Paper FS-08-2013

# Changing a Static Condition to a Dynamic Data-Driven Field with SAS®

Misty Johnson, State of Wisconsin Department of Health Services, Madison, WI

## ABSTRACT

SAS® programs that are data driven are efficient and reduce the possibility of error. The design of SAS programs should be carefully considered to incorporate the ease of use and ability to accommodate future change. This paper describes the edit of a SAS program to change a static condition to a dynamic, data-driven variable. Also discussed is the effect of the edit upon the business process and editing SAS programs to invoke minimum process change. The SAS program demonstrates the use of macro variables assigned with the %LET statement, the LIBNAME statement with an Excel engine and the mixed option, literals when referring to Excel sheet names and the creation of output text files with the FILE statement. Methods described in this paper use base SAS and the Access to PC Files module and is geared toward beginning to intermediate SAS users.

## INTRODUCTION & BACKGROUND

The Wisconsin Department of Health Services (DHS) contracts with more than 600 providers for social service programs, with annual contracts in excess of $670 million. The Community Aids Reporting System (CARS) is used to collect and store contract, expense and payment information. Provider payments, based upon contracts and expenses, are paid via check or direct deposit, also known as an Automated Clearing House (ACH) deposit.

CARS is a reporting system not a payment system, therefore, SAS is used to generate an interface file from CARS data that is sent to the bank to generate checks or direct deposit payments. The interface file contains data in a prescribed layout detailing the payment information and the receiving account, which has always been a checking account. When a vendor recently requested to change their receiving account to a savings account, it required edits to the SAS program as well as the input file.

This paper will explain these edits to change a static transaction code to a dynamic data-driven variable from the input file. This paper will also explain why this more lengthy solution was chosen over a quick solution within SAS.

## THE SAS PROGRAM

The monthly run of the CARS provides information on payments by vendor in the form of a flat file. Vendors wishing to receive payment via direct deposit submit their account and routing information to DHS via a paper form that is hand-keyed into a tab-delimited file. The flat file of payment information and the tab-delimited file of vendor account and routing information are used by the SAS program to create the interface file sent to the bank to create direct deposit payments.

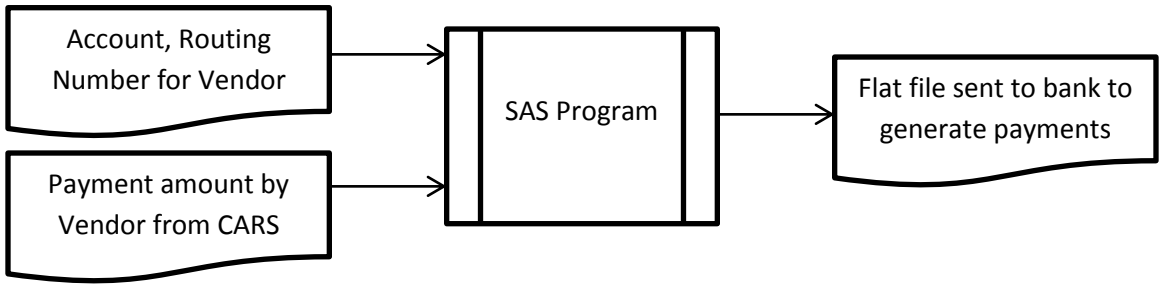


**Figure 1. Overview of the SAS program inputs and output.**

The interface file created by the SAS program contains data in a prescribed layout detailing the payment amount, bank routing numbers and account numbers for the vendor, plus other information such as transaction code, which determines what type of account will receive the payment.

Direct deposit payments have always been made to the vendor's checking account. Recently, a vendor requested to have the direct deposit payment routed to their savings account instead. The SAS program that generates the output interface file needed to be edited to show a new transaction code value to enable the payment to deposit to a different type of account.

## THE STATIC MACRO VARIABLE

The transaction code is assigned at the top of the program as a global macro variable that remains static. A benefit of assigning the transaction code as a macro variable as opposed to a hard-coded value within the code, is that if it ever needed changing it could easily be changed in only one place in the program, minimizing errors and ensuring consistency.

