MATCH-MERGING: 20 Some Traps and How to Avoid Them

Malachy J. Foley

University of North Carolina at Chapel Hill, NC

ABSTRACT

Match-merging is a common form of combining files. Yet, it has its pitfalls. This tutorial examines more than 20 traps that can snarl even a seasoned programmer, and provides proven strategies for avoiding these traps.

INTRODUCTION

The little old match-merge is a complicated thing indeed. How complicated? Well in 1997, Foley (see REFERENCES) identified a dozen traps associated with match-merge. Amazingly, that was just the beginning.

This paper identifies and examines another fifteen ways match-merge can ambush a programmer. It summarizes all 28 pitfalls outlined in the two papers and explores defensive strategies to avoid all of them.

Among other strategies, a beta version of a SASÒ macro is presented to catch traps. This macro, when used with the SAS log, will detect or sidestep 88% of the traps. You can use this macro even if you are unfamiliar with macro code.

This article is for anyone who uses the SAS BASE product and match-merge. After studying it, you should know what most of the match-merge pitfalls are and how to defend yourself against them.

DECEIVING CODE COMPLEXITY

The first trap to explore is the complexity of the SAS code used for match-merging. It is deceivingly simple. For example, look at the code for the classic match-merge.

Exhibit 1: Classic 4-statement Match-Merge

```
DATA ONE TWO;
  MERGE ONE TWO;
  BY ID;
RUN;
```

This code could hardly be simpler or more compact. The pitfall is that the code's utter simplicity lulls the user into thinking nothing can go wrong. Consequently, the user drops his or her guard, and that is when things start to go wrong. These four seemingly innocent lines of code have burnt more than their share of SAS programmers, often badly.

28 TRAPS

So, what specifically can go wrong with a simple match-merge. The following table lists twenty-eight (28) ways a match-merge can go awry. SAS provides a note, warning, or error message for some of these situations. However, for most of the traps listed, no messages are produced. In these cases, it is totally up to the programmer to be aware of the traps and to catch them before they produce a faulty merge.

TABLE 1: List of Traps

By variable traps		
- Type mismatch	Pre	ERR
- Missing By statement	Pre	
- Justification mismatch	Pre	
- Ambiguous By variables	Pre	
- Length mismatch	Pre	
- Value instability	Pre	
- Manipulation History	Pre	
- BY var not in file	New	ERR
- Different case BY value	New	
- Missing By values	New	
Statement-related traps		
- Missing DATA statement	New	ERR
- Missing MERGE statement	New	ERR
- Missing RUN statement	New	
- empty DATA statement	New	
- empty MERGE statement	New	
- empty BY statement	New	
- MERGE state. with 1 DS	New	
- Incomplete BY statement	New	
Other traps		
- Testifying simple code	P/N	
- Automatic retain	P/N	
- Overlapping variables	P/N	INF
- Fractional BY values	Pre	
- Many-to-many/few merge	Pre	NTE
- Calculations in Merge	Pre	
- IF in Merge Data Step	New	
- WHERE on input vars	New	
- Input Files must exist	New	ERR
- Input files not sorted	P/N	ERR

In the previous SUGI paper by Foley, some of the traps listed in the above table were identified and discussed at length. These traps are tagged "Pre" (for Previous) in Table 1. Experienced SAS programmers actually fall into these traps.

This paper picks up where the other paper left off. It analyzes the remaining pitfalls in Table 1. These traps are labeled "New". This paper also provides an array of defensive strategies that programmers can apply to avoid most of the traps.

Situations that are detected by SAS are marked as NTE, ERR and INF which correspond to NOTE, ERROR and INFO messages. The INFO message is produced by SAS only if the "OPTIONS msglevel=i" statement is invoked.

BY VARIABLE TRAPS

The first part of Table 1 shows 10 traps associated with BY variables. A BY variable is one of the variables listed in a BY statement. The SUGI 22 paper gives a comprehensive treatment of most of these traps. The next two sections of this paper examine the two BY variable traps not discussed in the previous paper.

DIFFERENT CASE BY VARIABLE VALUES

It happens more often than you would like. You are trying to merge two files and the BY variables, which are character variables, should be all one case (upper or lower) and they are not. Usually one input file has all of its BY variable values (the value assigned to a BY variable) in upper-case letters and the other file has its BY variable values in lower-case letters.

The SAS match-merge is case sensitive. For instance, an upper-case letter "A" will not match with a lower-case "a". This is as it should be. A lower-case letter is not the same as an upper-case letter. Yet, often when you receive data from different sources, the alphabet is not keyed uniformly in upper or lower case.

Here is an example of this trap.

