Data Transfer Between SAS® Software and Other PC Software

J. Michael Miscisin
Systems Seminar Consultants
Madison, WI 53716 (608) 222-7081

Abstract

The SAS system is a robust tool for interrogating large databases and for performing statistical analyses. Much of the PC software now available is very user friendly and can produce high quality presentations and graphics quite easily. At some point, though, the data one requires will not be in the proper format or on the most appropriate platform. This paper will explore simple and practical solutions to transfer data between SAS and some of the most popular PC-based software packages in use today (Lotus 1-2-3®, Microsoft Excel®, Paradox®, etc.) via ASCII files. Specifically, the creation of flat files from the Data Step and Proc Tabulate using various delimiters as well as the means to create SAS data sets from flat files constructed on PC-based software will be demonstrated.

Creating Simple Flat Files

Creating flat files for SAS Data sets can be accomplished quickly and easily. If we have a SAS data set named NEW with the variables STATE, VARA, and VARB, where STATE is character and the other two variables are numeric, we could create a simple flat file using the DATA Step.

```
DATA _NULL_;
   SET NEW;
   FILE OUTDS;
   PUT "STATE"," VARA "," VARB;
RUN;
```

The flat file from this DATA step is a quotation mark and comma delimited file used commonly by spread sheet and database applications.

```
"AZ ",2500 ,500
"AZ ",1200 ,700
"WI ",3200 ,100
"WI ",9600 ,300
"WI ",4900 ,200
"NH ",5400 ,800
```

Most popular PC software expect a flat file to contain some character to indicate where one variable ends and the next begins on each individual line of the file. While this character, often called a delimiter, could be almost any character we choose, the most common is a comma. In addition to a delimiter separating the individual variables, using quotation marks to enclose character values can be helpful when importing to other PC software packages. Typically, those values enclosed in quotation marks will be imported as character. Those without will be imported as numeric values.

The DATA step, while typically used to create SAS data sets, is used to create a flat file in this case. The first line of the DATA Step specifies the keyword _NULL_ as the data set name to prevent the creation of a SAS data set. We could have created both a SAS data set and a flat file by specifying a data set name on the DATA statement instead of the _NULL_ keyword.

The FILE statement points to an alias, OUTDS, for the flat file created in this DATA step. OUTDS refers to some file external to the SAS System. This alias can be created in a FILENAME statement. The name of the physical file depends on the operating system you use. Here are two examples:

```
DOS . FILENAME OUTDS 'C:\FLATFILE';
MVS . FILENAME OUTDS 'G185TY.SAS.TLAFLATFILE';
```

The PUT statement uses list style of put to create the individual lines of the flat file. Both variable values and character strings must be written to the flat file.

Since we need to enclose the character values in quotation marks and separate the variables with commas, we must include character strings in the PUT statement in addition to the variable names. All character strings must be enclosed in quotation marks for the SAS System to make the correct interpretation. Notice that the PUT statement above begins with a single quotation mark. It is followed by a double quote (the character string to write out) and then a single quote, completing the character string. The first variable comes next followed by another single quote, a comma to serve as the delimiter between variable values, and then a single quote to close this character string. This pattern repeats until you have listed all of the character variables you wish to include in your flat file.

Numerical variables are simply delimited by commas. The numeric variable portion of the PUT statement is a single quote, a comma, and another single quote followed by the numeric variable name. We repeat this pattern until all numeric variables we wish to include in
the flat file have been listed.

Notice that the PUT statement does not end with a comma. If a trailing comma did appear you would receive a single line of data from the multi line flat file. Ending with a comma causes software reading a flat file to read all lines as a continuous stream of data with no end of line indicators.

The QUOTE Function

Writing out double quotation marks as character strings can be confusing. The QUOTE Function can help alleviate some of this confusion by quoting the character variables in separate statements instead of on the PUT statement.

```
DATA _NULL_;
SET NEW;
FILE OUTS;
STATE=QUOTE(STATE);
PUT STATE, ' VARA , ' VARB;
RUN;
```

The code remains quite similar to the first example, but we use the QUOTE Function to place double quotation marks around the values for STATE before the PUT statement. We must still write out commas as character delimiters. The result of this DATA step is almost identical to the first.

Flat Files with Many Variables

The simple, straight forward method of creating flat files above works very well if you have only a handful of variables you would like to write out. However, writing a PUT statement for a data set with as few as 15 variables can become tedious. Additionally, we must know which columns contain which pieces of data on our flat file when we import the flat file to some other software.

With the method below, the SAS variable names will be the first line of data on the flat file. This makes importing much easier because the SAS System automatically supplies the SAS variable names which most spreadsheet or database software can then convert to column or field names. This SAS Macro creates a comma delimited flat file for all variables on a data set automatically.

```
%MACRO CREFLAT(FILE);
FILENAME Z "C:\&FILE";
DATA _NULL_;
FILE Z;
SET &FILE;
IF _N_=1 THEN DO;
```

This SAS Macro called CREFLAT contains several steps. The first, in line 1, establishes a file reference for the flat file we want to create. In this case, I have referred to a DOS file in the C:\ directory which will have the same name as the SAS data set we will read to create the flat file. You could easily modify it to create a file in MVS or other operating system.