Later in the code, the global macro variable, which is in effect for the duration of the SAS session or until it is changed by the user, is concatenated with a static required character, and the vendor's routing number and account number, which results in a text string. The text string is written to the interface file. The assignment and use of the transaction code is shown below:

```
%LET MTRANSCD = '22';    *ASSIGN STATIC TRANSACTION CODE;
……(SAS code) ……
*CONSTRUCTION OF TEXT STRING IN A LATER DATA STEP;
DATA DETAIL3;
…
RECORD3 = '6'||&MTRANSCD.||BANKNUM||ACCTNUM;
…
```

The SAS coding shown above demonstrates a work-around of the formatting used by SAS on a macro variable assigned with the %LET statement. When you attempt to assign a numeric macro variable with %LET statement, SAS first applies the BEST12. format and then converts the numeric to text. SAS then stores this macro variable, which may include blanks, as a text. This can become a problem if the leading spaces are not desired or expected. One solution is to use the STRIP function to remove both leading and trailing blanks. The coding above utilizes a matched pair of single quotes to define the value of MTRANSCD as a literal of only two characters; therefore, SAS doesn't have to apply the BEST12. format before it converts it to a character, thus, no extra spaces are stored with the macro variable.

The variables BANKNUM and ACCTNUM are from the tab-delimited file that specified these values for each vendor. Unfortunately, the value of the transaction code is in effect for all vendors about to receive ACH payment. My goal is to change the transaction code for only one vendor, not all. In the future, I may need to change the transaction code for other vendors as well.

## HOW TO BEST EDIT: A SAS-ONLY SOLUTION

This problem could have easily been solved by the simple addition of an IF-THEN-ELSE condition to accommodate the transaction code wishes of one or more vendors. I could have specified a listing of vendors wishing to receive payment into a savings account at the top of the program, for use by the IF-THEN-ELSE condition. The SAS-only solution is demonstrated below:

```
%LET SAVEACCT = ('123546', '555888'); *VENDORS WANTING SAVINGS ACCT DEPOSIT;
%LET MTRANSSV = '32';                 *TRANSACTION CODE FOR SAVINGS ACCT;
%LET MTRANSCK = '22';                 *TRANSACTION CODE FOR CHECKING ACCT;
……(SAS code) ……
IF VENDNO IN &SAVEACCT. THEN
       RECORD3 = '6'||&MTRANSSV.||BANKNUM||ACCTNUM;
ELSE   RECORD3 = '6'||&MTRANSCK.||BANKNUM||ACCTNUM;
```

This solution wouldn't require any restructuring of the input data of bank numbers and routing numbers by vendor. However, if another vendor in the future also wanted payments routed to their savings account, a SAS programmer would be needed to add the vendor number to the global macro variable assignment statement in SAS. This would require a process change, further requiring documentation and training for all staff involved. Requiring action of a SAS programmer adds the further complication of dependency upon a limited number of SAS programmers in the unit.

SAS is powerful and data speaks for itself. So why not let the data simply tell SAS its requirements?

## A BETTER SOLUTION: A DATA-DRIVEN VARIABLE

The designation of the savings or checking account has always been a part the vendor form; it was collected, but not used. Why collect information if you don't use it? The tab-delimited file containing the account and routing number by vendor was modified to include a new variable named Trans_Code, which contained a text value to specify account type; currently either checking (value of '22') or savings (value of '32'). In theory, other types of accounts

could be added in the future if necessary. A fictitious example of the Microsoft Excel® file containing account and routing number information by vendor is shown below.

| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | OBS | IGP | Muni_Code | Agency_No | Vendor_Name | Last | First | Phone | ank_Routing_N | Bank_Account_No | Trans_Code |
| 2 | 2 | 1 | 01000 | 1 | TREAS ADAMS CO | Smith | Mary | 555-5554 | 123456789 | 456789 | 22 |
| 3 | 5 | 0 | 10000 | 10 | TREAS CLARK CO | Jones | Mike | 555-6666 | 999921111 | 100200 | 32 |
| 4 | 284 | 0 | 92003 | 100537 | Mom's Child Care | Cleaver | June | 555-7777 | 010101010 | 2223334444 | 22 |
| 5 | 285 | 0 | 55371 | 100552 | Pleasantville School Distr | Bell | Lucille | 555-9999 | 075875075 | 2202202200 | 22 |

**Figure 2. Excel file input to SAS.**

This solution also means a process change for the data-entry staff. However, it is a small addition to the information they already enter, and unlike the SAS-only solution, requires the action of no other staff member. Also taken into consideration when adding this variable to the tab-delimited file, was the placement of this new column. The column was placed to follow the flow of the form containing the input information. The first field on the form should correspond to the first column on the input file, etc.