Exhibit 2: Ex. of BY Value Case Mismatch

```
----------------
DATA ONE TWO
  MERGE ONE TWO;
  BY ID;
RUN
----------------
```

FILE ONE

ID	NAME
A01	SUE
A02	TOM
A05	KAY
A10	JIM

FILE TWO

ID	AGE	SEX
a01	58	F
a02	20	M
a04	47	F
a10	11	M

FILE ONE TWO

ID	NAME	AGE	SEX
A01	SUE	.	
A02	TOM	.	
A05	KAY	.	
A10	JIM	.	
a01		58	F
a02		20	M
a04		47	F
a10		11	M

```
        NOTE: The data set ONE TWO has
              8 observations and 4 variables.
```

In this example, there are two input files called ONE and TWO. These files are combined using the classic 4-statement merge. Notice how none of the records matched in the output file, ONE TWO. The SAS log file shows no problems except for the NOTE (see above) which indicates that there are 8 observations in the output file.see DIf_case.* for information

- Maybe want to write sections on symptoms that you see and what they can mean, for example no match symptom can mean just /dif
- Is BOTH the best way to describe this union of files?
- OR you can get nothing see EX 2 in DIF_CASE??? -figure out what this means IF you have time later on.

One way to detect a case mismatch problem (as well as the justification mismatch described in the SUGI 22 paper) is to check the SAS log for the observation count. If the number of observations in the output file is equal to the sum of the observations in the two input files, then none of the records matched.

Another way to detect these mismatches is to add the following code after the merge DATA step. This code automatically checks the observation counts for two input files.

Exhibit 3: Check for justification/case mismatch.

```
DATA  NULL ;
  IF 0 THEN DO;
       SET ONE NOBS=NOBS1;
       SET TWO NOBS=NOBS2;
       SET ONE TWO NOBS=NOBS3;
       END;
  IF NOBS3=NOBS1+NOBS2 THEN DO;
    PUT "*** ERR: No records matched!";
    PUT "*** Check the justification/case";
    END;
  STOP;
RUN;
```

MISSING BY VALUES

One of the more subtle traps associated with BY variables is having an input file with missing values for the BY variables. The following exhibit illustrates this trap.

Exhibit 4:

```
DATA ALPH BET;
  MERGE ALPHA BETA;
  BY ID;
RUN;
```

ALPHA	BETA	ALPH BET

```
-------     --------     -----------
ID   V1     ID    V4     ID  V1   V4
      5           15          5   15
A35   3           22          5   22
                  A35   61    A35  3   61
```

The code in this case is the classic four-statement match-merge. The input files are normal in every aspect except that some of the values for the BY variable ID are missing.

The merge works properly and matches all the records according to the BY variable values. The trap lies in the fact that the BY variable values are faulty. It makes no sense to match records based on a nonexistent ID.

As such, the first record in the ALPHA file belongs to nothing and can not logically be matched to another record. Yet it is matched to two records in the BETA file.

The function of a BY variable in a match-merge is to IDENTIFY the record as belonging to someone or something. For instance, the ID could be the social security number of a person, an account number, or an inventory number. In each of these cases, the data in the record/observation is attached respectively to a specific person, account, or product.

When one of the record’s key variables is missing, it is impossible to ascertain to whom or what the data belongs. Therefore, the data in such a record is essentially useless. To keep a record with missing key variable values is misleading. It gives the impression that the information contained in the record belongs to some person or thing. Of course, there is no way of knowing to whom or what any of the information belongs.

When observations have missing values in their key fields, they probably should be deleted from the output file and flagged. At the very least, they should be flagged. One way of doing this is by adding an IF statement to the original four statements. For character ID’s, the code is:

EXHIBIT 5:

```
------------------
DATA ALPH BET;
  MERGE ALPHA BETA;
  BY ID;
  IF ID=" " THEN
    PUT "WRN - MISSING VALUE FOR A BY VARIABLE";
RUN;
------------------
```

For numerical ID's, the IF statement is:

EXHIBIT 6:

```
------------------
  IF ID le .Z THEN
    PUT "WRN - MISSING VALUE FOR A BY VARIABLE";
```

AUTOMATIC RETAIN

When there is a MERGE statement in a DATA step, all the input variables values are set to missing at the beginning of each BY group. These values are then retained until they are overwritten by new values which are inputted from one of the files. This retain function is implied by the MERGE statement and automatically implemented. For example, observe the following files.

EXHIBIT 7:

INPUT FILE ALPHA

ID	V1
A32	5
A35	3
A42	6

INPUT FILE BETA

ID	V4
A32	15
A32	22
A32	61
A42	47

OUTPUT FILE ALPH BET

ID	V1	V4
A32	5	15
A32	5	22
A32	5	61
A35	3	.
A42	6	47

In this example of a match-merge, note how the value 5 of V1 is retained for the entire A32 BY group. That is, see how the second and third records in the output file have V1=5, even though there is no corresponding record in the ALPHA file. Also note how V4=. for the A35 BY group, since V1 and V4 were set to missing at the beginning of the BY group. (BY groups are explained in section entitled MATCH-MERGE in the SUGI 22 paper.)

The automatic retain function causes much confusion about how the match-merge works. The confusion seems to come from two sources.

First, there is no such RETAIN in the "normal" DATA step that uses a SET statement. To retain a variable in a "normal" DATA step, you must explicitly use a RETAIN statement. So the first source of confusion is that one often forgets that the automatic retain actually exists when a MERGE statement is used.

