Paper 206

From MACRO to SAS/AF® Software: A Case Study of the Evolution of an Application

David M. Gardner, University of North Carolina at Chapel Hill

ABSTRACT

SAS® software applications developers often face a choice when beginning a new project: Use the MACRO facility or use SAS/AF? This paper traces the evolution of one such application from its beginning as a single-purpose macro, its subsequent adoption and enhancement for another, more-demanding job, and its eventual obsolescence and re-emergence in SAS/AF FRAME. The story illustrates the traditional tradeoffs between MACRO and SAS/AF. Were it not for the relative ease of programming in MACRO code, the application may never have been created . However, once created, it proved to be resistant to enhancement, and needed to be re-written in SAS/AF with expanded capability.

INTRODUCTION

In 1995 the Design and Statistics Unit of Frank Porter Graham Child Development Center at The University of North Carolina changed the way it entered research data into SAS data sets. Switching from traditional procedures reminiscent of the 80-column punched card era, the unit began to enter data through the SAS/FSP® software's FSEDIT procedure, which provided a more attractive graphical interface. Consistent with good data-management practices, the unit continued the double-keying of data, but now the process would be brought in house. Double-keying is generally a two phase process, requiring independent data entry by two keyers, followed by a "reconciliation" phase, during which the two data sets are compared and mistyped values corrected. The end result is a single data set representing the new data.

The first attempts at reconciliation used the COMPARE procedure to find the discrepancies in the two keyers' data sets. These discrepancies would be resolved by reviewing each data set in turn through FSEDIT screens, locating discrepant fields, and making the corrections. Typically, not all discrepancies could be corrected on this first pass. PROC COMPARE would have to be re-run, and additional problems identified and corrected.

The inefficiency of this iterative process spurred efforts to develop alternatives that would streamline the process.

THE EMERGENCE OF THE ORIGINAL RECONCILE MACRO

Relief would come soon. The unit had already developed a one-thousand-line SAS macro that compared two datasets and reported the discrepancies observation by observation. This information was used by non-programmers outside the unit to verify previously requested changes to the database. This "Two-way Compare Macro" was soon put to use in generating the initial list of differences between the keyers' data, a task previously done by PROC COMPARE.

The design of the Two-Way Compare Macro influenced the direction of its evolution. At its heart was a DATA _NULL_ step that merged two datasets. It was an obvious leap to incorporate a third data set—the one containing the reconciled data. This produced a complete record of all decisions made during reconciliation. Once this new "Three-way Compare Macro" was up and running, a further goal seemed within grasp: bringing the user in on the process. A DAT-step WINDOW statement was wedged into the code, and voila!, a batch-mode report generator stirred to life and began to converse with users. A lowly DATA _NULL_ step had just metamorphosed into a useful application program that would be midwife to many hundreds of newborn data sets in the following two years. Such were the origins of the "Reconcile Macro."

The reader may be wondering about the programming effort involved in the Reconcile Macro. The original Two-way Compare Macro ran to about 1,000 lines of code; the ultimate Reconcile Macro would weigh in at 2,600 lines of code. The entire development process was incremental and cumulative. Approximately 75 hours of programming went into the Two-Way Compare Macro; another 75 hours produced the Reconcile Macro. This initial investment would be more than matched by the effort required to maintain and enhance the application, which would consume about 150 hours over the next two years.

The Reconcile Macro served its purposes well in the two years following its development. Enhancements were accomplished by tacking on code before the main DATA step, or after it. Only under dire need was the main DATA step itself to put under a text editor's cursor. For at the heart of the application was a Rubik's Cube of a DATA step that frequently defied change. To the programmer attempting to modify it, it behaved like three distinct—though inextricably interlocked—layers. The outer layer was macro code, exquisitely sensitive to misplaced asterisks and unmatched quotes. SAS DATA step statements represented two layers: one type of statement whispered particulars to the DATA step compiler regarding the identity and order of variables to expect from the input datasets and create on the output dataset; the other type took control during execution and processed the data a piece at a time. The Reconcile macro was an intimidating creature that did not take easily to changes.

