Paper 227-27

# Exporting Large SAS® Data Sets to MVS External Files in a  Production Setting

Helen-Jean Talbott
CitiFinancial, Credit Policy Department, Baltimore, MD, USA

## ABSTRACT

Sometimes external files provide the easiest method to exchange data. Our Credit Policy department relies heavily upon SAS® software for data management, analysis, and reporting. Frequently we need to export data from our SAS data sets to MVS external files in order to send data to another company that does not have SAS capabilities. SAS FILE and PUT statements work very well for creating external files, but our files are large (more than 300 variables). We want to avoid writing specific code for each variable and we need to implement a solution in production that is flexible for handling SAS files where variables change frequently. This paper shows how PROC CONTENTS and the macro language can simplify this task. The end product is an external fixed-format MVS file containing all the variables from the SAS data set, written in alphabetical order, along with supporting documentation showing the final file structure.

## INTRODUCTION

Our Credit Policy Department uses SAS software to manage and analyze tremendous amounts of information stored in SAS data sets. At times, we have to send data to other organizations or companies and we are delighted when the recipients of our data also use SAS software because it makes our job so much easier. In situations where the recipient company does not have SAS resources, we have to find another method for writing the data so that they can read the information. Common solutions are writing data from the SAS data set to structured external files such as dBASE and EXCEL. If these options are not available, our next choice is to write data to an external text file (which will be called a "flat file" for the duration of this paper).

SAS FILE and PUT statements work very well for creating flat files. The FILE statement specifies the output file where the data will be written. There are many options available to use with the FILE statement to control how the output file is written. The PUT statement writes lines (which can contain variable values and character strings) to the output file specified in the FILE statement. A typical program might look like the following:

```
DATA _NULL_;
   SET ACCOUNTS;
   FILE OUT;
   PUT OFFICE   $6.
       ACCOUNT  $7.
       CUSTNAME $25.
       LOANTYPE $2.
       BALANCE  10.2
       BOOKDATE MMDDYY8.;
RUN;
```

The PUT statement in the above example uses a format list where the variable is specified and is followed by the desired format. This method works well and is easy to implement when the data sets have relatively few variables.

The above method is not easy to implement for our large files that have more than 300 variables. The FILE and PUT statements work fine, but writing specific code for each variable is time-consuming and subject to error. We also needed to implement a solution in production that is flexible for handling SAS files where variables change frequently.

This paper shows how PROC CONTENTS and the macro language can simplify this task. The end product is an external fixed-format MVS file containing all the variables from the SAS data set, written in alphabetical order, along with supporting documentation showing the final file structure.

## EXPLANATION OF PROGRAM

The following program was developed using SAS release 8.1 on a mainframe computer running under MVS. JCL for this program includes DD statements specifying the input file and two output files. The input file (DDname = IN1) for this example is a SAS data set containing information on all the real estate accounts open at the end of the month. There are 383 variables and thousands of observations. Variables are a mixture of numeric and character and contain information such as customer name, address, loan amount, various dates, interest rates, and payment information. There are two output files produced: (DDname = OUT1) the flat file containing data derived from the input SAS file, and (DDname = OUT2) a flat file containing the record layout for the first output file. LRECL and SPACE parameters should be set to match the files being created. LRECL=4000 for OUT1 serves as a default value for this example and is based on the number and length of variables in the input file.

