%FLATFILE, and Make Your Life Easier

M. Michelle Buchecker, SAS Institute Inc., Chicago, IL

ABSTRACT

%FLATFILE is a macro that will create a flat file from a SAS® data set. This macro takes 3 parameters: the SAS data library name, the SAS data set name, and the name of the flat file to create. The beauty of %FLATFILE is that it will query the data set to determine the variable names and write out the PUT statement INCLUDING the variables' permanent format (or a default if there is none).

Operating Systems: ALL
Version: SAS 6.07 or higher

INTRODUCTION

This paper discusses an easy way to create a fixed column flat file (ascii file, text file, whatever you want to call it) from a SAS data set. The DATA step provides the mechanism to create a flat file from a SAS data set, but has a few drawbacks. You must

- know the names of the variables you want to write
- specify the format to write for each variable
- know how to write the DATA step code
- calculate the column widths to ensure there is no overlap.

The macro in this paper will do all of the above for you!

Let's take an in-depth look at each point and see how the macro solves the problem.

KNOW THE NAMES OF THE VARIABLES YOU WANT TO WRITE

Past Solutions

This problem is a bit more complex. If you have ever used the CONTENTS procedure or the DATASETS procedure, you know that either of these two procedures will produce the names of the variables from a data set in your log or output window. However, the problem is we would like to **capture** that information and put it inside of our DATA step code.

There have been a few macros in the past that have done exactly that. But, as you can imagine, they were fairly complicated, since they had to

1. re-direct output to a file
2. issue the PROC CONTENTS or PROC DATASETS code
3. read the file produced by the procedure
4. strip off the unwanted information (of which there was much)
5. carefully search for variable names
6. store those variable names into macro variables
7. write the DATA step.

THE DICTIONARY.COLUMNS TABLE

Starting with Version 6.07 of the SAS® System, you now have the SQL dictionary tables at your disposal. The SQL dictionary tables are essentially data sets about your data sets (and catalogs and all other SAS files). This macro uses the dictionary table named DICTIONARY.COLUMNS. The DICTIONARY.COLUMNS table (SQL data sets are called tables) contains information like

- data set name
- data set library
- variable name
- variable type
- variable length.

There is one row (row is the SQL term for observation) for each variable in each data set that your current SAS session knows about.

Selecting Variable Names, Types, Lengths, and Formats

❶The first step of the macro uses the SQL procedure to read the DICTIONARY.COLUMNS table for the data set and library that were supplied as parameters to the macro. Notice the WHERE clause (SQL statements are called clauses since there is no semicolon preceding each one). The column name (SQL variables are called columns) LIBNAME is a column in DICTIONARY.COLUMNS referring to a SAS data library name. The column name MEMNAME refers to members (data sets) in that data library.

By subsetting DICTIONARY.COLUMNS based on the library and data set, you obtain information just on that data set. The select clause extracts just the

- column name (NAME)
- column type (TYPE)
- column format (FORMAT)
- column length (LENGTH).

Remember, there is one row (observation) for each column (variable). These results are then stored in the SAS view (data set) WORK.TEMP.

Creating Macro Variables for Variable Names and Formats

❷ The DATA step reads the data set created by PROC SQL and creates a series of macro variables named

- VAR1-VAR*n*
- FMT1-FMT*n*.

n is the total number of variables in the data set.

To create these macro variables, CALL SYMPUT is used. CALL SYMPUT takes two arguments. The first argument is the name of the macro variable to create. The second argument is the value the macro variable will contain. For example, we want the value of the first macro variable, VAR1, to be the **name** of the first variable in the data set you want to write out. The macro variable FMT1 is that variables' permanent format. If the variable does not have a permanent format, a format of BEST10. is assigned.

❸ Since we know the macro variable name always starts with VAR, the word VAR is enclosed in quotes because it is constant. Next, we have to append a number/counter. The DATA step automatically contains a variable called _N_ that

counts the number of iterations through the DATA step. The *put* function changes the numeric counter, _N_, to the character representation of that number so there are no notes written to the log that SAS is converting it for us. The *left* function removes leading blanks in front of the number, so we don't try to create a macro variable named VAR 1. Then the two vertical bars concatenate the left justified number to the word VAR.

Notice the second argument to CALL SYMPUT is NAME, which is the name of the variable from DICTIONARY.COLUMNS. Since NAME is not in quotes, the DATA step has to assume that it is a variable in the Program Data Vector (PDV). During execution, the DATA step looks inside the PDV for the value of NAME and finds the name of the first variable.