To better appreciate the challenges posed by such an accumulation of Macro code, consider the increasingly problematic nature of punctuation in SAS syntax. The following DATA step runs fine in open code:

```
data one;
     stop;
   * It’s okay to stop now. ;
run ;
```

But when wrapped in a MACRO definition, an error is provoked:

```
%MACRO MACROONE;
data one;
     stop;
   * It’s okay to stop now. ;
run ;
%MEND ;
%MACROONE ;
```

Can the reader spot the problem? For the solution, see Note 1.

As amusing as this brain-teaser is, it becomes somewhat less fun when surrounded by 5,000 lines of code. As the length of the macro increases, so does the length of time needed to find each instance of bad syntax. Such needle-in-a-haystack syntax problems became a major time waster in the maintenance of the Reconcile Macro.

NEW BEGINNINGS: THE SAS/AF VERSION

By Spring of 1998 several considerations arose that argued for a completely new re-write of the application. One was the difficulty in making enhancements. For example, the ability to revisit a previous value would require adding POINT options to SET statements. It could be done, but testing it would require building a very large number of test cases to ensure that the code handled all permutations of input data sets? With SAS Release 7 on the horizon with its anticipated extra-long variable names, requirements for significant changes loomed ever closer. To modify and test the macro to accommodate 32-letter variable names would have required about 60 to 80 hours of programming and testing time—a significant price to pay to make a minor change in a program that required only 300 hours to develop from scratch.

The Reconcile Macro had proven itself to be not only feasible to program, but indispensable to the operations of the Unit. Its main drawback[2] was the lack of maintainability due to the inherent nature of macro code. The solution was to re-write it in SAS/AF. The decision was informed by a desire to remain within the SAS environment (which ruled out applications built on other programming languages that could not access SAS data sets).

The new SAS/AF version of the application required about 300 hours to write, document, debug, and test. It produced about 5,000 lines of Screen Control Language and SAS language. Fortunately many parts of the original program—not the main DATA STEP, but some routines to generate reports—were easy to incorporate in the new application in SUBMIT—CONTINUE blocks.

The original macro invocation interface was preserved so users would not need to rewrite working programs. The main goal of simplifying the Macro was achieved by substituting SCL for DATA step code in the main loop. SCL provides a set of alternative techniques for operating on data sets. The advantage is that the programmer has greater control when dealing with data sets that have unknown structures (variable names and number and order of observations). The disadvantage is that the process of opening an input dataset, reading an observation’s worth of values, writing new values to an output dataset, and finally closing both datasets requires statements explicitly performing each operation. So, a given task took much longer to implement in SCL than as DATA step wrapped in MACRO language—maybe three times as long—but was much easier to debug and maintain.

SCL also allows the manipulation of data sets without intermingling variables from the target data sets with variables from the program itself. This was done by avoiding the use of the SCL SET statement, which links together similarly named variables from the SCL program and the specified dataset. Instead, the GETVARN function was used to read numeric values from data sets, and the PUTVARN function was used to write numeric values out. (The GETVARC and PUTVARC functions handle character values.) In the original macro, this problem was solved by a naming convention under which all variables unique to the DATA step began with the prefix "_WI_". This left only four letters with which to create meaningful names for more than 100 internal variables. The process of manipulating datasets with the SCL functions, which are now available in the 6.12 DATA step, proved to require a lot of initial programming time but minimal maintenance effort once programmed.

ONE APPLICATION: TWO IMPLEMENTATIONS

MACRO language and SAS/AF resulted in strikingly different implementations of nearly identical systems.

Better Integration with Other SAS Procedures: Reconciliation directly from an FSEDIT screen. (Figure 1) The integration of the FSEDIT screens into the reconciliation application allows some nice features. A user may click on the "FS EDIT IT" icon on the main reconciliation FSEDIT screen (Figure 2, bottom) to be taken directly to the observation and variable being displayed.