Next, the DATA step begins. As in the basic flat file program on the previous page, we use the _NULL_ keyword because we do not need to output a SAS data set. The SET statement uses the Macro variable &FILE as the SAS data set name. The value of &FILE will be established at the invocation of the Macro and will be described later in this paper.

The next section of the Macro from lines 9 through 24 writes the first line of the flat file containing the variable names. Since one line of the flat file will contain variable names, we only execute this section of code on the first observation read by the SET statement (if _N_=1 then do). Then we define two arrays. The array in line 10 (ARRAY CHAR) contains one
element for each character variable, while the second array, ARRAY NUMS, contains one element for each numeric variable.

The _CHARACTER_ and _NUMERIC_ keywords are used to indicate all of the character and numeric variables, respectively. The length statement specifies the length of the variable VARNAME that will contain the actual variable names.

Line 14 begins a DO loop. This DO loop loops from 1 to the number of character variables on the data set. The DIM function on this line returns the number of elements in array CHAR. Since array ABC contains the character variables, DIM(CHAR) is the number of character variables.

The CALL routine in line 15 retrieves the name of a character variable and places it into a new variable called VARNAME through each iteration of the DO loop. The VNAME CALL routine returns the variable name for the current element of the CHAR array, which is one of the character variables and places it into the variable VARNAME. So, the first argument of the VNAME CALL routine is the variable for which you want to retrieve the variable name. The second argument is the variable that will contain the actual variable name as its value. The LABEL CALL routine could be used in place of the VNAME statement to retrieve the variable labels instead of the variable names.

The PUT statement then writes out the value of VARNAME for this array element enclosed in quotation marks and followed by a comma. The trailing @ then holds the value for VARNAME until the end of the DO loop.

The program repeats the same process of retrieving variable names using array NUMS, the numeric variables. This array is similar to array CHAR used for the character variables. Using the data set as above, we would write out a line ending with a comma.

"STATE", "VARA", "VARB",

This will cause problems when reading the file with most software. We can remove the trailing comma with another PUT statement as in line 23.

Now we need to write out the actual data values to the flat file. Again, we use a DO loop to loop through all of the character variables (array CHAR, beginning on line 26), enclosing the values in quotation marks and following them with commas. We then perform a similar DO loop for the numeric variables on lines 30-32. Since we do not want to enclose numeric variable in quotation marks, they are not included in this case.

We also specify the BEST9. format. You could specify any numeric format here, but only one format for all of the numeric variables can be used in this Macro as written. 

Again we would receive a comma at the end of each line without the final PUT statement on line 34. The CREFLAT macro ends by unassigning the Z filename on line 38 to prevent overwriting Z with something else in a future step of your program.

Putting the Macro Method Into Practice

The easiest way to use this in everyday programming would be to store the program above in one file. Once it has been debugged, no changes should be required. I would then use a %INCLUDE to use this Macro in your program and then invoke CREFLAT, specifying the data set name you would like to write to a flat file. Let's assume I have stored the above program in C:\STND\CREFLAT. The code used to include and execute the CREFLAT Macro would be:

```
%INC 'C:\STND\CREFLAT';
%CREFLAT(XXXX);
```

Where XXXX is the name of the SAS data set you wish to write to a flat file. &FILE on the SET statement will also resolve to the name of the SAS data set you wish to write to a flat file.

Please exercise caution when using this method with data sets that contain more variables than you wish to write out. You may want to consider dropping the variables you do not want to write out in a preceding PROC or DATA step to avoid extraneous data on the flat file.

A Practical Example

Perhaps we would like to calculate means and sums for variables VARA and VARB on the NEW data set by state. PROC Means is a robust and easy tool to use to calculate these two statistics. We would like to include this data in a presentation but do not want to retype the data from the report PROC Means would produce. We can use PROC Means to produce an output SAS data set containing the statistics we want and write that data set to a flat file using the CREFLAT Macro.

```
PROC MEANS DATA=NEW NOPRINT NWAY;
CLASS STATE;
VAR VARA VARB;
OUTPUT OUT=MEANSOUT
    MEAN(VARA)=AVGVARA
    MEAN(VARB)=AVGVARB
    SUM(VARA)=SUMVARA
    SUM(VARB)=SUMVARB;
```
%INC 'C:\STND\CREFLAT.SAS';
%CREFLAT(MEANSOUT);

Here we have created a SAS data set called MEANSOUT. We then %INC the CREFLAT Macro and invoke it for the MEANSOUT data set. This creates a flat file also called MEANSOUT.

This flat file can now be easily imported into the presentation/graphics software used to create the presentation.