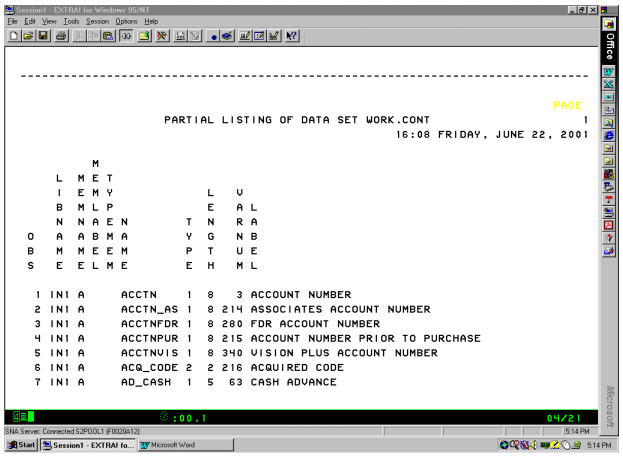
```
--- more JCL statements ---

//IN1      DD DSN=REOPEN.THISMON.DATA,
//            DISP=SHR
//OUT1     DD DSN=EXPORT.DATA,
//            DCB=(RECFM=FB,LRECL=4000),
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=DISK,
//            SPACE=(CYL,(400,200),RLSE),
//            LABEL=RETPD=90
//OUT2     DD DSN=EXPORT.INFO,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=DISK,
//            SPACE=(CYL,(1,1),RLSE),
//            LABEL=RETPD=90
//SYSIN   DD *
*******************************************;
* COP.PROD.CNTL(EXPORT)                   *;
*******************************************;
* IN1      DD DSN=REOPEN.THISMON          *;
* OUT1     DD DSN=EXPORT.DATA             *;
* OUT2     DD DSN=EXPORT.INFO             *;
*                                         *;
*                        DEC 20, 2001     *;
*                        HELEN-JEAN TALBOTT *;
*                                         *;
* PRODUCE FLAT FILE AND DOCUMENTATION FROM *;
* SAS FILE                                *;
*                                         *;
*******************************************;
```

The first step takes advantage of the output SAS data set created by PROC CONTENTS. The option NOPRINT turns off the output normally printed by PROC CONTENTS. For this example the output data set created by PROC CONTENTS is named CONT. There is one observation in data set CONT for each variable in data set IN1.A. The observations are in alphabetical order for the variable NAME which contains the names of the variables in data set IN1.A. The variables in data set CONT (see figure 1) contain information about the variables in IN1.A (such as the variable label and informat) and the data set (such as the date the data set was created). The following variables in data set CONT are used later in the program:

| | |
|---|---|
| FORMAT | variable format |
| LENGTH | variable length |
| NAME | variable name |
| TYPE | type of variable (1=numeric, 2=character) |
| VARNUM | variable number in the data set |

```
   PROC CONTENTS DATA=IN1.A OUT=CONT NOPRINT;
```



```
----------------------------------------------------------------------------
                                                                        PAGE
                      PARTIAL LISTING OF DATA SET WORK.CONT                1
                                                       16:08 FRIDAY, JUNE 22, 2001

        M
  L   M E T
  I   E M Y          L   V
  B   M L P          E   A L
  N   N A E N     T  N   R A
O A   A B M A     Y  G   N B
B M   M E E M     P  T   U E
S E   E L M E     E  H   M L

 1 IN1 A    ACCTN    1  8   3 ACCOUNT NUMBER
 2 IN1 A    ACCTN_AS 1  8 214 ASSOCIATES ACCOUNT NUMBER
 3 IN1 A    ACCTNFDR 1  8 280 FDR ACCOUNT NUMBER
 4 IN1 A    ACCTNPUR 1  8 215 ACCOUNT NUMBER PRIOR TO PURCHASE
 5 IN1 A    ACCTNVIS 1  8 340 VISION PLUS ACCOUNT NUMBER
 6 IN1 A    ACQ_CODE 2  2 216 ACQUIRED CODE
 7 IN1 A    AD_CASH  1  5  63 CASH ADVANCE
```
Session1 - EXTRA! for Windows 95/NT
File Edit View Tools Session Options Help
:00.1
04/21
SNA Server: Connected S2POOL1 (F0020A12)
5:14 PM
Start
Session1 - EXTRA! fo...
Microsoft Word
Office
Microsoft

Figure 1. Partial listing of data set WORK.CONT.