SPECIFY THE FORMAT TO WRITE FOR EACH VARIABLE

❹ The next series of statements checks to see if that variable already has a format. If so, the first IF condition is true and a macro variable named FMT*n* is created whose value is the name of the format. If no format is assigned, the next statement checks to see if it is a character variable. A character variable has a value of CHAR in the TYPE column from DICTIONARY.COLUMNS. If the variable is character, a character format is created based on the length of that variable. For example, if the length of the character variable is 12, a format of $12. is created. If a variable is numeric and has no format, a format of BEST10. is assigned.

The DATA step continues to loop through, creating 2 macro variables for each variable in the original data set.

❺ Finally, it's **extremely** helpful to know how many variables are in the data set you want written out, so the last statement says if this is the last observation, create a macro variable named NUMVAR whose value is the total number of variables.

KNOW HOW TO WRITE THE DATA STEP CODE

The general form for writing a data set out to a flat file is:

```
DATA _NULL_ ;
  SET SAS-data-set ;
  FILE 'name-of-file-to-write' ;
  PUT column-pointer variable-name format ...;
RUN ;
```

When calling the macro, you will supply the name of the data set and the name of the flat file. The macro will then build the PUT statement for you.

The SET Statement

❻Notice the SET statement in the last DATA step. The macro variables LIB and DSN are supplied when calling the macro. So why are there two periods after &LIB? The macro facility treats all periods that follow a macro variable reference as a delimiter to end a macro variable name. This is useful if you have additional text that needs to be the suffix to the value of the macro variable. The period is then thrown away. If one period's not enough, use two! The first still works as a delimiter and gets thrown away, but the second period is then treated as text to separate the libref from the data set name.

The FILE Statement

❼ The FILE statement has double quotes around the macro variable reference &FILE to allow the macro variable to resolve. The macro facility does not "peek inside" single quotes.

The PUT Statement

❽ The word PUT only needs to be in the DATA step once, so it is outside of the %DO loop.

❾The %DO loop will execute for as many times as there are variables in the original data set, which is determined by &NUMVAR. The code that is generated is sent to the DATA step compiler. The index counter, I, in the %DO loop is a macro variable and will be used to cycle through the macro variables we created earlier (VAR1 - VAR*n*, FMT1 - FMT*n*).

❿ To retrieve the value from the macro variable VAR1, we need to precede the macro variable name with an ampersand (ie. &VAR1). However, since the number at the end is not always a 1, we need to substitute the 1 with our index counter, &I. So now we have &VAR&I. So the macro facility would scan looking for a macro variable called &VAR. It won't find one, and will generate a warning message. Then &I will resolve to 1. But the &VAR did not resolve properly.

So if one ampersand isn't enough, try two!
&&VAR&I
With multiple ampersands, the macro facility takes two ampersands and makes them one. The word VAR isn't a macro trigger, so it just tags along for the ride. &I the first time through the %DO loop resolves to 1. So now we have &VAR1. Exactly what we wanted. The macro facility then re-scans &VAR1, and that resolves to the name of the first variable in the data set.

The same is true for the format, but this time the word FMT tags along for the ride so we have &FMT1.

CALCULATE THE COLUMN WIDTHS TO ENSURE THERE IS NO OVERLAP

The +1 moves the column pointer over to the next field to put a space between this column and the next. Notice there is no semicolon inside the %DO loop. The %DO loop continues looping, writing out the variables on one PUT statement. After all of the variables have been written to the PUT statement, the %DO loop ends execution.

11 However, the PUT statement still does not have a semicolon to end that statement. That is what the semicolon on the line by itself accomplishes.

CALLING THE MACRO

Now, let's create a flat file on MVS named MYID.FLAT.FILE based on the data set SASUSER.HOUSES. The code to call the macro is:

```
%flatfile(lib=sasuser, dsn=houses,
file=myid.flat.file)
```

If you are using a directory-based system, you might code:

```
%flatfile(lib=sasuser, dsn=houses,
file=flat.dat)
```

CONCLUSION

The SAS macro facility provides a dynamic, maintenance free way to write your SAS code. By incorporating this capability with the SQL dictionary tables, you can create an extremely powerful tool to

- improve your productivity
- create libraries of shared and re-usable code
- decrease the likelihood of syntax and logic errors.

For more information on the SQL dictionary tables, please refer to *SAS ® Technical Report P-222, Changes and Enhancements to Base SAS® Software, Release 6.07* (order #C59139).