Creating SAS Data Sets from Flat Files

Almost inevitably during your career using SAS Software, you will have some data in some other software package that you would like to use in the SAS System. One of the simplest ways to do this is through flat files. Most spreadsheet and PC database products will create flat files virtually automatically through the use of dialogs and menus. The flat file created by most of these products are quote and comma delimited, similar to those we created above using SAS Software.

If the other software we were using produced the following flat file

"TURNIPS",5478,874
"RUTABAGAS",9852,521
"POTATOES",8738,467
"PARNIPS",1286,238
"YAMS",3524,972
"CASSAVA",7513,648

we could easily read it into SAS.

DATA RTVEG;
  INFILE ROOTVEG DLM="",";"
  INPUT VEGNAME $ VARA VARB;
RUN;

RTVEG created above:

RTVEG DATA SET

<table>
<thead>
<tr>
<th>OBS</th>
<th>VEGNAME</th>
<th>VARA</th>
<th>VARB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TURNIPS</td>
<td>5478</td>
<td>874</td>
</tr>
<tr>
<td>2</td>
<td>RUTABAGA</td>
<td>9852</td>
<td>521</td>
</tr>
<tr>
<td>3</td>
<td>POTATOES</td>
<td>8738</td>
<td>467</td>
</tr>
<tr>
<td>4</td>
<td>PARNIPS</td>
<td>1286</td>
<td>238</td>
</tr>
<tr>
<td>5</td>
<td>YAMS</td>
<td>3524</td>
<td>972</td>
</tr>
<tr>
<td>6</td>
<td>CASSAVA</td>
<td>7513</td>
<td>648</td>
</tr>
</tbody>
</table>

Using the DLM= option works well with the flat file above. However, we would receive different results if missing data appeared in the flat file depicted here.

"TURNIPS",5478,874
"RUTABAGAS",9852,521
"POTATOES",8738,467
"PARNIPS",1286,
"YAMS",3524,972
"CASSAVA",7513,648

Running the same program used to read the original flat file to read this new file would yield a data set with missing values in incorrect locations.

RTVEG1 DATA SET

<table>
<thead>
<tr>
<th>OBS</th>
<th>VEGNAME</th>
<th>VARA</th>
<th>VARB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TURNIPS</td>
<td>5478</td>
<td>874</td>
</tr>
<tr>
<td>2</td>
<td>RUTABAGA</td>
<td>9852</td>
<td>521</td>
</tr>
<tr>
<td>3</td>
<td>PARNIPS</td>
<td>1286</td>
<td>238</td>
</tr>
<tr>
<td>4</td>
<td>CASSAVA</td>
<td>7513</td>
<td>648</td>
</tr>
</tbody>
</table>

Notice that for observation 2 the VARB value is now in VARA. This is because the FLOWOVER option is in effect by default for list style of input. The FLOWOVER option indicates that if the INPUT statement reads past the end of the current record, it is to read data from the next record. That is what occurred here.

Beginning with version 6.07, the DSD option became available. Using the DSD option causes the DATA step to treat the consecutive delimiters individually. This will force a missing value to appear in the SAS data set when value is missing between two delimiters. The DSD option assumes that the delimiter is a comma. Additionally, the DSD option strips all values of quotes before writing them to a variable; hence, we can completely eliminate the DLM= option on quote and comma delimited flat files. You may still specify other delimiters on the DLM= option. The program to correct the missing value problem now looks like this.

DATA RTVEG1;
  INFILE ROOTVEG1 DSD;
  INPUT VEGNAME $ VARA VARB;
RUN;
The new data set now has the missing values in the correct variables.

<table>
<thead>
<tr>
<th>OBS</th>
<th>VEGNAME</th>
<th>VARA</th>
<th>VARB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TURNIPS</td>
<td>5478</td>
<td>874</td>
</tr>
<tr>
<td>2</td>
<td>RUTABAGA</td>
<td>521</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>POTATOES</td>
<td>8738</td>
<td>467</td>
</tr>
<tr>
<td>4</td>
<td>PARSNIPS</td>
<td>1286</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>CASSAVA</td>
<td>7513</td>
<td>648</td>
</tr>
</tbody>
</table>

Other Methods to Transfer Data

There are many other means of transferring data that are more sophisticated. These include OLE, ODBC, SAS/ACCESS®, DDE, and others. While these other methods are very useful for exchanging the same data between software packages on a frequent basis, they are not as flexible for the one-time, ad hoc use that is exceedingly common in today’s business environment. We still find flat files a very reliable and simple method to transfer data between software platforms.

CONCLUSION

Transferring data between SAS and other software packages is a common activity in the current business environment and will continue to be. Creating flat files with the almost automatic SAS Macro method and reading flat files into SAS Software with the appropriate options can make the process swift, reliable, and relatively easy.

SAS and SAS/ACCESS are a registered trademarks of SAS Institute Inc. in the USA and other countries. Lotus 1-2-3 is a registered trademark of Lotus Development Corporation. Microsoft Excel is a registered trademark of Microsoft, Inc. Paradox is a registered trademark of Borland, Inc. © indicates USA registration.

References
