CC145

Because We Can: Using SAS® System Tools to Help Our Less Fortunate Brethren

John Cohen, Advanced Data Concepts, LLC, Newark, DE

ABSTRACT

We may be called upon to provide data to developers -- frequently for production support -- who work in other programming environments. Often external recipients, they may require files in specific formats and variable/column order, with proscribed delimiters, file-naming conventions, and the like. Our goal should be to achieve this as simply as possible, both for initial development and ease of maintainability. We will take advantage of several SAS tricks to achieve this goal.

INTRODUCTION - OUR BUSINESS PROBLEM

We have been asked to create a monthly production file extract in a specific, highly-structured format. We would like to create a process retaining as much of the flexibility of the SAS environment as possible. To minimize maintenance efforts, we will design a dynamic, automated process. Chances are that you will never face exactly this same combination of requirements. However, it IS quite likely that at least one of the techniques we will explore here will become a productive addition to your tool set.

OUR PROGRAMMING TASKS

We are under a requirement to change our file structure as little as possible so as to not affect any of the downstream processes dependent upon our work. If we DO change anything, there will be issues around testing and validation of these downstream processes – which cannot simply be assumed to be part of the on-going maintenance for the dependent processes. Rather these may need formal project development plans, prioritization, and scheduling of any maintenance work needed even for what for us seem to be simple changes. Otherwise these downstream processes will be at severe risk of being out of synch with ours. So the fundamental flexibility of SAS results in the burden of accommodating changes to fall particularly heavily on us. Making our approach to system/program design one which minimizes maintenance is thus of considerable importance. After all, one should always assume the possibility of lots of changes over time, all of which we will want to make appear as transparent to those using our output as possible. The three particular sub tasks are:

- The order of the columns should exactly match the order of our original source file, with any new variables being added at the end of the file
- We are asked to use a specific delimiter (ASCII 27)
- We want the file names (in a monthly process) to follow a specific format as designated by the downstream users

THE STARTING POINT

Our data, historically, have looked much like the example in Figure 1 below. The data (column ordering) are in the historical order. In SAS, providing that the column (variable) names have not changed, we pay little attention to the order of columns. But if we export our data into other formats, the ease of manipulation of these data may be compromised. Programs reading in fixed-format files, formatting of Excel spreadsheets, graphing of time series data, and the like will potentially require reordering of columns prior to executing these functions. Alternatively, downstream programs will need to be updated whenever we change our column order.

If we can devise a methodology to assure constant, consistent ordering of columns, the only high-risk issue will be instances where we add or remove columns. (Changing the order of the existing columns will be handled by our process automatically, if correctly designed.) If we further constrain our structure so that new variables are added only at the end of our file, we will minimize the maintenance required on the part of the support folks for these downstream processes.

To minimize maintenance we will let the incoming data stream help to write the program for us. This dynamic approach to programming, (sometimes called a program-to-write-a-program) has been around for a while. The format we will employ here (also, please see Howard) will take the incoming data, use these data values to write out valid SAS programming statements to a temporary file, then execute these statements with a %Include statement. (For more recent approaches to dynamic programming, please see Carpenter & Fehd.)

FIGURE 1 – THE HISTORICAL DATA

Plan Code	Plan	Employer Group	State Medicaid	Medicare Part D (added Jan. 2006)	HMO	PBM
000145					OXFORD	MEDCOHEALTH
002003			ALASKA MEDICAID			
002053			OHIO MEDICAID			
003034	AETNA (FL)				AETNA	
003058	AETNA (PA)				AETNA	
010314					HUMANA PART D	
019001	TUFTS					CAREMARK
019014				TUFTS PART D		CAREMARK
020004	UHC (RI)				UHC	MEDCOHEALTH
020136					MAMSI	MEDCOHEALTH
030001		GM CORP				MEDCOHEALTH
050179					CIGNA	
200146					UHC PART D	WALGREENS
200531				JOHN DEERE PART D	UHC PART D	PROCARE
300118	BCBS (TN)					CAREMARK
340001		DUPONT				MEDCOHEALTH
460001		CALPERS (CA)				MEDCOHEALTH
560001		VERIZON COMM.				MEDCOHEALTH
610001		US AIRWAYS				CAREMARK
680001	CAROLINA PLAN (SC)					EXPRESS SCRIPTS
700060					AARP PART D	WALGREENS
790026	ANTHEM (ME)					WELLPOINT
790033	WELLPOINT (KY)					WELLPOINT
790056				BLUE MED PART D		WELLPOINT