Second, the automatic retain associated with the MERGE statement works differently from the explicit RETAIN statement that is used in a "normal" DATA step. One difference is that the explicit RETAIN statement initializes only the variables mentioned in the RETAIN statement. While the automatic retain statement initializes ALL of the input variables. Another difference is that the explicit RETAIN statement initializes its variables only once at the very beginning of the DATA Step. While the implied retain in a MERGE DATA step initializes its variables at the beginning of every BY group.

Consequently, with the RETAIN statement in the "normal" SET DATA step, values are retained for EVERY iteration of the DATA step. While in the MERGE DATA step, values are retained only during the BY group.

Since the implied retain is often forgotten and acts differently then the RETAIN statement, it causes even experienced programmers trouble, often heaps of trouble. In fact, from comments the author has received, the automatic retain is the single biggest cause of confusion regarding the match-merge. As such, it is included in the list of traps.

GETTING FANCY

Once you are familiar with the match-merge code and the corresponding DATA step algorithm, the temptation is to start adding code to the four-statement match-merge. For instance, adding a little calculation, or an IF statement to the DATA step loop seems harmless enough. But, as the previous paper demonstrates, adding a single calculation (assignment statement) to the basic match-merge can cause unexpected results.

The next exhibit shows how adding just one IF statement to the merge DATA step can cause unexpected results.

EXHIBIT 8:

```
---------------------------
DATA ONE TWO;
  MERGE ONE TWO;
  BY ACCT;
  IF REASON=400 THEN REG=99;
RUN ;
---------------------------
```

SLIDE C too
ACCT--> ID

FILE ONE

ACCT	REASON	VAL
1	100	100
1	200	500
1	400	670
1	600	150
1	700	900
2	150	325

FILE TWO

ACCT	REG
1	50
2	57

FILE ONE TWO

ACCT	REASON	VAL	REG
1	100	100	50
1	200	500	50
1	400	670	99
1	600	150	99
1	700	900	99
2	150	325	57

Here, the intent of the IF statement is to set the region (REG) equal to 99 when the reason variable is equal to 400. Yet look at what happens when two files are merged using the code.

Because SAS automatically retains the values of all the input variables during a BY group, not only is REG set to 99 when REASON=400, but it is also set to 99 when REASON equal 600 and 700 (the rest of the BY group).

In this case, no SAS messages existed in the log to indicate that anything is amiss. What is worse, is that the code was used to suppress billings for reason codes 400. If the bills had been mailed, there would have been serious consequences.

One defensive strategy to avoid this trap is to never change any of the input variables, but rather to create new variables to receive the outcome of any manipulation of the input data.

The following code and output file demonstrate this strategy. It uses the same input files given in the previous exhibit. Again, the intent of the IF statement is to set the region equal to 99 when the reason variable is equal to 400. The difference between the next example and the previous example is that the input variable REG is not changed, but rather a new variable REG2 is created to receive the new/changed region information. Of course, the IF statement changes to reflect the new variable. In this example, the intended results are obtained.

EXHIBIT 9:

```
DATA ONE TWO;
  MERGE ONE TWO;
  BY ACCT;
  IF REASON=400 THEN REG2=99;
    ELSE REG2=REG;
run ;
```

FILE ONE TWO

ACCT	REASON	VAL	REG	REG2
1	100	100	50	50
1	200	500	50	50
1	400	670	50	99
1	600	150	50	50
1	700	900	50	50
2	150	325	57	57

Another defensive strategy that many SAS programmers actually employ and recommend is: to use only the basic 4-statement match-merge when merging files and to do all other processing in a separate DATA step. This is their policy. It may seem a bit extreme, but it is definitely the safest way to merge.

THE CASE OF WHERE=

This is a strange case. It is mentioned because it actually causes users problems and questions. It consists of using the WHERE= data set option with a match-merge. Specifically, a user has the following two input data sets:

EXHIBIT 10:

ALPHA

ID	V1
A20	5
A21	7
A24	4
A35	3

BETA

ID	V4
A20	15
A20	12
A24	92
A24	38
A35	61

The goal is to output only those records where V1=5. The program that follows was proposed to meet this goal.

```
EXHIBIT 11:
---------------------------
 DATA ALPH BET;
   MERGE ALPHA(WHERE=(V1=5))
         BETA;
   BY ID ;
 RUN ;
---------------------------
```

```
 ALPH BET
------------
ID   V1   V4
------------
A20   5   15
A20   5   12
A24   .   92
A24   .   38
A35   .   61
```

However, the above output was not exactly what the user was expecting from the program. He was surprised there were output records for ID's A24 and A35, since V1 was not 15 for these ID's.

One programmer admitted that a situation, like this one, took him days to figure out and correct.

Of course if the automatic retain that accompanies the MERGE statement is remembered and fully understood, it is obvious how to get the desired results. What happened in this case is that the automatic retain was not remembered.

```
---------------------------
DATA ALPH_BET(WHERE=(V1=5));
  MERGE ALPHA BETA ;
  BY ID ;
RUN ;
---------------------------
```

```
 ALPH_BET
------------
ID   V1   V4
------------
A20   5   15
A20   5   12
```

The way to sidestep this kind of trap, once again, is to keep the match-merge as simple as possible. Stick to the basic 4-statement match-merge. The WHERE can safely be used in a subsequent DATA Step.

