Paper CT-29

We Can Import It For You Wholesale: How to Use SAS® Macro to Import Hundreds of Excel Files

Matthew Gyory, DevTech Systems Inc, Arlington, VA

ABSTRACT

Importing a large number of Excel worksheets into SAS can be a time-consuming and frustrating process. Repeating LIBNAME or PROC IMPORT statements for each Excel file can quickly become overwhelming. However, with SAS Macro %DO loops, PROC SQL and the metadata SAS can extract from files, any SAS user can quickly and easily import dozens or hundreds of Excel files with some or all the associated sheets. This paper shows how.

INTRODUCTION

Accessing data is a fundamental necessity when using SAS. However, loading data into SAS can be a time-consuming and complicated process. This situation manifests itself when the data you need are in multiple Excel files. SAS, especially when combined with the ACCESS Engine, provides the user with the ability to directly read Excel files. However, use of the LIBNAME statement and PROC IMPORT can become tedious once the number of Excel files and sheets goes beyond a handful. What is a user to do if the data needed are in over 100 Excel files and sheets? To answer this question, you need to combine the SAS MACRO facility with PROC SQL to create a macro called %**importExcelFolder**.

SAS AND EXCEL, A COMPLICATED RELATIONSHIP

There are two primary ways we are told to read Excel files into SAS: PROC IMPORT and the LIBNAME statement. The syntax of these options is not difficult, but they each have their own peculiarities that complicate bringing data into SAS. The standard options also become increasingly burdensome when you want to read in multiple files. Each of these stumbling blocks led to the realization that there had to be an elegant way to combine SAS and Excel when the traditional methods failed.

IMPORTING EXCEL FILES INTO SAS

The PROC IMPORT statement is the most basic way to bring Excel file data into SAS. It does not require the ACCESS engine and can handle many other types of files. PROC IMPORT is also the basis for the SAS Import Wizard. The basic syntax of a PROC IMPORT would be the following:

```
PROC IMPORT DATAFILE="filename"
    OUT=SAS_ data set
    DBMS=Excel;
    SHEET="sheet";
run;
```

The LIBNAME statement requires the ACCESS engine and treats each Excel file basically as a SAS data library. The LIBNAME statement requires you to use to data step to actually read in the data and therefore allows more control in creating variables and setting variable type.

```
LIBNAME excel "filename";

DATA SAS_ data set;
    SET excel.'sheet$'n;
run;

LIBNAME excel clear;
```

The major differences between the two approaches are that with the LIBNAME statement you can read in multiple sheets at a time, using the same method you would use to append SAS data set, and the treatment of the Excel sheet names. Since the LIBNAME statement method treats an Excel file like a SAS data library, if you wanted to append three Excel sheets together, that can easily be done using a single SET statement. The LIBNAME statement method also requires sheet names to be entered in the 'sheet$'n format. The $ character is added because of the

way the sheet names are encoded and the quotes (single or double are acceptable) are used since sheet names could include spaces. The n is used to tell SAS that 'sheet$'n is a name literal, much in the same way the '14OCT2012'd is used to tell SAS to store a date value.

LIMITATIONS OF LIBNAME STATEMENT AND PROC IMPORT

PROC IMPORT and the LIBNAME statement are strong basic methods for importing Excel data into SAS. However, each has limitations. PROC IMPORT can handle many different forms of data, but cannot create variables and cannot read in multiple sheets at a time. The user also must have the Excel file closed when using PROC IMPORT. The LIBNAME statement allows the user to create variables and read in multiple sheets from the same file and the user can have the file open while attempting to read in the data, but requires the user to have the SAS ACCESS engine. Using LIBNAME also allows the user to view Excel sheets in SAS prior to import. The choice between using PROC IMPORT and the LIBNAME statement is entirely up to the user; however, this macro uses the LIBNAME statement as it allows greater flexibility in variable management and more direct data access. The main problem with using either method is that they become cumbersome when you need to import more than a few Excel files.

WHAT TO DO WHEN YOU HAVE 100+ EXCEL FILES TO IMPORT

Regardless of your preferred method, all users become stumped when confronted with multiple Excel files to read in and a desire to not have page after page of importing code. To circumvent this issue, a user would need some method of examining a folder, determining which files were Excel files, determining which (or all) sheets to import into Excel and how to name the resulting data sets. SAS provides the tools to accomplish this task by combining PROC SQL with the Macro facility. The result is the %importExcelFolder macro.