Improving the look and feel: The use of widgets in the SAS/AF implementation enlivened the main reconciliation screen (Figure 2) and allowed for the display of more information. Adding additional widgets to implement additional user options is rather straight-forward. Of particular use in this regard is the Tab Layout object, which allows multiple "tabs" that function somewhat like multiple FRAME entries based on the same SCL entry. SAS/AF applications tend to accumulate widgets the same way Macros accumulate additional lines of unmaintainable code. The Tab Layout object offers a way to create more "real estate" on the screen to house new widgets. (By the way, it is easier to start with a Tab Layout object first, than to transfer all the widgets in a FRAME to a newly added Tab Layout.)

Added functionality: "Matching" ID values. (Figure 3) The Reconcile Macro looked for observations with mistyped IDs. It would then halt execution and require users to manually edit the data sets to correct the mistyped ID values. The SAS/AF application allows this editing process by invoking PROC FSEDIT without leaving the application. It is possible to invoke PROC FSEDIT from within a MACRO also, but not from within a single DATA step—like the one at the heart of the Reconcile Macro.

The FRAME entry that handles mismatched observations (Figure 3, top) uses an Extended Table object to provide for an indefinite number of objects to display.

Other FRAME entries (not shown) allowed the user to select a variable or observation, select an FSEDIT screen, and obtain on-line help.

Increased modularity: SAS/AF allows applications that are fairly modular. The ability to incorporate other SAS PROCs (such as FSEDIT, SORT, and FSEDIT) in SCL entry is a valuable feature. Other standard modular features include the limited scope of SCL variables, the ability to call one FRAME entry from within another, and the ability to define METHODS as publicly-defined subroutines.

Improved debug tools: In the Macro version of Reconcile, bugs caused by misplaced punctuation required often required extensive debugging; two or three hours to locate a single misplaced asterisk or semi-colons was not uncommon. In SCL, these bugs are pinpointed during compilation, which occurs during development time rather than run time.

SCL offers powerful aids in debugging. SCL entries allow the color-coding of SCL source code, so that, for example, corresponding DO—END statements could appear in the same color and attributes (underline or reverse). Though not necessary during the development of Reconcile, the SCL debugger is a powerful resource in locating subtle bugs—it has no counterpart in MACRO.

CONCLUSION

This case study of the life history of a SAS application might be thought of as two phases: prototyping (the development of the experimental macro version), followed by implementation in a more permanent form (the SAS/AF version). Both macro language and SAS/AF were useful at particular stages of the project. Without the ability to conjure up a reconciliation utility fairly quickly three years ago, the Unit may never have had the chance of experimenting with it. Once the concept proved its worth, then the resources could be justified in producing a more maintainable application with an improved user interface.

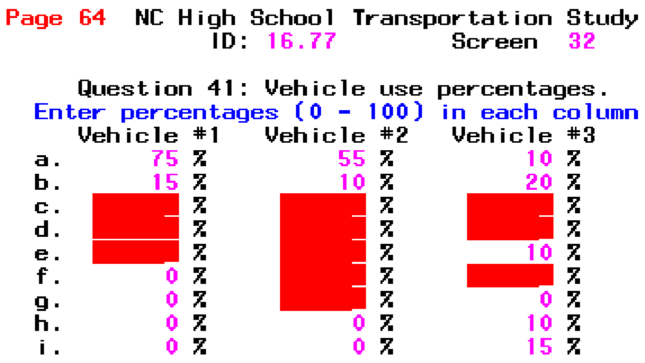
A newfound appreciation of the tradeoffs between the potential of a promising new program and the cost of maintaining that program over the long-term is a perennial benefit in every applications-development undertaking. And as long as programs are produced that justify the investment in their development, the applications-development process itself will continue to be appreciated.

NOTES

[1] The problem is the single quote in the comment. When the MACRO processor "tokenizes" this star-semicolon comment, an unclosed quote is created. The MACRO does not compile because the %MEND statement becomes part of the comment!. The problem can be solved by using slash-star comments (/* */), which are handled differently. (Or, one could double the quote marks!)

[2] A drawback only from the standpoint of the organization as a whole. From the viewpoint of the application developer, lack of maintainability could be a real benefit: Job security for the lifetime of the application!