BETTER SAFE THAN SORRY

The last two cases showed how the automatic retain can occasionally trip up even the most experienced programmer. They showed how the trip ups were caused by adding just a little bit of extra code to the basic four-statement match-merge. These kinds of trip-ups can have serious consequences, like the under billing of clients.

A good defensive strategy at any time, in any profession, to avoid problems is the KISS principle. KISS is an acronym for "Keep It Short and Simple". It is a well-known fact in reliability theory that the fewer components a system has, the less likely it is to fail. KISS is just a another way of stating this fact.

When applied to match-merging the principle can be stated as follows: "Use only the basic 4-statement match-merge and to do all other processing in a separate DATA step". In other words, whenever possible, NEVER use a WHERE, an IF statement, an assignment statement, or anything else in the match-merge code.

If manipulation is required, if at all possible, do it in a separate DATA step after the merge DATA step. This is extreme, but it works.

When programming, there certainly is a temptation to get fancy, and add a little code. Nevertheless "Keeping It Short and Simple" is frequently a good, easy, safe trick, which avoids untold amounts of anguish and work.

STATEMENT-RELATED TRAPS

Most programmers are interrupted frequently during the day by phone calls, office visits, and the like. After each interruption, they must recapture their train of thought and continue with their programming where they left off. Sometimes after an interruption, a tiny part of the programming that was in process is skipped.

Take the four-statement match-merge, for instance. One can unintentionally leave out whole statements or part of the statements. Often the missing part of a match-merge goes undetected and causes errors in the program’s output. The next sections of this paper explore all possibilities of missing pieces of code. These sections also look at the consequences of the missing code, and strategies to avoid having missing code in your programs.

MISSING STATEMENTS

What happens when one of the four statements in the classic merge is missing? There are four possibilities.

SAS puts an error message in the log file when there is either a missing DATA statement or a missing MERGE statement. For a missing DATA statement, the message is:

```
ERROR 180-322: Statement is not valid or it is
                            used out of proper order.
```

For a missing MERGE statement, the message is:

```
ERROR: A SET, MERGE, or UPDATE statement not present.
```

SAS gives no messages for a missing BY statement. Rather it performs a one-to-one merge, instead of a match-merge. Forgetting a BY statement is a common and potentially catastrophic event.

When a RUN statement is missing, the merge will usually execute properly. However, when the merge DATA step is the last step in the program and RUN is missing, the merge will not execute and SAS issues no message.("step boundaries") This is one reason why it is always good programming practice to include the RUN statement at the end of each DATA step.

The consequences of omitting one of the four classic match-merge statements can be as simple as losing the time it takes to run the job, review the log and correct the code. Or the consequences can be as serious as getting a wrong output without even knowing it.

To sidestep all of these consequences, you can look at the log file for all the messages regarding the DATA step and check that the output file has the expected number of observations.

However, another procedure to sidestep missing statements is to use a macro which already has all the statements in it. This procedure is easier and more reliable than checking the log. Such a macro is described below.

EMPTY STATEMENTS

The previous section reviewed what happens if one of the four statements of the classic match-merge is missing. But what happens when the statement is empty.

A statement is empty when it has nothing placed between the statement keyword and the semicolon. For example, a DATA statement is empty when there is no file name specified between DATA and the semicolon.

When you have an empty DATA statement (DATA ;), SAS puts no message in the log file. Rather SAS performs the match-merge and automatically assigns an output data set name in the WORK library.(see One_DS.*)

When you have an empty MERGE statement (MERGE;), SAS gives no message. It simply outputs the most recently created data set! In the unlikely event that such a data set does not exist, SAS will issue the following message:(see One_DS1.*)

```
ERROR: There is not a default input data set
                                ( LAST  is  NULL ).
```

When the BY statement is empty (BY ;), SAS provides no message and performs a one-to-one merge. In this case, SAS acts just like it does when there is a missing BY statement. (see no_byvar.*) (gives no error, warning or note!)

Of course, the RUN statement is usually empty, and as such, is always fine.(can have a CANCEL argument) (move this above?)

PARTIAL STATEMENTS

Finally, in looking at what can go wrong with the SAS statements themselves, partial statements are examined. In this paper, a partial statement is defined as a statement that has some, but not all, of its parameters specified.

The RUN statement is always "complete" since it requires no parameters.

The DATA statement usually has one file specified as a parameter. Indeed, this paper suggests that it have only one file specified to keep the merge code simple. Of course, the DATA statement does not require a data set to be specified, and this scenario is covered in the section on EMPTY STATEMENTS.

If a MERGE statement has only one input file specified, SAS will output that file as is, without giving any messages. One way to avoid this situation is to use macro code which checks that the MERGE statement has at least two input files. Such code is found in the section entitled MACRO later in this paper.