A TASK MADE EASIER BY MACROS

SAS Macros makes completing repetitive tasks easier and more efficient. A small amount of code can be reused as much as needed to free the user from page after page of copying and pasting. The Macro facility also allows you to create global variables that can be accessed throughout a macro program. Combining these features with PROC SQL creates the opportunity to read in large number of Excel files and sheets in a single macro call.

DEVELOPMENT OF A MACRO TO SCAN A FOLDER FOR EXCEL FILES

To create %importExcelFolder, you first need to understand what SAS requires to read an Excel file sheet. The method used throughout the macro is the LIBNAME statement. This method was chosen based on personal preference, but the essential logic of the macro would work with either method. The three things SAS needs to read an Excel sheet and create a data set are: the file location, the sheet name and the SAS data set name. No other information is required, nor used in this macro. The macro uses this necessary information as its parameters to create a master import data set and then imports all applicable files and sheets.

KEY COMPONENTS OF THE MACRO

The %importExcelFolder macro uses three parameters to read in Excel files. Filepath is the file path directory of the folder you want to scan and import. Dataname is the name you want to give the resulting data sets and has three options (more on those later). Sheets is the sheet (or sheets) to be imported from the Excel files and also has three options. Using these parameters %importExcelFolder has the following key components:

1. Using filepath, the macro scans the folder and compiles a list of Excel files to be imported
2. Using dataname, the macro determines if it can create unique data set names and creates a list of data set names connected to the available files and sheets
3. Using sheets, the macro determines if the sheet(s) exist in the Excel files and creates a final list of sheets, data set names and file locations for a LIBNAME import
4. Before each of the above steps is finalized, the macro checks and makes sure that the parameters have valid entries and reports any errors it encounters
5. Once all possible Excel sheets have been imported, the macro reports the name of all resulting data sets that were successfully imported and also reports those files not successfully imported (if any)

Each of these components will be examined and the methods used explained so the results of %importExcelFolder can be duplicated and customized to any user's needs.

BASIC FUNCTIONS OF THE MACRO

The %importExcelFolder macro has several basic functions, all with the goal of producing a SAS data set that can be

used to read in an entire folder's worth of Excel files. The inner functions of the macro can be broken down into the following steps:

1. Check the parameters to make sure they were entered correctly
2. Scan the folder and create a list of the Excel files
3. If a single sheet is to be read in, check the data set names and make sure the names are unique. The unique data set names are then added to the master import data set
4. If multiple sheets are to be read it, multiple observations are created for each unique sheet name and file combination. data set name checks occur after each sheet has an observation and if the names are unique they are added to the master import data set
5. The data set is cleaned to ensure that there are no unexpected errors during import
6. Macro variables are created for each observation in the master import data set and the Excel sheets are read into SAS
7. The macro produces a report telling the user which files and sheets were imported and the resulting data set names. The macro also reports those files that failed to be imported.

Each of these functions and their results will be reviewed so a clear picture of the macro's process can be understood.

Basic Parameter Checks

All parameters in the macro are required, so the first check run is to make sure filepath, dataname and sheets have actual values. The macro also makes sure the user did not add quotes to the sheet names. The user is informed about the errors and the macro does not process if these checks fail.

Scanning for Excel Files

Once the preliminary checks have passed, the macro begins to scan the location provided in the filepath parameter. The macro first scans the file location and creates a data set of all files in the folder and makes sure this data set is not empty. If the data set is empty, the macro quits and a message is sent to the user. If there are files in the folder, the macro then determines which files are Excel files (using the .xls and .xlsx extensions) and deletes all non-Excel files from the data set. Before creating the master import data set, the macro checks again to make sure there are Excel files to import and if there are none, the macro informs the user and quits.

The following SAS code is used to create the list of all files in a folder. The FILENAME function takes a variable name and a location (the filepath parameter) and returns a number if the file is non-empty and creates the *fileref* r_dlist. The DOPEN function returns a non-zero number for the *fileref* r_dlist if it can be opened. If r_dlist can be opened, we then go into an if-do statement with a do loop inside it. The DNUM function returns the number of files in r_dlist and then iterates through each of them using the DREAD function. The DREAD function returns the name of the files in r_dlist for each item in the directory. These file names are used to determine the number of Excel files and their file path names for use in the master import data set.

```
DATA r_dirlisting;
    DROP rc did i;
    rc=FILENAME("r_dlist","&filepath");
    did=DOPEN("r_dlist");
    IF did > 0 THEN do;
           do i=1 to DNUM(did);
            name=DREAD(did,i);
            OUTPUT;
           END;
           rc=DCLOSE(did);
    END;
    ELSE PUT 'Could not open directory';
run;
```

The master import data set – complete with file locations, data set names and sheet names – is the result of two different possibilities: the user is reading in a single sheet or the user is reading in multiple (or all) sheets. Reading in a single sheet is the simpler possibility to handle and will be addressed first. However, both possibilities contain similar steps when creating data set names.

