Paper BtB-10

Taming a Spreadsheet Importation Monster

Nat Wooding, J. Sargeant Reynolds Community College

ABSTRACT

As many programmers have learned to their chagrin, it can be easy to read Excel spreadsheets but problems can arise when one needs to concatenate the data from two or more sheets. Unlike SAS®, Excel does not require that a column have only data of one type so as it scans the columns in a spreadsheet, SAS uses rules to decide the nature of the data in each column. Problems will occur if in one spreadsheet a column is read as characters while in other spreadsheets the data are read as numeric. If there are legitimate character data in the column, then the user will have to decide how to handle them. However, if SAS sees too many blank cells at the start of a column, the incoming variable will be in character format. This paper uses information in dictionary tables to generate SAS code that examines all of the columns (variables) in the workbook and determines which are seen in both formats. Then, data steps read in the "bad" spreadsheets using a data set option to force the problematic columns to be treated as numeric.

Note that this approach requires that the site has licensed SAS/ACCESS Interface to PC Files and the coding is based on a Windows environment.

INTRODUCTION

Over the years, several methods have appeared for reading Excel files. Among these are OBDC (Open Database Communication), PROC IMPORT, DDE, and, with the advent of one of the SAS V9 releases, the Excel LIBNAME method. This paper will only use the LIBNAME method. For the last several releases, SAS has stored the data set's metadata, the data about the data, in so-called dictionary tables. These tables are not directly user-accessible in the way that we view and access SAS data sets meaning that you cannot reference them in a Data Step. However, their contents may be captured by using Proc Contents with an Output statement or by PROC SQL. PROC SQL offers the advantage that information such as Dataset names may be directly loaded into a Macro Variable and in this paper, we will use the SQL approach. Do note that when we use the LIBNAME Excel access method, the spreadsheets in a workbook appear to SAS as members of a SAS data base so the tools that we use to work with a SAS data set may usually be used with the spreadsheets in a workbook.

When we read a single spreadsheet, SAS scans the columns and, based on some options which we can provide, decides the characteristics of each column and uses these characteristics to define the attributes of the new SAS variables in the set(s) in our SAS database. When we concatenate two or more spreadsheets, problems will arise if the characteristics of the variables in one spreadsheet do not match those in another spreadsheet and the concatenation step will fail. As an example, consider the case of appending two spreadsheets of data from environmental data loggers. The two spreadsheets include the same columns (and they are named alike) but in the first case, one of the sensors was not working in the monitoring instrument and the column is left blank. SAS will examine the spreadsheet and will see this as a character variable. In the second spreadsheet, the column is populated with numerical readings and SAS will see them as numbers. The SAS will fuss at us when we try to append the two sheets and will petulantly not join them. If the user is working on a project in which he or she knows the structure of the incoming workbooks, the user could avoid the problem by writing code such as

```
DATA New;
       SET Indd'.sheetone$'n ( DBSASTYPE = ( PH = NUMERIC   O2 = NUMERIC ) );
RUN;
```

This paper attempts to automate the process and avoid extra coding on the user's part when multiple spreadsheets are involved.

CONSTRAINTS

In order to keep the code from being too complex, I have made some assumptions about the Excel files such as there being only one header row. All of the sheets have to be in the same workbook and the code will read all of the sheets. However, if your files do not match one or more of these assumptions, hopefully you will be able to uses the ideas presented here to slay your Excel dragon with the minimum of scorch marks on your person.

OUR SAMPLE PROBLEM

As a sample problem, I created a workbook with four spreadsheets labeled One, Two, Three and One Two. Each contained the columns FirstName, LastName, Height, and Weight. The "name" variables were populated in all cases but height and weight were blank in one of the spreadsheets but not in the other three. Hence, if I imported these spreadsheets and appended them, the appending step would fail since the blank Height and Weight variables would be seen as being in character form.

THE LIBNAME STATEMENT AND A WORKBOOK

Starting in V9, it became possible to refer to an Excel workbook with a LIBNAME statement as in

```
LIBNAME INDD "C:\sesug2013\Win Over Excel.xlsx";
```

Note that the LIBNAME statement includes the full path to the workbook. The beauty of this approach is that for most situations, the spreadsheets stored in the workbook can be treated as if they were SAS data set (this approach is not limited to Excel since Access data bases may also be accessed via this method). Once the LIBNAME statement is issued, an icon for the Libref appears in Libraries in the Explorer Window of Display Manager and if it is clicked, the user sees the sheets in the workbook and can examine them as if they were SAS data sets.

WORKING WITH DICTIONARY TABLES

Dictionary tables were introduced in Version 6 in the early 1990s. For a general discussion of these tables, see DiIorio and Rho, SUGI 29, 2004, or Lafler SUGI 30, 2005. These tables provide the names of the data sets within a SAS data base as well as the names and characteristics of each variable stored in each data set. Hence, there are tables showing attributes at the database level and at the data set level.