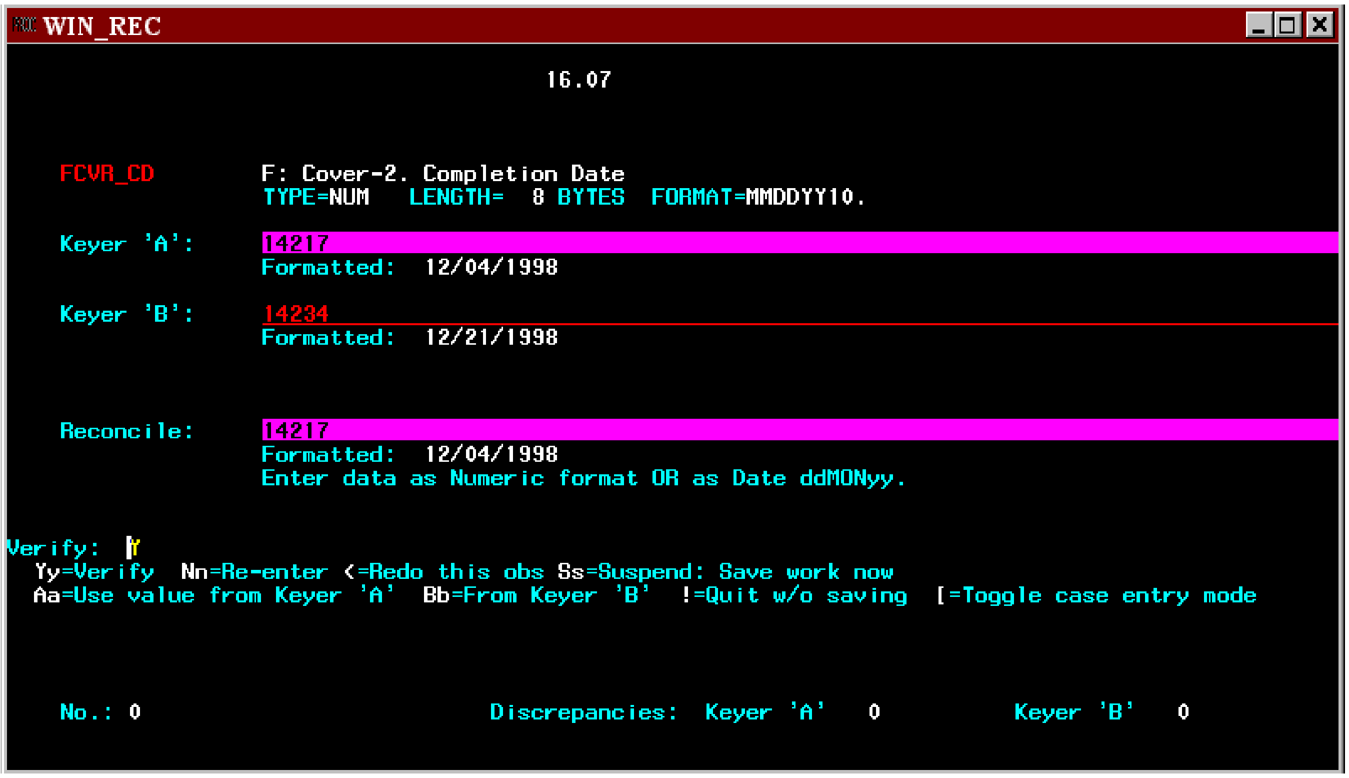
The author welcomes comments and suggestions:
The University of North Carolina
Frank Porter Graham Child Development Center
CB # 8185
Chapel Hill, NC 27514-8185
919/966-0021
davidm_gardner@unc.edu