A BY statement is incomplete when there are fewer BY variables specified than are required to completely identify the observations for proper matching. An incomplete BY statement is very hard to spot. Two ways to detect this situation are: (1) to compare the actual number of observations in the output file against the number of expected observations; and (2) to examine a print out of the observations in the output file. Both these ways are cumbersome and time-consuming. So a better defense is to be very careful in choosing and specifying the variables in a BY statement. The section entitled BY VARIABLE CHECK LIST has more information on this topic.

OVERLAPPING VARIABLES REVISITED

The SUGI 22 paper examined overlapping variables, explained why they were undesirable, and offered some defensive strategies to evade them. Here the overlapping variable is revisited and more defensive strategies are offered.

When two input files have some non-BY variables in common, those variables are called overlapping variables.

The following is an example of a match-merge with one overlapping variable, V4.

EXHIBIT 12:

```
------------------
DATA ALPH BET;
  MERGE ALPHA BETA;
  BY ID;
RUN;
------------------
```

FILE ALPHA

ID	V1	V4
A32	5	10
A35	3	6

FILE BETA

ID	V4
A32	15
A32	22

FILE ALPH BET

ID	V1	V4
A32	5	15
A32	5	22
A35	3	6

Note how, in the first record, the value of V4 in the second merge file overwrites the value of V4 in the first merge file. Even a missing value can overwrite a valid value.

Since overlapping variables cause SAS to overwrite values, the merit of the overlapping variable is often dubious. When a programmer is unaware of overlapping variables or forgets about them you will hear the question "why are the values of my variables changing".

One way to check for overlapping variables is to examine a PROC CONTENTS of all the input files and see if any overlapping variables exist. This method is cumbersome, time-consuming, and error-prone.

Another method is to compare the number of input variables with the number of output variables. If there are two files and no overlapping variables, the number of output variables (O) should be the sum of number of input variables (I1+I2) less the number of BY variables (B). In formula form, O=(I1+I2)-B. If

there are n files and no overlapping variables, then O=(I1+...In)-B*(n-1). This method is less error-prone and cumbersome than the first, but still a bit difficult to employ.

Another method to detect overlapping variables is to add a system option before your merge DATA step.

EXHIBIT 13:

```
OPTIONS MSGLEVEL=i;
DATA ONE TWO;
   MERGE ONE TWO;
   BY ID;
RUN;
```

This option will make SAS print messages to the log which identify each overlapping variable, including the BY variable which legitimately need to overlap. The message is of the form:

```
INFO: The variable ID on data set WORK.ONE
      will be overwritten by data set WORK.TWO.

INFO: The variable V4 on data set WORK.ONE
      will be overwritten by data set WORK.TWO.
```

This method of discovering overlapping variable is better than the previous two. Yet, you must remember to add the OPTIONS statement and to distinguish between the BY variables and the non-BY variables in the INFO statements.

Another technique for catching overlapping variables is to use macro code to implement the variable-counting method. The section entitled MACRO has such code.

DEFENSIVE STRATEGIES

Many things can go wrong when you attempt to do a match-merge. Usually it is a case of forgetting a statement or part of a statement, or forgetting one of the many nuances associated with the match-merge. Various ways to detect or prevent a trap from leering its nasty head have been suggested here and in the previous paper. However, there are so many suggestions, it is hard to know how to proceed to actually sidestep the many traps. The next sections of this paper attempt to put it all together and come up with a comprehensive strategy to avoid as many traps as possible.

MACRO

Here is the first strategy. It is a beta version of a "do-it-all" macro which avoids or detects 17 traps.

For example, earlier in this paper eight statement-related traps were discussed. These traps occur where a programmer forgets or mis-keys a statement or part of a statement, obtaining unexpected, and often undetected, results. One way to sidestep all these statement-related difficulties, is to use a macro with the classic match-merge code already written into it. Such a macro is proposed below. Copying this macro (called MATCH) into your program and

invoking it makes it nearly impossible to miss a statement or the parameters necessary for a basic match-merge.

Also described above were three cases where adding extra code to the basic match-merge caused unexpected results. Using the MATCH macro guarantees that no extra code is accidentally added to the basic merge.

The MATCH macro includes the code discussed in the section entitled DIFFERENT CASE BY VARIABLE VALUES(this is the title as of 1/8). Thus, MATCH catches the two traps of different cases and different justification in the BY variables values.

Furthermore, MATCH alerts the programmer to BY variables LENGTH mismatch and fractional BY variables. Both of these pitfalls are discussed in the prior paper. Either can indicate a manipulation history mismatch. And both can cause incorrect merges.

Finally, MATCH detects overlapping variables.