As I mentioned, the data contained in the dictionary tables can be viewed with PROC CONTENTS and, if you use an OUTPUT option, the data can be written to a SAS data set. The code is simply

```
PROC CONTENTS DATA = indd._all_  OUT = newset;
RUN;
```

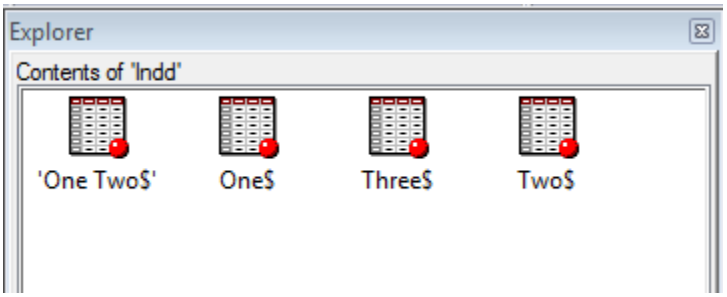
I have come to prefer using SQL to acquire these data so my code is

```
LIBNAME INDD "C:\sesug2013\Win Over Excel.xlsx";
PROC SQL;
     CREATE TABLE Vars AS
     SELECT *
        FROM Dictionary.columns
     WHERE LIBNAME ='INDD';
QUIT;
```

The resulting data set, VARS, will contain the names of the spreadsheets and attributes of each of the variables as SAS sees them. We are primarily interested in the names of the spread sheets, the names of the columns (Variables, in SAS terminology), whether SAS sees them as being character or numeric, and in the case of a real character variables, the variables length and format.

AND NOW, THE MAIN CODE

If we were to look at the INDD library, we would see



Display 1. The Contents of the INDD library

Again, note that SAS shows a "$" appended to each of the sheet names. To reference the sheets in a SAS job, we have to treat the names as named literals such as 'One$'n. However, if we use PROC COPY to copy them to our Work directory but once in as SAS data base such as the Work directory, we cannot open them unless we rename the sets and remove the '$'. This can be done manually or with PROC DATASETS. This step is not needed in the scope of this paper but the feature could be useful if one needed to examine the former worksheet in the form that SAS sees it.

Our next task will be to look at all of the variables in our four data sets and determine which ones appear as both character and numeric data.

```
PROC SORT DATA = vars OUT = vars1;
     BY name type;
RUN;

PROC TRANSPOSE DATA = vars1 OUT = vars1 ( DROP = _NAME_ _LABEL_ ) ;
     BY name;
     ID MEMNAME;
     VAR Type;
RUN;
```

	Column Name	_One_Two__	One_	Three_	Two_
1	FirstName	char	char	char	char
2	Height	num	char	num	num
3	LastName	char	char	char	char
4	Weight	num	char	num	num

Display 2. The Contents of VARS1 Showing the Names of Our Variables and Their Initial Storage Type

Here we can easily see that if we were to append the four spreadsheets, we would have problems with Height and Weight. Our goal now is to process this data set and have SAS identify the problematic variables. Our next step will be to select the variables that are stored in both types and create a macro variable containing the names of the variables which will later be used in a Data step to read the sheets "properly".

```
Data CHARVARS;
        SET Vars1  END = eof;
        LENGTH VarNames  $ 300;
        RETAIN VarNames ;
        ARRAY  Vars _all_ ;
        NCount = 0;
        CCount = 0;
        DO OVER Vars;
           Ncount +   ( Vars = 'num'  );
           Ccount +   ( Vars = 'char' );
        end;
        IF Ncount GT 0 AND CCount GT 0 then wanted = 1;
        IF Wanted THEN VarNames  = COMPBL( VarNames  || Name || ' = NUMERIC ' );
        IF Ccount GT 0 AND Ncount = 0 THEN OUTPUT CHARVARS
        IF eof THEN CALL SYMPUT( 'VarList' , VarNames   );
RUN;
```

Later, we will read each problematic sheet again and our SET statement will include a DBSASTYPE option which will use the macro variable that we just created.

We have one more potential problem. If the spreadsheets have character variables whose maximum length varies among the sheets, we may have problems when we append the sheets since if a spreadsheet with a variable with a longer length is encountered in the appending after the reading of a sheet with a shorter length variable, the longer length variable values would be truncated. For example, if each sheet had only one row and one column and the first one had the name “Wood” and the second “Wooding”, then the resultant two obs would both read “Wood”. We now need to find the maximum length for each character variable and the maximum format width as well.

```
PROC SORT DATA = vars OUT = Lengths ( KEEP = NAME LENGTH );
     BY NAME LENGTH;
RUN;

DATA LENGTHS;
     MERGE LENGTHS CHARVARS ( IN = IN2 KEEP = NAME );
     BY NAME;
     IF LAST.NAME; *We want the longest length. The data are sorted in ascending
order;
     IF IN2;
RUN;

DATA Lengths;
     SET Lengths END = eof ;
     BY NAME;
     LENGTH STRING1 STRING2 $1000;
     Retain String1 'LENGTH ' STRING2 'FORMAT ';
     STRING1 = CATX( ' ' , STRING1 , NAME , '$' , LENGTH );
     FMT     = COMPRESS( CATX( ' ' ,'$' , LENGTH , '.' ));
     STRING2 = CATX( ' ' , STRING2 , NAME , FMT  );
     IF EOF THEN DO;
          CALL SYMPUT( 'LENS' , COMPBL( STRING1 || ';' ));* for a length statement;
          CALL SYMPUT( 'FMTS' , COMPBL( STRING2 || ';' ));* for a format statement;
     END;
RUN;
```