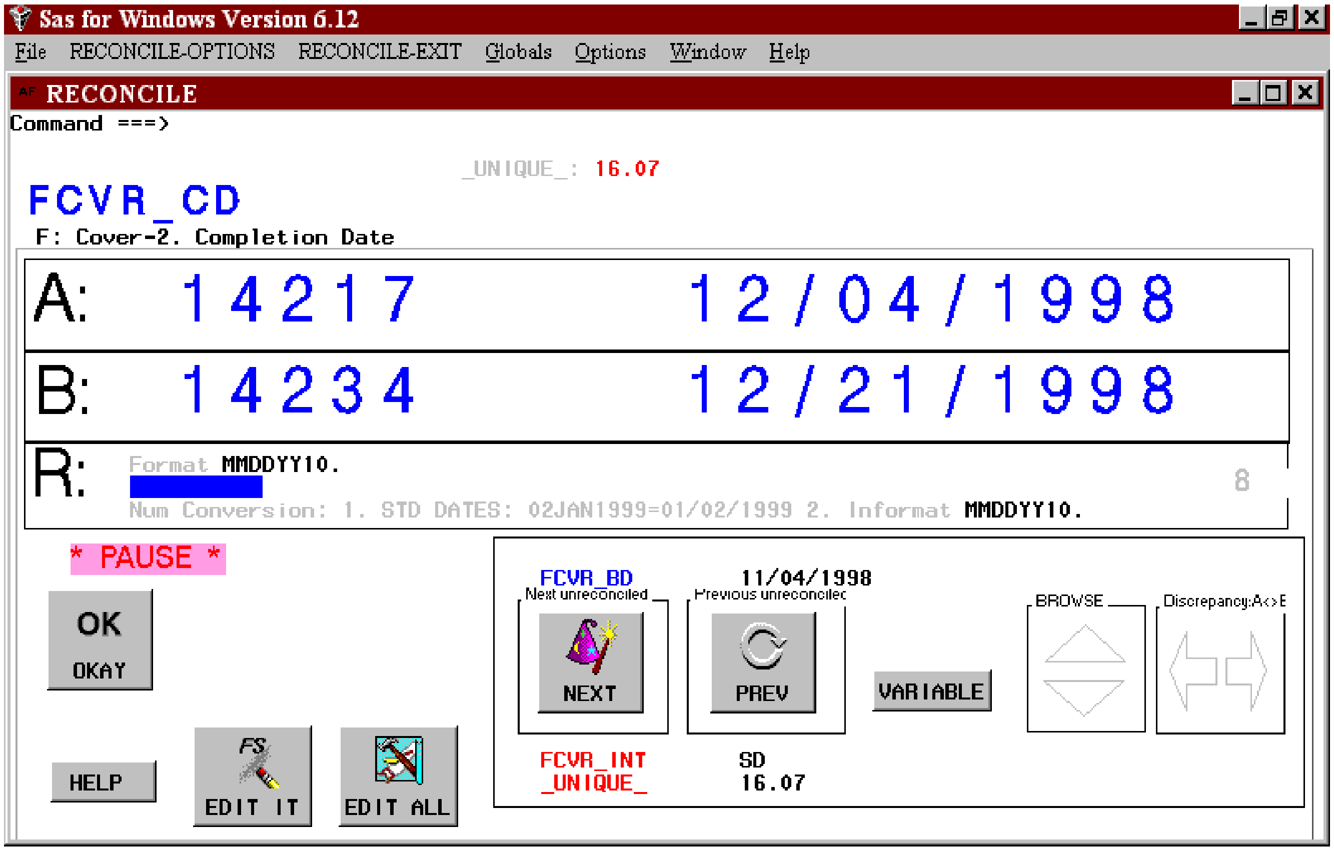
```
Page 64   NC High School Transportation Study
               ID: 16.77          Screen   32

     Question 41: Vehicle use percentages.
  Enter percentages (0 - 100) in each column
     Vehicle #1    Vehicle #2    Vehicle #3
  a.         75 %          55 %          10 %
  b.         15 %          10 %          20 %
  c.            %             %             %
  d.            %             %             %
  e.            %             %          10 %
  f.          0 %             %             %
  g.          0 %             %           0 %
  h.          0 %           0 %          10 %
  i.          0 %           0 %          15 %
```

Figure 1. Improvements made possible by SCL. This FSEDIT screen was launched from the screen at the bottom of Figure 2 on the next page. The shaded fields are discrepant values awaiting reconciliation. Sometimes seeing the patterns of discrepant data provides clues to systematic keying mistakes. (This functionality requires customization of the SCL program behind the FSEDIT screen.)