For your reference, here is a listing of the MATCH macro. You can use this macro even if you are unfamiliar with macro code. The next section of this paper describes how to use the macro..

```
%MACRO MATCH(IN_FILES=, OUT_FILE=, BY_VARS=);
   * Copyright (c) 1998 by Malachy J. Foley;
   * All rights reserved except as noted;
   %LET FLAG=0; %* THIS IS AN ERROR FLAG;

   *--------------------------------------;
   * CHECKING THE STATEMENT PARAMETERS;
   %LET NIN=0;
   %DO %WHILE(%SCAN(&IN_FILES,&NIN+1) NE );
       %LET NIN=%EVAL(&NIN+1);
       %END;
   %IF &NIN NE 2 %THEN %DO;
        %LET FLAG=1;
        %IF &NIN<2 %THEN
          %PUT *** ERR: TOO FEW INPUT FILES SPECIFIED;
        %IF &NIN>2 %THEN
          %PUT *** MATCH REQUIRES EXACTLY 2 INPUT FILES;
        %END;
     %ELSE %DO;
        %LET FILE1=%SCAN(&IN_FILES,1);
        %LET FILE2=%SCAN(&IN_FILES,2);
        %END;

   %LET NOUT=0;
   %DO %WHILE(%SCAN(&OUT_FILE,&NOUT+1) NE );
        %LET NOUT=%EVAL(&NOUT+1);
        %END;
   %IF &NOUT NE 1 %THEN %DO;
        %LET FLAG=1;
        %PUT *** MATCH REQUIRES EXACTLY 1 OUTPUT FILE;
        %END;

   %LET NBY=0;
   %DO %WHILE(%SCAN(&BY_VARS,&NBY+1) NE );
```

```
    %LET NBY=%EVAL(&NBY+1);
    %END;
%IF &BY_VARS=  %THEN %DO;
     %LET FLAG=1;
     %PUT *** ERR: BY VARIABLES MUST BE SPECIFIED;
     %END;

%*------------------------------------------------;
%IF &FLAG=0  %THEN %DO;
      * CHECK FOR OVERLAPPING VARS;

      * THE TEMPORARY FILE;
      DATA KILLZYXW;
        MERGE &IN_FILES;
        BY &BY_VARS;
        STOP;
      RUN;

      PROC CONTENTS DATA=&FILE1
                  OUT=CONTNTS1(KEEP=VARNUM) NOPRINT;
      PROC CONTENTS DATA=&FILE2
                  OUT=CONTNTS2(KEEP=VARNUM) NOPRINT;
      PROC CONTENTS DATA=KILLZYXW
                  OUT=CONTNTS3(KEEP=VARNUM) NOPRINT;
      DATA _NULL_;
        IF 0 THEN DO;
             SET CONTNTS1 NOBS=NVAR1;
             SET CONTNTS2 NOBS=NVAR2;
             SET CONTNTS3 NOBS=NVAR3;
             END;
        IF NVAR3 NE (NVAR1+NVAR2-&NBY) THEN DO;
             PUT "*** ERR: OVERLAPPING VARIABLES.";
             CALL SYMPUT('FLAG','1');
             END;
        STOP;
      RUN;
%END;
%*------------------------------------------------;
%IF &FLAG=0  %THEN %DO;
      * CHECK FOR LENGTH MISMATCH;

      PROC CONTENTS DATA=&FILE1(KEEP=&BY_VARS)
                  OUT=CONTNTS1(KEEP=NAME LENGTH TYPE
                               RENAME=(LENGTH=LEN1)
                              ) NOPRINT;RUN;
      PROC SORT; BY NAME;RUN;
      PROC CONTENTS DATA=&FILE2(KEEP=&BY_VARS)
                    OUT=CONTNTS2(KEEP=NAME LENGTH
                                 RENAME=(LENGTH=LEN2)
                                 ) NOPRINT;RUN;
      PROC SORT; BY NAME;RUN;

      DATA _NULL_;
        MERGE CONTNTS1 CONTNTS2;
        IF LEN1 NE LEN2 THEN
          PUT "*** WRN: BY VAR LENGTH MISMATCH " NAME;
      RUN;
```

```
      %END;
%*-------------------------------------------------;
%IF &FLAG=0  %THEN %DO;
      * SET UP FOR CHECKING MISSING BY VALUES;
      *  (NOTE 200 CHARACTER LIMIT);
      DATA _NULL_;
        SET CONTNTS1 END=LAST;
        LENGTH LIST_N LIST_C $200;
        RETAIN LIST_N LIST_C;
        IF TYPE=1 THEN DO;
           L=LENGTH(LIST_N);
           LIST_N=SUBSTR(LIST_N,1,L)||" "||NAME;
           END;
        ELSE DO;
           L=LENGTH(LIST_C);
           LIST_C=SUBSTR(LIST_C,1,L)||" "||NAME;
           END;
        IF LAST THEN DO;
           CALL SYMPUT("BYVARSN",LIST_N);
           CALL SYMPUT("BYVARSC",LIST_C);
           END;
      RUN;
      %LET NBYN=0;
      %DO %WHILE(%SCAN(&BYVARSN,&NBYN+1) NE );
          %LET NBYN=%EVAL(&NBYN+1); %END;
      %LET NBYC=0;
      %DO %WHILE(%SCAN(&BYVARSC,&NBYC+1) NE );
         %LET NBYC=%EVAL(&NBYC+1); %END;
      %END;
%*-------------------------------------------------;
%IF &FLAG=0  %THEN %DO;
      * THE CLASSIC 4 STATEMENT MERGE;
      DATA &OUT_FILE;
        MERGE &IN_FILES;
        BY &BY_VARS;

        * CHECK FOR MISSING BY VALUES;
        %IF &NBYN NE 0 %THEN %DO;
                 ARRAY NNNNN(&NBYN) &BYVARSN; %END;
        %IF &NBYC NE 0 %THEN %DO;
                 ARRAY CCCCC(&NBYC) &BYVARSC; %END;
        %IF &NBYN NE 0 %THEN %DO;
           DO IWXYZYX=1 TO &NBYN;
             IF NNNNN(IWXYZYX) LE .Z THEN
              PUT "*** WRN: MISSING BY VALUE DETECTED"
                     NNNNN(IWXYZYX)=;
             * LITTLE CHECK FOR FRACTIONAL BY VALUES;
             IF NNNNN(IWXYZYX) NE
                         (INT(NNNNN(IWXYZYX))) THEN
               PUT "*** WRN: FRACTIONAL BY VALUE "
                     NNNNN(IWXYZYX)=;
           END;
           %END;
        %IF &NBYC NE 0 %THEN %DO;
           DO IWXYZYX=1 TO &NBYC;
             IF CCCCC(IWXYZYX)=" " THEN
               PUT "*** WRN: MISSING BY VALUE "
```

```
                    CCCCC(IWXYZYX)=;
            END;
            %END;
        RUN;
        *-----------------------------------------;
        * CHECK FOR JUSTIFICATION/CASE PROBLEMS;
        DATA _NULL_;
          IF 0 THEN DO;
               SET &FILE1 NOBS=NOBS1;
               SET &FILE2 NOBS=NOBS2;
               SET &OUT_FILE NOBS=NOBS3;
               END;
          IF NOBS3=NOBS1+NOBS2 THEN DO;
            PUT "*** ERR: NO RECORDS MATCHED! (CHECK";
            PUT "*** THE JUST/CASE OF CHAR BY VALUES)";
            END;
          STOP;
        RUN;
    %END;
   %ELSE %PUT *** WRN: DUE TO ERRS, MERGE IS CANCELLED;
%MEND MATCH;
```