TASK 1A - CAPTURING THE ORIGINAL FILE COLUMN ORDER

Our first step is to find the order of the columns in our original SAS data set. We will use a PROC CONTENTS with and OUT= option to capture the metadata, or the data describing the structure (including the variable/column order) of that original SAS data set. Conveniently, the output from the OUT= option of PROC CONTENTS is, in fact, another SAS data set. As we are experienced SAS programmers, then, this means that we will have at our disposal all of the standard SAS tools to manipulate the output data set. The program to capture the original/historical/baseline variable/column order might look like the following:

TASK 1A – PROGRAM TO CAPTURE THE ORIGINAL FILE COLUMN ORDER

```
proc contents
   data=perm.original_file out=orig;

proc sort data=orig(keep=name VARNUM);
   by name;
```

and the resulting SAS data set might have characteristics something like this:

TASK 1A – RESULTING OUTPUT

```
Data Set Name: ORIGINAL_FILE              Observations:       18
Member Type:   DATA                      Variables:          6
Created:    10:11 Monday, April 22, 2005    Observation Length:   91

-----Alphabetic List of Variables and Attributes-----
```

#	Variable	Type	Len
3	Employer_Group	Char	22
5	HMO	Char	6
6	PBM	Char	15
2	Plan	Char	23
1	Plan_Code	Char	6
4	State_Medicaid	Char	15

TASK 1B - CAPTURING THE CURRENT FILE COLUMN ORDER

Our next step is to find the order of the columns in our current SAS data set. We will again use a PROC CONTENTS with an OUT= option to capture the *metadata*, and the results should look remarkably similar. The program to capture the current variable/column order might look like the following:

TASK 1B – PROGRAM TO CAPTURE THE CURRENT FILE COLUMN ORDER

```
proc contents
   data=current_file out=curr;

proc sort data=curr(keep=name varnum);
   by name;
```

with the output looking like this:

TASK 1B – RESULTING OUTPUT

```
Data Set Name: CURRENT_FILE               Observations:       24
Member Type:   DATA                      Variables:          7
Created:    15:12 Tuesday, March 13, 2006   Observation Length:   116

-----Alphabetic List of Variables and Attributes-----
```

#	Variable	Type	Len
3	Employer_Group	Char	22
6	HMO	Char	6
5	Medicare_Part_D	Char	25
7	PBM	Char	15
2	Plan	Char	23
1	Plan_Code	Char	6
4	State_Medicaid	Char	15

<=== NOTE: new column

TASK 1C - COMPARING THE ORIGINAL FILE COLUMN ORDER TO THE CURRENT FILE

Our next task is to compare our original file column order to that of the current file. Having sorted both data sets, Orig and Curr, by the same variable, **Name**, we can now merge these two data sets by the variable Name. In the simplest case, all variables in one are also in the other and nothing needs to be done. If there are discrepancies, on the other hand, we will want to create a process for reconciling these. We will do this by creating an additional ordering variable – **Order** – to track whether or not variable names are in both metadatasets (and order will be set to 1) or only in one of the two (and order will be set to 2). The resulting program might look like this:

TASK 1C – PROGRAM TO COMPARE THE ORIGINAL FILE COLUMN ORDER TO THE CURRENT FILE AND RECONCILE DISCREPANCIES

```
data compare;
   merge orig(in=orig rename=(varnum=orig_varnum))
   curr(in=curr rename=(varnum=curr_varnum));
   by name;

   if orig and curr then status = ‘Both’;
    else if orig then status = ‘Orig’;
    else status = ‘Curr’;

   if status in (‘Both’,‘Curr’) then order = 1;
    else order = 2;
```

The resulting data set Compare should look like the following, with the combined list of variable names sorted in **name** order, the respective **varnum** orders being captured, and a **Status** and **Order** value beig set. In this example all original variables still exist in the current data set, with one new variable name, Medicare_Part_D in the current file needing to be reconciled. The value of **Orig_varnum** will be missing for this row (it did not exist in the original file), the value for **Status** is “Curr” (it only exists in the current file), and the value of **Order** is “2”. If a variable **name** occurred only in the original file, the respective values of Status would be “Orig” and **Order** would be “1”.