The second step modifies the format information based upon knowledge of values in the input data set and the way data should be expressed in the final flat file. In this example, dates will be expressed as mmddyyyy, social security numbers will be expressed as 999-99-9999 with dashes, and several fields such as account numbers and offices will be expressed as integers. Formats for other character variables are based on the variable length and formats for other numeric variables are set to 13.2.

```
 DATA CONT;
      LENGTH FORMAT $12;
      SET CONT END=LASTOBS;


      *** FORMAT CHANGES ***;
      IF FORMAT = 'MMDDYY' THEN DO;
         LENGTH = 10;
         FORMAT = 'MMDDYY10.';
         END;

      IF FORMAT IN ('SSN','P_SSN') THEN DO;
         LENGTH = 11;
         FORMAT = 'SSN11.';
         END;

      IF NAME ='ACCTNFDR' THEN DO;
         LENGTH = 16;
         FORMAT = '16.0';
         END;

      IF NAME IN('AD_TRUE','BALANCE','CO_AMT',
         'CR_HIGH','CR_LIM','PHONE_NO','ZIP',
         'ACCTN', 'ATP') THEN DO;
         LENGTH = 11;
         FORMAT = '11.0';
         END;

      IF FORMAT = ' ' THEN DO;
         IF TYPE = 2 THEN
         FORMAT = COMPRESS('$' || LENGTH
            || '.');
         ELSE IF TYPE = 1 THEN DO;
            LENGTH = 13;
            FORMAT = '13.2';
            END;
         END;
```

The number of variables in the input data set is determined by examining the variable number (VARNUM) and retaining the maximum value in the variable MAXVAR. The SYMPUT function then assigns this value to the macro variable (also called MAXVAR) so that it is available for other procedures later in the program.

```
      *** DETERMINE NUMBER OF VARIABLES ***;
      RETAIN MAXVAR 0;
      IF _N_ = 1 THEN MAXVAR=VARNUM;
      ELSE MAXVAR = MAX(MAXVAR,VARNUM);
      IF LASTOBS THEN DO;
         CALL SYMPUT('MAXVAR',MAXVAR);
         END;
      DROP MAXVAR;
```

The third step determines the position (POSNO) for each variable. The position for the first variable is set to 1. The position for each successive variable is determined by adding the value of the variable LENGTH. This is accomplished through the SUM statement (POSNO + LENGTH). POSNO increases as each observation in data set CONT (thus each variable for the final flat file) is processed. The other important feature of DATA step DSNINFO is that the variable names and formats are assigned to macro variables (such as V1 and F1 for the first variable and corresponding format) to make it easy to write values to the final flat file later in the program.

```
   DATA DSNINFO;
      SET CONT;
      RETAIN X 0
      POSNO 0;
      X+1;
      IF _N_ = 1 THEN POSNO = 1;
      CALL SYMPUT('V' || LEFT(PUT(X,3.)),NAME);
      CALL SYMPUT('F' || LEFT(PUT(X,3.)),
         FORMAT);
      OUTPUT;
      POSNO + LENGTH;
   RUN;
```

The fourth step defines a macro called VN. It contains an iterative %DO statement for writing values for each of the variables to the output data set. When invoked later in the program, the expressions &&V&I and &&F&I will resolve to macro variables such as &V1 and &F1 that were created above. The code &&V&I &&F&I executes repeatedly until the index variable "I" reaches the value represented by &MAXVAR which is the number of variables in the data set.

```
 %MACRO VN;
    %DO I=1 %TO &MAXVAR;
      &&V&I &&F&I
    %END;
 %MEND VN;
```

The last step in writing the flat data file uses a DATA _NULL_ step in combination with the FILE statement and the PUT statement. The SET statement reads in the input data set REOPEN. The FILE statement specifies the output file to use for the PUT statement that follows. LRECL=4000 specifies the logical record length of the output file and agrees with the value used in the JCL discussed above. %VN invokes the VN macro and provides a very easy way to write values for each variable for each observation.