This macro is a beta version. So it may have some bugs, and you should use it at your own risk. Also, it does not work in all cases. For example, it is definitely not designed to handle more than two input files at a time. However, it could be generalized to handle more than two input files. Also, it can be refined. For instance, it would be nice to have it write the "Fractional BY value" warning only once. Despite these limitations, it is useful and it provides an idea of the kind of code that can be used to avoid many of the traps associated with the match-merge. You can use the macro as is or you can modify it. It may be used freely for in-house programming which is not sold. It is not intended to be sold by itself or as part of a larger program without the author’s permission.

HOW TO USE THE MATCH MACRO

The simplest way to use the MATCH macro is to copy it into your program as is. Then, invoke it whenever you are going to do a match-merge later in the program. Here is an example of how to invoke it.

```
  %MATCH(in_files=ONE TWO, out_file=ONE_TWO, by_vars=ID);
```

Notice that MATCH, among many other things, checks that:

- there are exactly 2 input data sets in the MERGE statement;
- there is exactly 1 output file in the DATA statement; and,
- there is at least one BY variable specified in the BY statement.

MATCH assumes that the two input files are sorted in ascending order of the BY variables.

MATCH is particularly suited for merging two relatively small files. If you have several small files to merge, consider merging them two at a time with MATCH.

If you have large files where the computing overhead involved in running MATCH would be prohibited, consider running some sample subset of your files

through MATCH. Or you can simplify the second to last section of MATCH to include only the classic 4-statement merge. This would eliminate all the overhead that would accompany two large files.
Or test the large files once and after you are satisfied they are working, simplify the second to last section.

It is recommended that you use MATCH as often as possible and that you do all your data manipulation and subsetting in a later DATA step

One thing you can not do in a subsequent DATA step is subsetting based on the IN= data set option. In this case, it recommended that you carefully add the IN= subsetting to the second to last part of the MATCH macro where the 4-statement merge is located. Be sure to set IN= to a variable name that is already used in any of the input files.

This macro, in a single ploy, avoids 17 traps. It is not the only trick you can use to defend yourself against faulty merges, but it certainly is the easiest.

LOOK AT THE LOG

It is always a great strategy to look at the SAS log. SAS directly detects seven match-merge traps. The MATCH macro was designed to be used in conjunction with the SAS log. Specifically, SAS detects five traps, not detected by MATCH, and writes a NOTE or ERROR message as follows:

- Many-to-many/few merge NOTE (a)
- Type mismatch (char/num) ERROR (b)
- BY variable not in file ERROR (c)
- Input Files do not exist ERROR (d)
- Input files not sorted ERROR (e)

SAS also detects the following two traps detected by MATCH

- Missing DATA statement ERROR (f)
- Missing MERGE statement ERROR (g)

Finally, if the "OPTIONS msglevel=i;" statement is used, SAS catches

- Overlapping variables INFO (h)

The corresponding messages that SAS writes for all of the above traps are:

(a) NOTE: MERGE statement has more than one data set
with repeats of BY values.
(b) ERROR: Variable ID has been defined as both character
and numeric.
(c) ERROR: BY variable xxx is not on input data set yyy.
(d) ERROR: File zzz does not exist.
(e) ERROR: BY variables are not properly sorted on data
set yyy.

