYMPUTX creates a macro variable

```
  STOP;
RUN;
```
The STOP statement immediately stops the data set from processing any further observations. This is important when the input data set has multiple observations as it stops unnecessary processing.

```
PROC SQL NOPRINT;
  SELECT nobs INTO: nobs
    FROM SASHELP.vtable
    WHERE libname EQ "WORK"
      AND memname EQ "INPUT"
  ;
QUIT;
```
Instead of directly interrogating the data set, you can use PROC SQL to interrogate the SASHELP data vies (or dictionary tables if you want to) to select the number of observations in a data set directly into a macro variable.

You could also use the CONTENTS or DATASETS procedures to extract the metadata into a data set and interrogate that. This highlights the versatility of SAS and that there is no "right" way of doing any task in SAS.

WHY USE EXTERNAL FILE FUNCTIONS?

In general because it's much faster than interrogating the SASHELP dictionaries or running PROC CONTENTS and then a DATA step. The DATA _NULL_ step is generally as quick as the external file functions, but I think the functions are slightly more elegant ... and they take up 3 lines of code instead of 5!

IDENTIFYING FILES AND FOLDERS WITHIN A DIRECTORY

Quite often we'll need to identify either the files within a directory or the sub-folders within a directory. Sometimes it's to determine the latest available data in a dated sub-directory or to find the latest version of a document.

External File Functions

```
%LET filrf=mydir;
%LET rc=%SYSFUNC(FILENAME(filrf,"pathname"));
```

Assign the required pathname to a file reference (to MYDIR using the FILENAME function)

```
%LET did=%SYSFUNC(DOPEN(&filrf));
```

Use then DOPEN function to open the directory

```
%LET memcount=%SYSFUNC(DNUM(&did));
```

Use the DUM function to identify the number of items in the specified directory (this will include sub-folders as well as files)

```
%IF &memcount GT 0 %THEN
              %DO i=1 %TO &memcount;
  %LET lstname=%SYSFUNC(DREAD(&did,&i));
  %PUT I=&i. LSTNAME=&lstname.;
%END;
```

Loop through all of the items in the folder

The DREAD function reads the name of the directory member into a specified macro variable. This can then be interrogated to see whether it's a file or directory

```
%LET rc=%SYSFUNC(DCLOSE(&did));
```

Use the DCLOSE function to close the directory

These macro variables can be executed within a DATA step so that all the directory files can be imported into a data set.

PIPEs

Some operating systems (Windows and Unix) allow the use of unnamed pipes. These allow the user to send operating system commands from within SAS and return the results to SAS (usually to a data set).

SAS v9.2 and Windows 7/Server 2008 doesn't "play nicely" with PIPE commands.
See http://support.sas.com/kb/41/863.html for details.

```
%LET DirCmd=%STR('DIR "pathname" /b');
```

Define the OS (MS-DOS) command you're interesting in seeing the results of (a directory listing).
The reason for creating a macro varianble like this is the amount of double- and single-quotes needed around the OS command.

```
FILENAME dir PIPE &DirCmd.;
```

Define a fileref using the PIPE device keyword to tell SAS you want to use an unnamed pipe.

```
DATA _test;
  INFILE dir TRUNCOVER;
  INPUT dirmember $CHAR256. @;
RUN;
```

Use the fileref as normal for reading in an external file.

CREATING A NEW DIRECTORY

There are several ways of creating a new directory. We can manually navigate to the folder and create a new directory. Or we can get SAS to do this for us (especially useful for dated sub-directories or scheduled programs).

External File Functions

```
DATA _NULL_;
  folder="folder_pathname";
  new_folder='Test';
  rc=DCREATE(new_folder,folder);
RUN;
```

The CREATE function creates the required sub-directory in a specified folder.

The FOLDER variable needs to have the final delimiter available on the end (i.e. C:\)

PIPEs

As creating a directory is an operating systems command, an alternative to the DCREATE function we can use unnamed pipes again (with the caveats expressed above).

```
%LET DirCmd=%STR('MKDIR "pathname\new "');
```

Define the OS (MS-DOS) command (creating a sub-directory).

```
FILENAME dir PIPE &DirCmd.;
```

Define a fileref using the PIPE device keyword to tell SAS you want to use an unnamed pipe.

```
DATA _null_;
  INFILE dir TRUNCOVER;
  INPUT dirmember $CHAR256. @;
RUN;
```

Use the fileref as normal for reading in an external file.