This solution is a minor change for the data entry staff member and also makes practical sense. The input tab-delimited file now contains all the vendor information the SAS job needs to create an output interface file. Before, the vendor input told SAS only the account and routing numbers, and left SAS to assign the account type. The SAS job was edited to receive this new variable from the input file.

The code to declare the inputs and output of the SAS job is shown below.

```
%LET MCHEK    = '<FILEPATH>\EMV90318.DTA';  /* INPUT FLAT FILE FROM CARS */
%LET MACHREC  = '<FILEPATH>\VO90318E.ACH'; /* OUTPUT INTERFACE FILE, ACH PAYMNT*/
LIBNAME ACHFILE EXCEL "<Filepath>\VendorACH.xls" mixed=yes;
```

Earlier, I used a %LET statement to declare a macro variable to hold a text value to set the transaction type. Here, a %LET statement is used to declare a text string to determine the location of the input flat file created by the CARS containing payment data by vendor, and the location where the output interface file will be written, containing all the information the bank needs to generate ACH payments for the vendors. A libname is established for the location of the tab-delimited input file of ACH information by vendor. This Excel file is updated as necessary by CARS staff, and must be imported each time. This job is made easier with use of a libref utilizing an Excel engine to specify where SAS will find the input data. Notice the use of the *mixed* option. This option is necessary to allow SAS to bring in columns from Excel that could contain both numeric and text data. If this option is not declared, SAS would discard any text data found in a column that is mostly numeric, or any numeric data found in a column that is mostly text (Heaton, 2006). In theory, the Excel input data would have columnar data that was all the same format. However, it is possible that an entry could be formatted differently than the column; use of the mixed option ensures that SAS will not drop data that is different than others in the same column.

The code below is the DATA step that imports the current ACH file.

```
DATA ACHINFO
     (DROP=TRANS_CODE   RENAME=(TRANS_CODET=TRANS_CODE));
SET ACHFILE."ACH_Listing$"N;
* MAKE SURE TRANS_CODE IS A TEXT, SET TO DEFAULT OF 22 IF MISSING;
IF TRANS_CODE THEN TRANS_CODET=PUT(TRANS_CODE,2.);  /*NEW VARIABLE*/
ELSE TRANS_CODET='22';  /*DEFAULT VALUE*/
RUN;
```

The ACH file is imported via a libref; the two part name is the libref followed by the tab name in Excel. Notice the use of the name literal in the SET statement; this is required because tab names in Excel do not follow the same naming conventions as SAS. Tab names in Excel always end with a dollar sign ($) that is not visible in Excel. Further, tab names may contain spaces, special characters, or start with a number, all of which are not normally allowed in SAS (Heaton, 2006).

I use a PUT statement to create a new variable named Trans_CodeT to represent the Trans_Code variable converted to a character of length two. If this precaution weren't taken, SAS could later do the data conversion for me, but I prefer to explicitly do the conversion. In a perfect world, the transaction code would be left alone in Excel with its text formatting, and need no special attention when it comes to SAS. However, multiple people in the unit hand-key information into the Excel file, and if the column were formatted as general or numeric, this extra care is indeed necessary. After ensuring that the Trans_Code variable is a text, I drop the original variable and rename the created variable Trans_CodeT to Trans_Code.