(f) ERROR 180-322: Statement is not valid or it is used
out of proper order.
(g) ERROR: A SET, MERGE, or UPDATE statement not present.

```
(h) INFO:  The variable xxx on data set yyy will be
           overwritten by data set zzz.
```

Pay close attention to NOTE (a). It is only a note, nonetheless it points to a real pitfall. It indicates that records within a BY group are being match willy-nilly. And this pitfall is not checked for by MATCH. See the SUGI 22 paper for more information on this topic.

BY VARIABLE CHECK LIST

The MATCH macro and the SAS log together directly detect or avoid 22 traps. Yet these two ploys are no substitute for good input data.

Particular attention must be given to the BY variables. Obviously, if your BY variables are faulty in any way, the records from the input files will never match correctly (Garbage in, Garbage out). Said another way, no matter how good your programming skills are, or how good your defensive strategies are, your BY variables have to be in tip-top condition. Enough can not be said about the importance of good BY variables.

In an ideal world, there would always be, one and only one, set of BY variables to uniquely identify each data item. In the real world, BY variables are slippery critters. In one file they are justified one way, in another file, the other way. They are mis-keyed. They are not keyed, leading to missing values. And in general, they are mis-treated in many ways.

The twin ploy of MATCH macro and SAS log catches most traps. The only traps that sneak past these two defensive strategies are BY variable related. In fact, from Table 1, the only three (3) traps of substance that are not completely detected by the two ploys are:

- Ambiguous By variables
- By Value instability
- BY variable Manipulation History

The previous paper discusses each of these traps.

There are a couple of pitfalls that are not even mentioned in Table 1. They are subtle (non-consistent) case mismatches and subtle (non-consistent) justification mismatches. When there is only an occasional mismatch in the case or justification of a character BY variable value, MATCH will not detect it. In fact, it is very hard to detect. One way to sidestep these two traps is to guarantee that BY variables match by applying the following code to them in every input file before the files are merged.

```
ID= LEFT(UPCASE(ID));
```

A pre-condition to a good merge is to have your BY variables in tip-top shape. Consider the following check list of conditions.

Do all BY variables...

- Use check digits to avoid mis-keying?
- Have no unnecessary duplicate BY values?
- Uniquely and unambiguously identify the data item?
- Have no missing BY values?
- constitute a legitimate key variable

- Have a known manipulation history?
- As a group, complete identify the data item?

Do the character BY variables...
- Have uniform justification?
- Have uniform case usage (upper/lower)?

Do the numeric BY variables...
- Use only integer numeric BY values?

COMPREHENSIVE STRATEGY

One of the goals of this paper was to develop a comprehensive strategy to avoid as many match-merge traps as possible. The strategy proposed here for a simple match-merge is to use a three-pronged attack: the MATCH macro, the SAS log, and the BY variable check list.

If you want or need to go beyond the basic four-statement match- merge, it is recommended that you: (1) use a modified version of the three-pronged attack; (2) keep the merge as simple as possible; (3) fully understand the automatic retain; (4) understand the match-merge DATA step algorithm given on page 151 of SAS Language: Reference (Version 6, First Edition), under the title "DATA Step Processing during Match-merging"; and (5) remember that the match-merge is full of traps.

CONCLUSION

While the SAS code to perform a file merge is often simple, the merge process itself can be difficult. In an ideal world, the BY variables are carefully designed and chosen well before any data is collected. They truly and unambiguously identify, label, and distinguish each record in a data set. In the real world, BY variables are frequently jerry-rigged or a fuzzy merge is required. Even when good BY variables are available, the BY value and the merge process need to be carefully checked.

There are at least 28 traps associated with match-merge. These traps can go undetected and cause unexpected results. Sometimes these results are aggravating or costly to detect and correct. Other times, undesirable results go undetected and can cause real problems in billings, inventories, etc.

The best defense is an offense. This paper provides a comprehensive proactive strategy to avoid or detect all of the traps. The strategy includes a beta version of a macro which, in conjunction with the SAS log, catches or dodges most of the traps.

REFERENCES

Foley, Malachy J. (1997) "Advanced Match-Merging: Techniques, Tricks and Traps" Proceedings of the Twenty-Second Annual SAS Users Group International Conference, 199-206.

SAS Institute, Inc. (1990), SAS Language: Reference, Version 6, First Edition, Carry, NC: SAS Institute Inc.

AUTHOR CONTACT

The author welcomes comments, questions, corrections and suggestions.

Malachy J. Foley
2502 Foxwood Dr.
Chapel Hill, NC 27514

Email: FOLEY@unc.edu