TASK 1C – DATA SET *COMPARE*

Name	Orig_varnum	Curr_varnum	Status	Order
Employer_Group	3	3	Both	1
HMO	5	6	Both	1
Medicare_Part_D	**.**	**5**	**Curr**	**2**
PBM	6	7	Both	1
Plan	2	2	Both	1
Plan_Code	1	1	Both	1
State_Medicaid	4	4	Both	1

TASK 1D – RE-ORDER THE COMBINED FILE COLUMN ORDER

The last step in using these metadata files is to reorder the combined metadata file **Compare** in the format indicated by our comparison and reconciliation process. Our plan is to sort Compare by three variables, the **Order** variable, then **Orig_varnum**, and finally **Curr_varnum**. This should result in the following:

- Variables in both data sets will retain the original order
- Variables only in the original file will retain their original column order and will have space reserved (but carry missing values)
- Variables only in the current file will be added at the end of the new file, retaining their order in the current file

TASK 1D – PROGRAM TO RE-ORDER THE COMBINED FILE COLUMN ORDER

The program is a simple sort as follows:

```
proc sort data=compare out=final_order;
   by order orig_varnum curr_varnum;
```

With the resulting data set **final_order** looking something like:

TASK 1D – DATA SET FINAL_ORDER

Name	Orig_varnum	Curr_varnum	Status	Order
Plan_Code	1	1	Both	1
Plan	2	2	Both	1
Employer_Group	3	3	Both	1
State_Medicaid	4	4	Both	1
HMO	5	6	Both	1
PBM	6	7	Both	1
Medicare_Part_D	**.**	**5**	**Curr**	**2**

CONCLUSION TO TASK 1 – WHAT HAVE WE ACHIEVED?

We have successfully captured the metadata description of our original file, keeping the variables of interest from the PROC CONTENTS OUT= option, performed the same task on the current file, compared and reconciled any discrepancies, and identified the order of rows for which we have in mind our new file. The resulting SAS data set, **final_order**, will be the input for our next step.

TASK 2 – PROGRAM-TO-WRITE-PROGRAM – PREPARE TO REORDER THE PHYSICAL DATA

Our next task is to use the data from our last step to write a program (actually, a single Retain statement). We will use a **libname** statement to define a temporary file and a **Data _Null_** to write and store the SAS programming statement(s) in that temporary space. The exact format (i.e., do you choose to write a syntactically-complete SAS statements or instead statement fragments inserted within appropriate "completion" phrases at the point of calling in these fragments) is as much as anything else a question of programming style. We will show here how to create complete SAS statements from key word though sequential value processing to the final ubiquitous trailing semi-colon.

After our **libname** statement (with a **libref** of "**Out**"), we start our Data step using the data set **final_order** created above and a flag to let us know when we have processed the last observation (**end=eof**). When we encounter the first observation (**if _n_ = 1**) we will write to **Fileref Out** our key word "Retain". We then write the value of the variable name to **Fileref** for each observation in the data set **final_order**. When we reach the end of file (**if eof**) we write the trailing semi-colon to **fileref Out**. The result should look like a syntactically correct retain statement with the variable names listed in the order we chose through our compare and discrepancy reconciliation process in Task 1.

TASK 2 – PROGRAM TO WRITE PROGRAM

```
filename out 'C:\retain.txt';                   /** define temporary file **/
data _null_;
   set final_order end=eof;                 /** flag for last observation **/
   file out;                      /** where our statements will be stored **/
   if _n_ = 1 then put @4 'Retain' @; /** start with keyword, hold line **/
   put @11 name;                    /** each observation has value posted **/
   if eof then put @4 ';';  /** at very end of file, add the semi-colon **/
run; /** a reminder to make sure this step executes before the next one **/
```

TASK 2 – CONTENTS OF FILEREF OUT

```
/234567890123456789012345678901   (column numbers 1 through 30)
------------------------------
   Retain Plan_code
          Plan
          Employer_group
          State_Medicaid
          HMO
          PBM
          Medicare_Part_D
   ;

   /*** syntactically, fully-valid SAS program (fragment)! ***/
```

CONCLUSION TO TASK 2 – WHAT HAVE WE ACHIEVED?

Simply, we have taken the desired list of variables and the preferred order as captured in the metadataset **final_order** and expressed that as a **Retain** statement stored in temporary fileref **Out**. Note that we have created a statement formatted (with indentation) much as if we had hand-coded this program fragment. As per the comment in the sample program, we must use a run statement to ensure that this **Data _null_** step executes ***before*** we try to call back the created statement (or we will potentially be **%including** a blank statement – which will not necessarily generate an error message, but most certainly will not give us the results we seek).