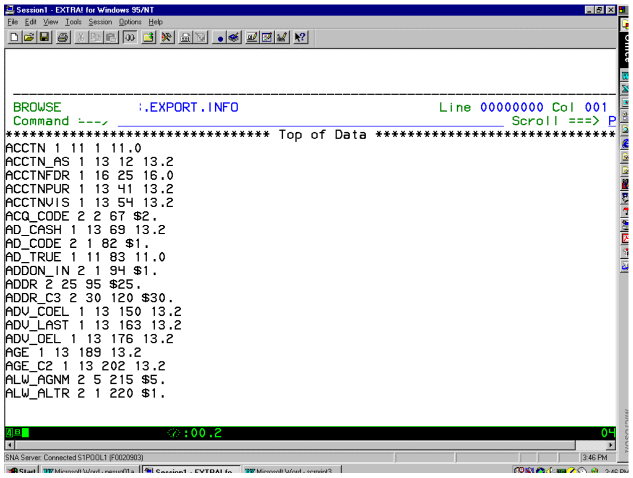
```
 DATA _NULL_;
    SET IN1.A;
    FILE OUT1 LRECL=4000;

    PUT %VN
   ;
 RUN;
```

The final portion of the program produces a flat file containing the file layout for the file created above. This process uses a DATA _NULL_ step in conjunction with the FILE and PUT statements. The PUT statement writes the variable name, variable type, length of the variable, position, and format based on the information stored in data set WORK.DSNINFO created above. Variables are presented in alphabetical order (see figure 2). Records are written to the file specified by DDNAME OUT2 in the JCL.

```
DATA _NULL_;
   SET DSNINFO;
   FILE OUT2;

   PUT NAME TYPE LENGTH POSNO FORMAT
         ;
RUN;
```


Session1 - EXTRA! for Windows 95/NT
File Edit View Tools Session Options Help

```
BROWSE           :.EXPORT.INFO                    Line 00000000 Col 001
Command ===>                                              Scroll ===> P
********************************* Top of Data ******************************
ACCTN 1 11 1 11.0
ACCTN_AS 1 13 12 13.2
ACCTNFDR 1 16 25 16.0
ACCTNPUR 1 13 41 13.2
ACCTNVIS 1 13 54 13.2
ACQ_CODE 2 2 67 $2.
AD_CASH 1 13 69 13.2
AD_CODE 2 1 82 $1.
AD_TRUE 1 11 83 11.0
ADDON_IN 2 1 94 $1.
ADDR 2 25 95 $25.
ADDR_C3 2 30 120 $30.
ADV_COEL 1 13 150 13.2
ADV_LAST 1 13 163 13.2
ADV_OEL 1 13 176 13.2
AGE 1 13 189 13.2
AGE_C2 1 13 202 13.2
ALW_AGNM 2 5 215 $5.
ALW_ALTR 2 1 220 $1.
```

Figure 2 . Partial listing of external file EXPORT.INFO.

## CONCLUSION

Situations occur where the best method of delivering data to another organization or company is through external flat files. The FILE and PUT statements are extremely useful for writing data from SAS data sets to external files. For large files, the task of writing all of the variable names and formats needed in the PUT statement can be extremely time-consuming and is subject to errors. A better process is to automate this process by taking advantage of the PROC CONTENTS procedure and the MACRO language. PROC CONTENTS is applied to the SAS data set to produce an intermediate output data set containing information about the variable names and formats in the original SAS data set. Format information in this intermediate data set is modified based on knowledge of data in the original SAS data set (such as which fields should be integers and how long numeric fields should be in the final file). Variable names and formats are loaded into macro variables for use in an iterative %DO statement that avoids having to hardcode each variable name and format. Length of the variables controls the position for writing the variable in the final data set. The end product is an external fixed-format MVS file containing all the variables from the SAS data set, written in alphabetical order, along with supporting documentation showing the final file structure.

## REFERENCES

SAS Institute Inc. (1990), *SAS Guide to Macro Processing, Version 6, Second Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), *SAS Language, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), *SAS Procedures Guide, Version 6, Third Edition*, Cary, NC: SAS Institute Inc.