Single Sheet data set Name Creation

If the user is going to read in the same Excel sheet from each file, then the importing procedure is more direct and orderly. The sheet column in the master import data set is set to the value used in the sheets parameter and the macro then goes through each of the three possible data set naming options.

The user has three options for the dataname parameter: data, file and a user-defined name. By entering data, the resulting data sets will all be called data, followed by a number, which equals their order in the master import data set. If a user selects data, the program just makes sure that there are fewer than 10^{28} files in the folder. If there are more than 10 octillion Excel files, the macro will report an error and cancel out. Once that check is made, all files are assigned a data set name based on their order in the master import data set and the macro continues on to cleaning the master import data set and importing all files that contain the selected sheet.

The steps followed are the same if a user-defined data set name is selected. The only difference is that the macro calculates the length and then the maximum number of possible data sets. For example, if the user-defined name is "imfdata," then the macro would make sure there are fewer than 10^{25} files and then either report an error or create the data set names.

If the user selects "file" as the dataname parameter, then the macro uses the file names to create data set names. The macro strips out all file extensions, spaces and characters that cannot be used in SAS data set names. If the file starts with a number, the first character is replaced by a "_" during the master import data set cleaning steps. The new data set name is then compressed to remove any remaining spaces and truncated to the last 32 characters if the name is too long. The final check is data set name uniqueness. If there are any duplicates, the macro reports an error and stops processing.

Multiple Sheet data set Name Creation

The macro process for creating names for data sets when the user wants to read in multiple, or all, Excel sheets is similar to the process for reading in a single sheet. The checks on name uniqueness and making sure there are not too many data sets for SAS to handle remain the same. The primary difference is in adding additional observations to the master import data set so that each sheet has its own entry and eventually a unique data set name.

A %DO loop is used to iterate through each Excel file and uses the following code to create entries for each sheet with corresponding file names and file paths.

```
LIBNAME mcrsheet "%sysfunc(trim(&&filename&c))" dbmax_text=32767;

    PROC SQL noprint;
           CREATE table outsheet&c as
           SELECT distinct memname
           FROM DICTIONARY.COLUMNS
           WHERE libname="MCRSHEET" AND memtype="DATA" AND INDEX(memname,'$');
    quit;

    DATA outsheet&c;
           SET outsheet&c (RENAME=(memname=sheet));
           fileloc="&&filename&c";
           name="&&name&c";
    run;

LIBNAME mcrsheet clear;
```

The %DO loop is going through each file path and file name and using the dictionary.columns data set to create a data set with observations for each sheet. The INDEX function indentifies all of the sheets in the file since there will be a $ at the end of each sheet name. The data set then has the proper file name and filepath location added for each sheet.

When the %DO loop is finished, all of the outsheet data sets are compiled into a single data set that will eventually be transformed into the master import data set. Since we now have an observation for every sheet we want to read into SAS, data set names are created as previously described. The only differences are the tests used for determining if the data set name is too long and how the macro copes with truncating potential names.

In all cases, the data set names now represent the sheet name appended to the front of whatever the user selected for the dataname parameter. If the user selected 'data' for the data set names, %importExcelFolder appends the sheet name and data together along with the number corresponding to the data set. If that data set name is longer than 32 characters, the macro will truncate the name to the last 32 characters. The same check is run if a user-

defined name is chosen. This method was chosen because sheet names and data set names could easily repeat if the data set names were truncated to the first 32 characters, resulting in data sets being overwritten during import.

If the user selects file names to be used as data set names, the macro cleans out unacceptable characters, concatenates the sheet name and compressed file name and then shortens the potential data set name to the first 32 characters. If the macro cannot create unique data set names from this list, it produces an error and quits. Once the final data set names have been completed, the macro moves on to cleaning the master import data set and prepares for importing the data.

Preparing for Import

Prior to starting the import process, the macro replaces the first character of the data set name with a '_' if it is a number. The macro removes the final character from the sheet name if the user selected all, or a select number of sheets. This deletion was the result of testing the macro and learning that the method to collect all the sheet names results in the $ remaining on the end. We do not need this character during import, so it is removed. At this stage, those sheets that were not selected by the user (if multiple sheets were the option chosen) are deleted from the import data set. The data set is then run through a cleaning macro that makes sure no unacceptable characters remain in the data set names. The final step before importing the Excel files is to create a place variable, which is simply the observation number for each file or sheet to be read into SAS. The macro also records the number of observations to power a %DO loop. The macro has now produced the master import data set and can begin to import Excel data into SAS.