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

The Code

The macro code is:

```
options mprint;
%macro flatfile(lib=,dsn=,file=);
  %let lib=%upcase(&lib);  /* uppercase library and data set names */
  %let dsn=%upcase(&dsn);

❶proc sql;
      create view temp as
        select name, type, format, length
        from dictionary.columns
        where libname = "&lib" and memname = "&dsn";
    quit;

❷data _null_;
      set temp end=last;
❸      call symput ('var'!!left(put(_n_,3.)),name);
❹     if format ne ' ' then
        call symput ('fmt'!!left(put(_n_,3.)),format);
      else
        if upcase(type) = 'CHAR' then
          call symput ('fmt'!!left(put(_n_,3.)),'$'!!put(length,3.)!!'.');
        else
          call symput ('fmt'!!left(put(_n_,3.)),'best10.');

❺    if last then call symput('numvar',left(put(_n_,3.)));

    data _null_;
❻    set &lib..&dsn;
❼    file "&file";
❽    put
❾    %do i = 1 %to &numvar;
❿      &&var&i &&fmt&i +1
      %end;
❶❶    ;   /* end put statement */
    run;
 %mend;
```

The Log

Here is an example log run under Windows (formatted slightly to fit the page):

```
578   %flatfile(lib=sasuser, dsn=houses, file=f:\sugi\flat.dat)
MPRINT(FLATFILE):   PROC SQL;
```

```
MPRINT(FLATFILE):   CREATE VIEW TEMP AS SELECT NAME, TYPE, FORMAT, LENGTH
                    FROM DICTIONARY.COLUMNS WHERE LIBNAME = "SASUSER"
                    AND MEMNAME = "HOUSES";
NOTE: SQL view WORK.TEMP has been defined.
MPRINT(FLATFILE):   QUIT;
NOTE: The PROCEDURE SQL used 0.55 seconds.


MPRINT(FLATFILE):   DATA _NULL_;
MPRINT(FLATFILE):   SET TEMP END=LAST;
MPRINT(FLATFILE):   CALL SYMPUT ('var'!!LEFT(PUT(_N_,3.)),NAME);
MPRINT(FLATFILE):   IF FORMAT NE ' ' THEN
                      CALL SYMPUT ('fmt'!!LEFT(PUT(_N_,3.)),FORMAT);
MPRINT(FLATFILE):   ELSE IF UPCASE(TYPE) = 'CHAR' THEN
             CALL SYMPUT ('fmt'!!LEFT(PUT(_N_,3.)),'$'!!PUT(LENGTH,3.)!!'.');
MPRINT(FLATFILE):   ELSE CALL SYMPUT ('fmt'!!LEFT(PUT(_N_,3.)),'best10.');
MPRINT(FLATFILE):   IF LAST THEN CALL SYMPUT('numvar',LEFT(PUT(_N_,3.)));

NOTE: The DATA statement used 1.04 seconds.


MPRINT(FLATFILE):   DATA _NULL_;
MPRINT(FLATFILE):   SET SASUSER.HOUSES;
MPRINT(FLATFILE):   FILE "f:\sugi\flat.dat";
MPRINT(FLATFILE):   PUT STYLE $ 8. +1 SQFEET BEST10. +1 BEDROOMS BEST10. +1
                        BATHS BEST10. +1 STREET $ 16. +1 PRICE DOLLAR12. +1 ;
MPRINT(FLATFILE):   RUN;

NOTE: The file "f:\sugi\flat.dat" is:
      FILENAME=f:\sugi\flat.dat,
      RECFM=V,LRECL=256

NOTE: 15 records were written to the file "f:\sugi\flat.dat".
      The minimum record length was 71.
      The maximum record length was 71.
NOTE: The DATA statement used 1.1 seconds.
```

The Output

Here is the resulting file:

```
RANCH          1250          2          1 Sheppard Avenue       $64,000
SPLIT          1190          1          1 Rand Street           $65,850
CONDO          1400          2        1.5 Market Street         $80,050
TWOSTORY       1810          4          3 Garris Street        $107,250
RANCH          1500          3          3 Kemble Avenue         $86,650
SPLIT          1615          4          3 West Drive            $94,450
SPLIT          1305          3        1.5 Graham Avenue         $73,650
CONDO          1390          3        2.5 Hampshire Avenue      $79,350
TWOSTORY       1040          2          1 Sanders Road          $55,850
CONDO          2105          4        2.5 Jeans Avenue         $127,150
RANCH          1535          3          3 State Highway         $89,100
TWOSTORY       1240          2          1 Fairbanks Circle      $69,250
RANCH           720          1          1 Nicholson Drive       $34,550
TWOSTORY       1745          4        2.5 Highland Road        $102,950
CONDO          1860          2          2 Arcata Avenue         $110,700
```