Now we have two MACRO variables, one with the contents of a LENGTH statement and the other with the formats that we will need. To proceed, we need to build the statements of a data step that will read and concatenate the sheets. First, get rid of all but one example of each variable.

```
PROC SORT DATA = Vars1 NODUPKEY;
     BY _Name_;
RUN;
```

At this point, we only care about the contents of the variable MEMNAME which contains the names of sheets in the workbook. Thanks to the NODUP in our sort, we have only one line for each spread sheet. We will now build a Data step "on the fly" which we will execute with CALL EXECUTE.

```
DATA _NULL_;
     SET Vars END = eof ;
     LENGTH STRING $300;
     IF _N_ = 1 THEN DO;
          String =COMPBL( 'Data  Combined ; &LENS ; &FMTS ; SET ' );
          CALL EXECUTE (String);
     END;
     MEMNAME = COMPRESS ( MEMNAME , "'" );
     Sheet   = COMPBL  ( " Indd." || NLITERAL ( MEMNAME )      );
     String  = COMPBL  ( Sheet || " (DBSASTYPE = (&VarList )) ");
     CALL EXECUTE (String);
     IF EOF THEN CALL EXECUTE( ' ; RUN;' );
RUN;
```

Note that macro variable &VarList contains the names of the variables which would be otherwise identified as being in character format and it has the complete DBSASTYPE code. It resolves as

```
Height = NUMERIC Weight = NUMERIC
```

When we execute the data step, our log shows

```
NOTE: CALL EXECUTE generated line.
1   + Data Combined ; LENGTH FirstName $ 9 LastName $ 7 ; ; FORMAT
FirstName $9. LastName $7. ;
; SET
2   +  Indd."One Two$"N (DBSASTYPE = ( Height = Numeric Weight = Numeric
))
3   +  Indd."One$"N (DBSASTYPE = ( Height = Numeric Weight = Numeric ))
4   +  Indd."Three$"N (DBSASTYPE = ( Height = Numeric Weight = Numeric ))
5   +  Indd."Two$"N (DBSASTYPE = ( Height = Numeric Weight = Numeric ))
6   +  ; RUN;
```

Output 1. Our Generated Code Which Was Invoked Via Call Execute

The log shows no problems with mixed data types.

```
NOTE: There were 1 observations read from the data set INDD.'One Two$'n.
NOTE: There were 1 observations read from the data set INDD.'One$'n.
NOTE: There were 1 observations read from the data set INDD.'Three$'n.
NOTE: There were 1 observations read from the data set INDD.'Two$'n.
NOTE: The data set WORK.COMBINED has 4 observations and 4 variables.
```

Output 2. Results of Appending the Four Former Spreadsheets.

Also, there are no complaints about there being different lengths in the data sets.

Here I used CALL EXECUTE to create my SAS statements. I could just as well have used PUT statements to write them to a file and then used a %INCLUDE but I prefer the CALL EXECUTE approach.

CONCLUSION

Users can encounter problems when attempting to append data from spread sheets that have all cells blank in the rows for numeric variables and which SAS sees as being character data. If other sheets are read as having numeric data in those columns, SAS will not append the data sets. Furthermore, character variables whose length varies in the sheets may have truncation problems. This paper shows how to have SAS use Dictionary Tables to explore the variables in the problematic spread sheets and offers a coded solution that reads the workbook and fixes the problems. While I have attempted to make this code as robust as possible, I am sure that there are situations "out there" which will cause it to fail and I would be interested to learn of them.

ACKNOWLEDGEMENTS

I would like to formerly recognize all those well-meaning individuals and faulty instruments who leave data fields blank and otherwise mess up the spreadsheets which they have been given for recording data or, heaven forbid, even change the format of the sheet or data input form. I'm sure that there is a circle in Hell that Dante overlooked but which houses these people. I know that I have recommended that some individuals go there when their time comes.

REFERENCES

DiIorio and Rho, http://www2.sas.com/proceedings/sugi29/237-29.pdf

Heatone, http://www2.sas.com/proceedings/sugi31/020-31.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Feel free to contact the author at:

Name: Nat Wooding
Enterprise: J. Sargeant Reynolds Community College, Office of Institutional Effectiveness
Address: 1651 East Parham Road
City, State ZIP: Richmond, VA 23228
Cell Phone: (804) 357-6655
E-mail: NWooding@Reynolds.EDU
Alternate Email: Nathani@Verizon.Net