Paper SA-09-2015

Importing multiple spreadsheets in an Excel Workbook: An introduction to Macros

Nathan Becker, Pearson Vue, Bloomington, MN
William Muntean, Pearson Vue, Chicago, IL

Abstract

The SAS® import procedure makes importing Microsoft Excel® files quite simple. However, this procedure can only import a single spreadsheet. Importing more than one sheet from a larger Excel workbook requires using the procedure multiple times and knowing the name of each spreadsheet. Furthermore, handling Excel files that vary in spreadsheet naming conventions necessitates changing SAS code every time encountering a new naming scheme. This is undesirable, especially for repetitive reports. The following paper provides a solution to importing multiple Excel spreadsheets without knowing their names. The concept of macros is introduced and demonstrates the power of generalizable code by being reusable.

INTRODUCTION

Although there are many ways to get Excel files into SAS, the steps below allow getting Excel files into SAS with limited knowledge of the Excel sheets or columns. A search of the internet will return several results on using the SAS Import function. Inspecting Excel sheet names using PROC CONTENTS is one useful method of obtaining sheet names (see Babcock, 2010). The method presented here inspects the sheet names and puts them into a data set so that a user can iterative through the sheets and perform functions on them. To accomplish this, we introduce the concept of SAS macros, which are tantamount to reusable functions. This removes the need to modify the SAS code when importing different Excel files. The paper first covers the steps required to import sheets of an Excel spreadsheet into SAS and then concludes by wrapping up the SAS code within a macro statement.

Step 1 – Create Library Name

The first step is to create a macro variable and library name. When a variable is used several times throughout a SAS program, a macro variable makes it convenient to change the value without modifying a substantial amount of code. When importing different Excel files, the path to the file will change. Therefore, we set a macro variable to the path at the beginning of our program:

```
%Let File = ‘C:\ExcelFile.xlsx';
```

The %LET statement creates the macro variable, which is called by adding preceding the variable name with a “&” (see below). A LIBNAME statement is a convenient method of referring to a storage location, which can take on many forms. In this example, we set a library name for an Excel file using the following code.

```
LIBNAME XDATA &File;
```

Step 2 – Get Excel Sheet Names

After setting up a library name, the PROC CONTENTS inspects the Excel file and returns the names of the sheets. To accomplish this, we use the _ALL_ keyword on our library, which allows PROC CONTENTS to inspect the entire library. For an Excel library, this argument returns information on every spreadsheet contained therein. PROC CONTENTS provides a thorough set of information about a SAS data set. However, we are only interested in retaining the sheet names. The OUT statement in conjunction with the KEEP statement accomplishes this; they create a data set called Sheets that contains the sheet names of the Excel file. Once in a data set, we can loop through the sheet names and import each one.

```
PROC CONTENTS DATA=XDATA._ALL_ NOPRINT OUT=Sheets (KEEP=Memname);
RUN;
```

Step 3 – Cleaning up the data set

This step cleans up the Sheets data set created in the previous section. Because PROC CONTENTS returns a record for every column in a spreadsheet, there is potential for duplicate sheet names in the data set. The SORT procedure with the NODUPLICATES statement removes duplicates, resulting in a data set that has a single entry for each Excel sheet.

```
PROC SORT DATA=Sheets NODUPLICATES;
          BY Memname;
RUN;
```

In addition to removing duplicates, we want to strip off the last character of each name. This is because SAS reads in each sheet name with a “$” appended to the end of the name. And lastly, a unique id is added to each row in the Sheets data set.

```
DATA Sheets;
          SET Sheets;
          ID=_N_;
          Memname = SUBSTR(Memname, 1, Length(Memname)-1);
RUN;
```

Step 4 – Get number of sheets in the Excel file

Because the Sheets data set has all the sheet names of the Excel file, we can obtain the number of sheets by accessing the NOBS variable and setting it to a macro variable. The CALL SYMPUT function creates the macro variable. It takes on two arguments: the first sets the macro variable name (e.g., ‘Recount’) and the second sets its value.

```
DATA _NULL_;
          IF 0 THEN SET Sheets NOBS=X;
          CALL SYMPUT('Recount', X);
          STOP;
RUN;
```

Step 5 – Importing the Excel sheets into SAS

The following block of code loops through the sheet names and imports them into individual data sets.

```
%DO i = 1 %TO &Recount;
```

The code below creates a macro variable (Name) which is the sheet name for each iteration of the loop. The INTO in the PROC SQL creates one or more macro variables that are separated by the colon and name of the variable.

```
          PROC SQL NOPRINT;
          SELECT Memname INTO :Name FROM Sheets WHERE ID = &i;
```

By using the CONNECT TO EXCEL statement in the SQL procedure, we can create a SAS data set labeled with sheet name. Below we are using the macro variable File that is defined in the first step using the Let statement. Later we will switch the location from the Let statement to the Macro call in step 6 below. Here we set the File equal to the path and name.

```
          PROC SQL;
                    CONNECT TO EXCEL (PATH=&File);
                              CREATE TABLE &Name AS
                              SELECT * FROM CONNECTION TO EXCEL
                                        (SELECT * FROM [&Name.$]);
                    DISCONNECT FROM EXCEL;
          QUIT;

%END;
```

Step 6 – Creating a macro

SAS makes it easy to create a macro. Change the first line of code from

```
%Let File = ‘C:\ExcelFile.xlsx';
```

to

```
%macro Excel(File = "NA");
```

and add the following line at the end of the program:

```
%mend;
```

Thus, we have wrapped the code inside a macro statement. This allows us to reuse this function throughout an entire SAS program and project by calling the macro with the following statement:

```
%Excel(File = 'C:\ExcelFile.xlsx');
```

The macro takes on one argument, the variable File. This the location and name of the Excel file to be read by the macro.

CONCLUSION

Getting data from one Excel file can be accomplished in many ways. When importing many Excel files with unknown sheet names, creating a short macro simplifies the task. PROC CONTENTS is a convenient procedure to obtain thorough information about a data set, which is useful when using PROC SQL to import data. Wrapping everything inside a macro allows for reusable code throughout a project.

REFERENCES

Gwen Babcock (2013). Working with Excel spreadsheets using the SAS/ACCESS® LIBNAME statement. Northeast SAS Users Group Forum 2013. Available at http://www.lexjansen.com/nesug/nesug13/89_Final_Paper.pdf

SAS Institute (2011). SAS/ACCESS® 9.3 Interface to PC Files: Reference. Available at http://support.sas.com/documentation/cdl/en/acpcref/63181/HTML/default/viewer.htm#p0u4tggt81xbgtn1uctve6pm1f6q.htm. Cary NC: SAS Institute Inc.

SAS Institute (2011). What's New in the SAS 9.2 Macro Language Facility. Available at http://support.sas.com/documentation/cdl/en/mcrolref/61885/HTML/default/viewer.htm#mcrolrefwhatsnew902.htm. Cary NC: SAS Institute Inc.

Contact Information

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

Nathan Becker
Pearson VUE
nathan.becker@pearson.com