DELETING A DIRECTORY/EXISTING FILE

As we can create directories we can also delete (empty) directories or external files (provided we have the security permissions to do so). The FDELETE function allows us to delete directories or files directly controlled by SAS. Alternatively we can submit the appropriate operating system command to delete the directory/file.

WHY USE EXTERNAL FILE FUNCTIONS?

External file functions are, in my opinion, much neater and more efficient commands to use. The PIPE commands require careful consideration of single- and double-quote placement and can get confusing (especially when these commands are embedded within macros).

The help files available for the external file functions are very clear and concise and the functions themselves are logically named so it should be (relatively) easy to understand what is roughly going on in the code even if you don't know the actual functions.

MACRO ERROR-TRAPPING

One of the common uses for the SAS File I/O group of functions is in the design of macros, specifically error-trapping to ensure that the data passed through to the macro in user-defined macro parameters is correct.

For instance, someone uses a macro you've written which requires an input data set but the user incorrectly enters the name of that data set. The user checks the log and sees that the code has errored. They come to you 9as the macro author) saying that the macro is broken. You then have to point out that it is user error as the data set they've specified doesn't exist!

You can make your life easier by building in error-trapping code to check for errors like this which can be used to provide useful log messages for your user community.

COMMON ERROR TRAPS

The SAS File I/O functions have the following functions that can be used inside macros to identify data entry errors.

The LIBREF function checks if a library reference exists. The function returns zero if the library reference exists.

```
%IF (%SYSFUNC(LIBREF(&LibRef.))) %THEN %DO;
  %PUT;
  %PUT ERROR: Library Reference %UPCASE(&LibRef.) does not exist.;
  %PUT        Macro terminating.;
  %PUT;
  %GOTO EndMacro;
%END;
```

Following confirmation that the specified data library is valid, the EXIST function is used to check if a specified data set exists within that library. The function returns non-zero value if the data set exists.

```
%IF %SYSFUNC(EXIST(&LibRef..&DataSet.)) %THEN %DO;
  %PUT NOTE: %UPCASE(&LibRef.).%UPCASE(&DataSet.) exists.;
  %PUT;
%END;
%ELSE %DO;
  %PUT ERROR: %UPCASE(&LibRef.).%UPCASE(&DataSet.) does not exist.;
  %PUT        Macro terminating.;
  %PUT;
  %GOTO EndMacro;
%END;
```

The following section of code can be used to confirm whether a variable exists within a data set.

Confirming a variable exists within a specified data set

```
%LET dsid=%SYSFUNC(OPEN(&DataSet.));
```

Open the data set using the OPEN function.

```
%LET VarExist=%SYSFUNC(VARNUM(&dsid,&Var.));
```

The VARNUM function is used to return the number of the variable's position in the data set. If it returns zero then the variable does not exist in the data set

```
%LET rc=%SYSFUNC(CLOSE(&dsid));
%IF &VarExist. EQ 0 %THEN %DO;
.......
%END;
%ELSE %DO;
.......
%END;
```

Close the data set using the CLOSE function. Use the macro variable VAREXIST to perform a different task if the variable exists or not.

Similar functionality to the EXIST function is available for checking whether files assigned in a file reference actually exist using the FEXIST function. The only difference is that if the file exists the function returns a value of 1 (not a non-zero value) and a value of zero if the file does not exist.

```
FILENAME myfile "pathname_to_file";
%LET fexist=%SYSFUNC(FEXIST(myfile));
```

The PATHNAME function can be used to determine the full pathname of a specified file reference.

```
FILENAME mydir "pathname_to_directory";
%LET pathname=%SYSFUNC(PATHNAME(mydir));
```

This function will include all the information contained within the file reference (this could be a directory reference or a full reference to an actual file).

CONCLUSION

These two sets of functions allow us as programmers to control more than just data manipulation and reporting of a clinical trials database. Utilising these functions allows programmers to interrogate operating systems to discover what files and folders already exist as well as give us the ability to create or delete folders from within SAS.

As has been demonstrated in this paper it is possible to interrogate metadata both from within the SAS System and from the operating system environment and control them using the functions detailed above as well as some of the alternatives available.

REFERENCES

SAS® 9.2 Language Reference: Dictionary, Fourth Edition

CONTACT INFORMATION

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

Lawrence Heaton-Wright
Quintiles
500 Brook Drive,
Green Park,
Reading,
Berkshire.
RG2 6UU
United Kingdom

Work Phone: + 44 118 4508320
Email: lawrence.heaton-wright@quintiles.com
Web: www.quintiles.com