TASK 3 – %INCLUDE & EXECUTE PROGRAM FRAGMENT, SET FILE NAME, EXPORT USING REQUIRED DELIMITER

After our initial two PROC CONTENTS (on the original and current files), we have been have been working entirely with metadata, descriptions of the contents and structure of SAS data sets. We are about to resume working directly with the actual data, as it were. We will

- **%include** – or call the program fragment – a **retain** statement within the appropriate place in our next step
  - the **Retain** statement will be called before our first Set statement, thereby changing the order of the variables in the incoming data set – **current_file** -- to that indicated on the retain statement
- use the SAS **Macro language** to minimize maintenance around the requested output file naming convention
- export the resulting SAS data set to a delimited flat file using **PROC EXPORT**
  - we will specify the requested delimiter character in our PROC EXPORT **delimiter**= option

TASK 3A – PROGRAM TO %INCLUDE & EXECUTE PROGRAM FRAGMENT

```
                                                 /** what we will submit **/
data final;
   %include out;                     /** Note: same fileref as in step 2 **/
   set current_file;  /** %include Retain statement before Set statement **/
run;

     /****************************************************************/

                                                   /** what will execute **/
data final;
   Retain Plan_code
          Plan
          Employer_group
          State_Medicaid
          HMO
          PBM
          Medicare_Part_D
   ;
   set current_file;  /** %include Retain statement before Set statement **/
run;
```

TASK 3B – SET OUTPUT FILE NAME, SELECT DELIMITER, AND EXPORT FILE

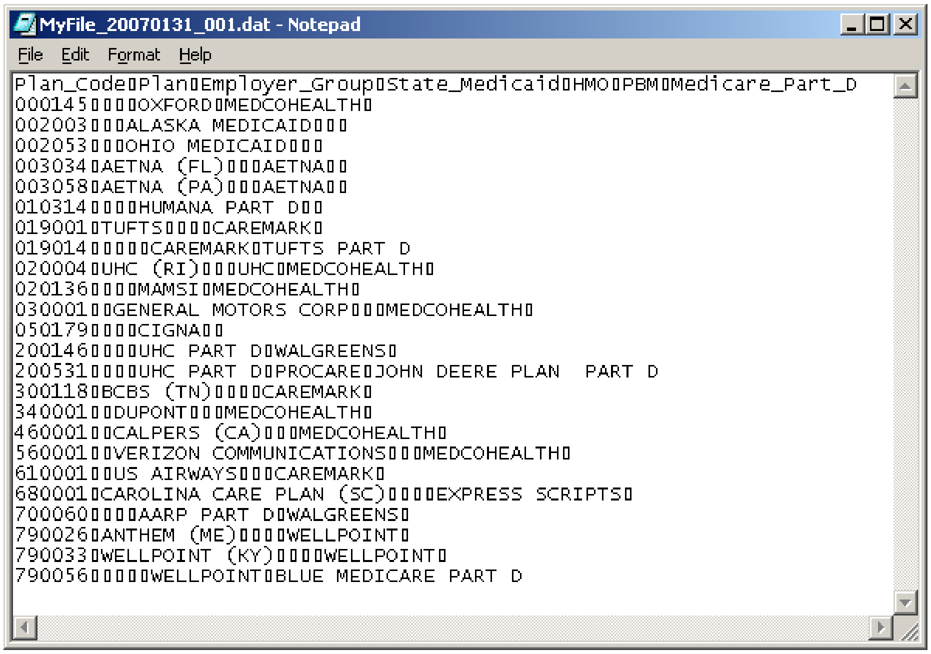
Thanks to the strategically-placed retain statement, data set **Final** will be in the column order desired, but will contain the data from the current file. A simple **%Let** statement will assign a value to the SAS Macro variable **filename**. As our monthly process documentation will warn us to set this value before running our program, maintenance of the correct name/naming convention is supported by our SOP/process list. We use this macro variable to set the value of the name of the file we are exporting in our **PROC EXPORT** step. To refer to the macro variable (as the latter part of the file name we are defining on the **outfile** option) we add an **ampersand** (&) prefix to the variable name. For the macro variable substitution to occur, we enclose the full directory/path/file name within double-quotes. Finally, we select the requested delimiter from our hand ASCII reference table. The program might look as follows:

TASK 3B – PROGRAM TO SET OUTPUT FILE NAME, SELECT DELIMITER, AND EXPORT FILE

```
%let filename = 2007_0131_001.dat; /** set value of macro variable filename **/
                          ↓
proc export data=final
   outfile= “C:\MyFile_&filename” /** call value of macro variable filename  **/
   dbms=dlm;                      /** export as a delimited flat/text file   **/
   delimiter=’1b’x;               /** use ASCII 27 as delimiter              **/
run;
                           ↓                          /** expected log messages **/
NOTE: C:\MyFile_2007_0131_001.dat was successfully created.
                                                /** file size/network dependent **/
NOTE: PROCEDURE EXPORT used: real time  8.98 seconds
                             cpu time   0.54 seconds
```

and the output – a flat file/text file opened up from our hard drive using Notepad -- looks like this:

TASK 3 – OUTPUT – THE MONTHLY FLAT FILE EXTRACT, DESIRED COLUMN ORDER, FILE NAME, AND DELIMITERS



```
Plan_Code▯Plan▯Employer_Group▯State_Medicaid▯HMO▯PBM▯Medicare_Part_D
000145▯▯▯▯▯OXFORD▯MEDCOHEALTH▯
002003▯▯▯ALASKA MEDICAID▯▯▯
002053▯▯▯OHIO MEDICAID▯▯▯
003034▯AETNA (FL)▯▯▯AETNA▯▯
003058▯AETNA (PA)▯▯▯AETNA▯▯
010314▯▯▯▯HUMANA PART D▯▯
019001▯TUFTS▯▯▯▯CAREMARK▯
019014▯▯▯▯▯CAREMARK▯TUFTS PART D
020004▯UHC (RI)▯▯▯UHC▯MEDCOHEALTH▯
020136▯▯▯▯MAMSI▯MEDCOHEALTH▯
030001▯▯GENERAL MOTORS CORP▯▯▯MEDCOHEALTH▯
050179▯▯▯▯CIGNA▯▯
200146▯▯▯▯UHC PART D▯WALGREENS▯
200531▯▯▯▯UHC PART D▯PROCARE▯JOHN DEERE PLAN  PART D
300118▯BCBS (TN)▯▯▯▯CAREMARK▯
340001▯▯DUPONT▯▯▯MEDCOHEALTH▯
460001▯▯CALPERS (CA)▯▯▯MEDCOHEALTH▯
560001▯▯VERIZON COMMUNICATIONS▯▯▯MEDCOHEALTH▯
610001▯▯US AIRWAYS▯▯▯CAREMARK▯
680001▯CAROLINA CARE PLAN (SC)▯▯▯▯EXPRESS SCRIPTS▯
700060▯▯▯▯AARP PART D▯WALGREENS▯
790026▯ANTHEM (ME)▯▯▯▯WELLPOINT▯
790033▯WELLPOINT (KY)▯▯▯▯WELLPOINT▯
790056▯▯▯▯▯WELLPOINT▯BLUE MEDICARE PART D
```

CONCLUSION

We have described here several techniques allowing us to achieve our business requirements while helping to minimize the programming maintenance effort. We have queried the SAS data set metadata and manipulated these SAS data sets using our usual SAS tool kit. Once our analysis was complete (regarding how we want our final data to look), we employed certain dynamic programming techniques to let our data help write our programs. We introduced simple elements of the SAS macro language to help structure our monthly process and used PROC EXPORT with a specific delimiter option to create an extract file best-suited for use by our downstream users.

We can continue to run our process and create our own file on a monthly basis with sufficient flexibility to meet our particular needs. We can ALSO continue to output an extract version of our file for downstream users in a manner which will minimize the maintenance efforts required on our part as well as that of the downstream users. While your particular business and programming problems will likely not be exactly the same as ours, it is certain that some of these techniques may nevertheless prove invaluable in your work.

REFERENCES

Art Carpenter & Ronald J. Fehd, 2007, "List Processing Basics: Creating and Using Lists of Macro Variables," **Proceedings of the NorthEast SAS Users Group Conference**. CD ROM Paper HW02.

Neil Howard, 2002, "Data Step Essentials," **Proceedings of the Twenty-Seventh Annual SAS Users Group International Conference**, 27. CD ROM. Paper 50.

SAS Institute Inc., **SAS® Macro Language: Reference**, First Edition, Cary, NC: SAS Institute Inc., 1997.

SAS Institute Inc., **SAS/ACCESS® 9.1 Interface to PC Files: Reference**, Cary, NC: SAS Institute Inc., 2004.

Selected ASCII Resources

ASCII Code Table (one of many sources)
http://www.ascii.cl/

Extended Example of Using ASCII Codes (SAS Support Web Site)
Sample 721: Code escape sequences into TITLE and FOOTNOTE statements
http://support.sas.com/ctx/samples/index.jsp?sid=721&tab=code

CONTACT INFORMATION

Your comments and questions are valued and encouraged. You may contact the author at:

John Cohen
Advanced Data Concepts, LLC
Newark, DE
jcohen1265@aol.com
(302) 559-2060