Importing Excel Files

Once the master import data set is created, the data set names, the datafile locations and sheet names are all turned into macro variables using PROC SQL's INTO : statement. The macro uses a %DO loop to iterate through all possible sheets and files, depending on the user's preferences. The actual importing is done in a simple combination of a LIBNAME statement and DATA step.

```
LIBNAME newdata "%sysfunc(trim(&&location&c))" dbMax_text=32767;

    DATA work.&& data set_name&c (drop=obs);
          SET newdata."%sysfunc(TRIM(&&sheet&c))$"n;
          obs=_n_;
          IF obs>0 THEN CALL SYMPUTX ("rep","yes");
    run;

LIBNAME newdata clear;
```

The obs variable and rep macro variable are used to generate a final report on which files and sheets were successfully imported and which were not. The entire macro boils down into creating a data set, the master import data set, which contains the variables necessary to make these eight lines of SAS code function.

Import Report

The final step of the macro, assuming it has not produced an error and quit during processing, is to produce up to two reports on the success of importing the Excel files and sheets. The program will produce a report detailing all of the files successfully read into SAS. This report includes the full path location, the file name, the data set name and the sheet imported. If there are any sheets or files that were not successfully read in, the macro will produce a report highlighting the full path location and file name of those files. The goal of this report is to allow the user to quickly see which files and sheets were read into SAS, what the data set names are and if there were any unsuccessful imports.

MACRO LIMITATIONS

The %importExcelFolder macro is a powerful tool that can accomplish in a few moments what might take hours to painstakingly program. However, the macro is limited by its design. The obvious limitation is that it requires the ACCESS engine in order to function. Without this addition, the macro will not be able to use the LIBNAME statement to import Excel files. The macro could be designed to use PROC IMPORT, but the preference was for the LIBNAME statement, as explained previously.

An additional limitation is that the dataname parameter might not be able to generate unique data set names. While it is unlikely that a user entering data or a short name to be used for all data sets will encounter problems, those users who wish to use file names could encounter problems. To address this issue, I included numerous checks and error messages. The names generated during the macro also have the potential to be easily confused. The final report was created so users could see how the names of the final data sets match to the files and sheets imported into SAS.

The %importExcelFolder will also only create temporary SAS data sets. If the user wanted to read in dozens of Excel files and create permanent data sets, this macro will only help with the first half of that goal. However, since most data require some formatting and the user may want to combine data sets, the macro provides the basic needs of massive importation of Excel files: getting the data into SAS.

Frequent users of the SAS ACCESS engine have probably encountered the problem of SAS reading in an Excel file and classifying a numeric column as character or it being stumped by a dollar sign or comma. This macro cannot address formatting problems due to problems with the SAS ACCESS engine/Excel interface. This limitation could cause problems if it persists through several data sets, but trying to address the issue is also impractical.

A final limitation is one that is also obvious: this method only works with Excel files. The macro could be modified to import all files SAS can read or at least include CSV files. Such a change would naturally require a renaming of the macro.

CONCLUSION

SAS and Excel are a powerful combination. However, importing a large number of Excel files with SAS can easily become a daunting task. The use of macro %DO loops, PROC SQL steps and SAS functions that can reveal information about entire system folders can make the task of importing large numbers of Excel files easy and routine. The %importExcelFolder macro utilizes all of these tools to simply and directly import data from Excel to SAS quickly, efficiently and successfully.

REFERENCES

SAS Certification Prep Guide: Base Programming for SAS 9. Second Edition, 2009. Cary, NC: SAS Institute, Inc.

SAS Macro Language 1: Essentials. 2010. Cary, NC: SAS Institute, Inc.

SAS OnlineDoc 9.2. 2009. Cary, NC: SAS Institute, Inc.

ACKNOWLEDGMENTS

I would like to thank David Colin, Cris DeBrey and Manuel Figallo for their guidance, recommendations and incidental bits of code during the creation of %importExcelFolder.

CONTACT INFORMATION

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

Name: Matt Gyory
Enterprise: DevTech Systems, Inc
Address: 1700 N Moore St Suite 1720
City, State ZIP: Arlington, VA 22209
Work Phone: 703-778-2665
E-mail: mgyory@devtechsys.com