## ACKNOWLEDGMENTS

## CONTACT INFORMATION

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

Helen-Jean Talbott
CitiFinancial
300 St. Paul Place, BSP10A
Baltimore, Maryland 21202 USA

(410) 332-3849
FAX: (410) 332-2838

## ENTIRE PROGRAM LISTING

```
--- more JCL statements ---

//IN1      DD DSN=REOPEN.THISMON.DATA,
//            DISP=SHR
//OUT1     DD DSN=EXPORT.DATA,
//            DCB=(RECFM=FB,LRECL=4000),
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=DISK,
//            SPACE=(CYL,(400,200),RLSE),
//            LABEL=RETPD=90
//OUT2     DD DSN=EXPORT.INFO,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=DISK,
//            SPACE=(CYL,(1,1),RLSE),
//            LABEL=RETPD=90
//SYSIN   DD *
*********************************************;
* COP.PROD.CNTL(EXPORT)                     *;
*********************************************;
* IN1      DD DSN=REOPEN.THISMON            *;
* OUT1     DD DSN=EXPORT.DATA               *;
* OUT2     DD DSN=EXPORT.INFO               *;
*                                           *;
*                        DEC 20, 2001       *;
*                        HELEN-JEAN TALBOTT *;
*                                           *;
* PRODUCE FLAT FILE AND DOCUMENTATION FROM *;
* SAS FILE                                 *;              ;
*                                          *;           RUN;
********************************************;

PROC CONTENTS DATA=IN1.A OUT=CONT NOPRINT;

DATA CONT;
   LENGTH FORMAT $12;
   SET CONT END=LASTOBS;

   *** FORMAT CHANGES ***;
   IF FORMAT = 'MMDDYY' THEN DO;
      LENGTH = 10;
      FORMAT = 'MMDDYY10.';
      END;

   IF FORMAT IN ('SSN','P_SSN') THEN DO;
      LENGTH = 11;
      FORMAT = 'SSN11.';
      END;

   IF NAME ='ACCTNFDR' THEN DO;
      LENGTH = 16;
      FORMAT = '16.0';
      END;

   IF NAME IN('AD_TRUE','BALANCE','CO_AMT',
      'CR_HIGH','CR_LIM','PHONE_NO','ZIP',
      'ACCTN', 'ATP') THEN DO;
      LENGTH = 11;
      FORMAT = '11.0';
      END;

   IF FORMAT = ' ' THEN DO;
      IF TYPE = 2 THEN
      FORMAT = COMPRESS('$' || LENGTH || '.');
      ELSE IF TYPE = 1 THEN DO;
         LENGTH = 13;
         FORMAT = '13.2';
         END;
      END;

   *** DETERMINE NUMBER OF VARIABLES ***;
   RETAIN MAXVAR 0;
   IF _N_ = 1 THEN MAXVAR=VARNUM;
   ELSE MAXVAR = MAX(MAXVAR,VARNUM);
   IF LASTOBS THEN DO;
      CALL SYMPUT('MAXVAR',MAXVAR);
      END;
   DROP MAXVAR;

DATA DSNINFO;
   SET CONT;
   RETAIN X 0
   POSNO 0;
   X+1;
   IF _N_ = 1 THEN POSNO = 1;
   CALL SYMPUT('V' || LEFT(PUT(X,3.)),NAME);
   CALL SYMPUT('F' || LEFT(PUT(X,3.)),
        FORMAT);
   OUTPUT;
   POSNO + LENGTH;
RUN;

%MACRO VN;
   %DO I=1 %TO &MAXVAR;
      &&V&I &&F&I
      %END;
%MEND VN;

DATA _NULL_;
   SET IN1.A;
   FILE OUT1 LRECL=4000;

   PUT %VN
    ;
 RUN;

 DATA _NULL_;
    SET DSNINFO;
    FILE OUT2;

    PUT NAME TYPE LENGTH POSNO FORMAT
```