In the coding below, the flat file of payment data by vendor is imported via an INFILE statement with the MISSOVER option to tell SAS to skip over any missing fields it may find at the end of each row. The INFILE statement tells SAS exactly where it will find the variables in this flat file and their informat. The ACH file and the payment file share a variable named MUNICD, which is the municipal code for the vendor. The two files will be merged on this common variable; in preparation, each file is sorted on the common variable with a PROC SORT statement.

```
*BRING IN PAYMENT DATA BY VENDOR FROM FLAT FILE;
DATA CHECKS;
INFILE &MCHEK. MISSOVER;       /* EMVXXXXX.DTA - FLAT FILE FROM CARS */
INPUT @ 7 MUNICD $ 5.
      @ 22 PAYEE $ 22.
      @ 54 AMOUNT COMMA13.0;
RUN;

* SORT PAYMENT FILE AND ACH FILE BY MUNI CODE;
PROC SORT DATA=CHECKS;  BY MUNICD; RUN;
PROC SORT DATA=ACHINFO; BY MUNICD; RUN;
```

Lastly, the payment file and the ACH file are merged on the common variable, MUNICD, to create the final interface file. A variable needed in the interface file, RECORD3, is the text string that involves the transaction code that determines the type of account that will receive the ACH payment. The old code used the static version of the transaction code in a macro variable named &MTRANSCD. The new code uses a variable from the input ACH file named TRANS_CODE that has been carefully formatted to be exactly 2 characters long. The old code is commented out and remains here for reference only. There is more to this DATA step than shown here; there are other text-string variables that are required in the output interface file.

```
DATA DETAIL3;                  /* MERGE PAYMENT FILE AND ACH TABLE BY MUNI CODE */
MERGE CHECKS ACHINFO;
BY MUNICD;
*RECORD3 = '6'||&MTRANSCD.||BANKNUM||ACCTNUM;      /* OLD CODE WITH STATIC VAR*/
 RECORD3 = '6'||TRANS_CODE||BANKNUM||ACCTNUM;      /* NEW CODE WITH DYNAMIC VAR*/
/*… (CREATE OTHER VARIABLES NEEDED FOR INTERFACE FILE )… */
FILE &MACHREC. LS=125;         /* OUTPUT INTERFACE FILE */
PUT
@ 3 RECORD3 $ 29.
/*… (OTHER VARIABLES NEEDED FOR INTERFACE FILE )… */
RUN;
```

The output interface file is written with a FILE statement with use of a Line-Size option to limit the number of characters per line; the interface file will eventually be uploaded to the z/OS environment. A PUT statement is used to tell SAS where to write the variables in the output interface file with their prescribed formats. Other variables are written to the interface file, but are not shown.

## CONCLUSIONS

Change is inevitable; when writing SAS programs, declare important static values as macro variables so they can be easily found by later programmers. Macro variables declared with a %LET statement will always result in a text string; numeric values will first be formatted as BEST12., then converted to text. Avoid or remove unexpected spaces in the macro variable by use of literals or use of the STRIP function upon invocation.

When making changes to any SAS program, be sure to keep the user in mind. Strive for minimal programmer intervention and minimal process change for the user; find a solution that works best for users of the input files and user of the SAS program. Allow the data to drive your output as much as possible, with minimum human intervention to decrease the risk of errors. Realize that anytime data is manually entered, there is risk of formatting change or missing data. Use SAS tools such as formatting with a PUT statement and assignment of a value if missing to correct these potential errors.

## REFERENCES


- SAS Institute Inc. 2011. *SAS® Certification Prep Guide: Advanced Programming for SAS®9, Third Edition*. 310, 318, 354. Cary, NC: SAS Institute Inc.
- Schacherer, Chris. 2011. "What's in a (FILE)NAME: The important role of a simple statement." *Proceedings of the MidWest SAS Users Group 2011 Conference*. Kansas City, KS. Available at http://www.mwsug.org/proceedings/2011/appdev/MWSUG-2011-AD12.pdf .
- Heaton, Ed. 2006. "So, Your Data are in Excel!" SAS Institute Inc. 2006. *Proceedings of the Thirty-first Annual SAS® Users Group International Conference*. Cary, NC: SAS Institute Inc. Available at http://www2.sas.com/proceedings/sugi31/020-31.pdf


## RECOMMENDED READING

- *The Little SAS® Book*
- *Base SAS® Procedures Guide*

## CONTACT INFORMATION

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


Misty Johnson
State of WI Department of Health Services
1 W Wilson St Rm 750
Madison, WI 53701
608-267-9561
misty.johnson@wi.gov