From MACRO to SAS/AF® Software: A Case Study of the Evolution of an Application



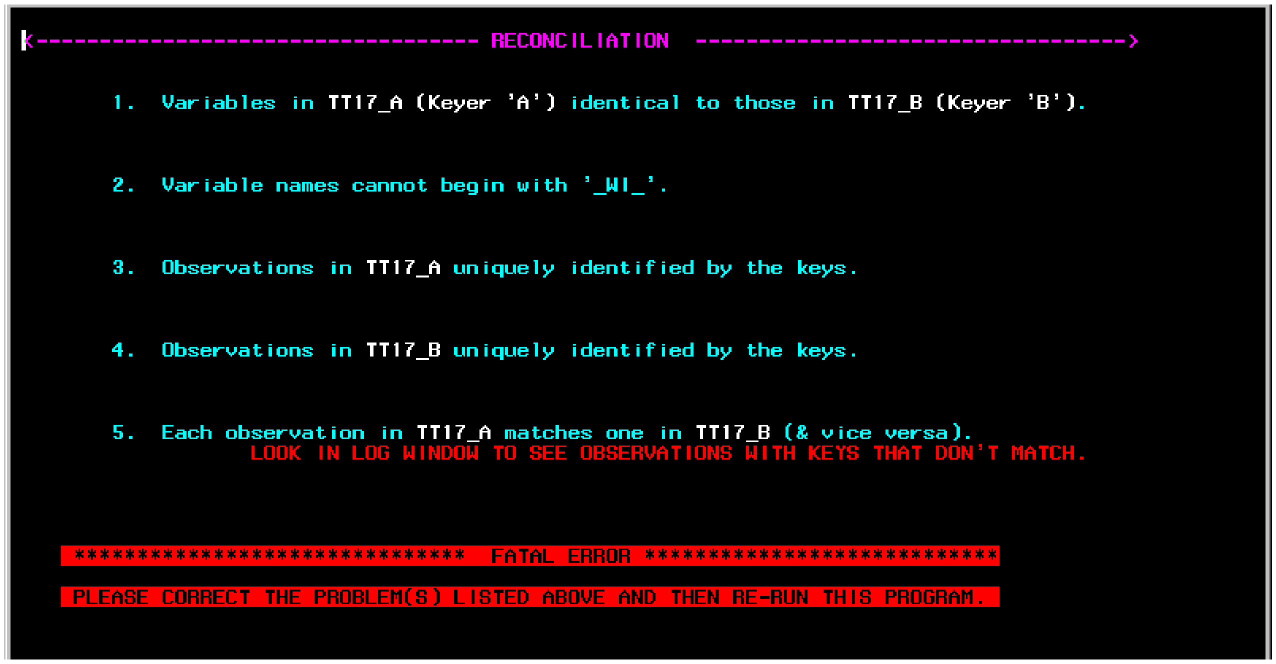
```
WIN_REC

                                      16.07


   FCVR_CD          F: Cover-2. Completion Date
                    TYPE=NUM    LENGTH=  8 BYTES  FORMAT=MMDDYY10.

   Keyer 'A':       14217
                    Formatted:   12/04/1998

   Keyer 'B':       14234
                    Formatted:   12/21/1998


   Reconcile:       14217
                    Formatted:   12/04/1998
                    Enter data as Numeric format OR as Date ddMONyy.


Verify:  Y
  Yy=Verify  Nn=Re-enter <=Redo this obs Ss=Suspend: Save work now
  Aa=Use value from Keyer 'A'  Bb=From Keyer 'B'  !=Quit w/o saving  [=Toggle case entry mode



   No.: 0                     Discrepancies:  Keyer 'A'   0         Keyer 'B'   0
```



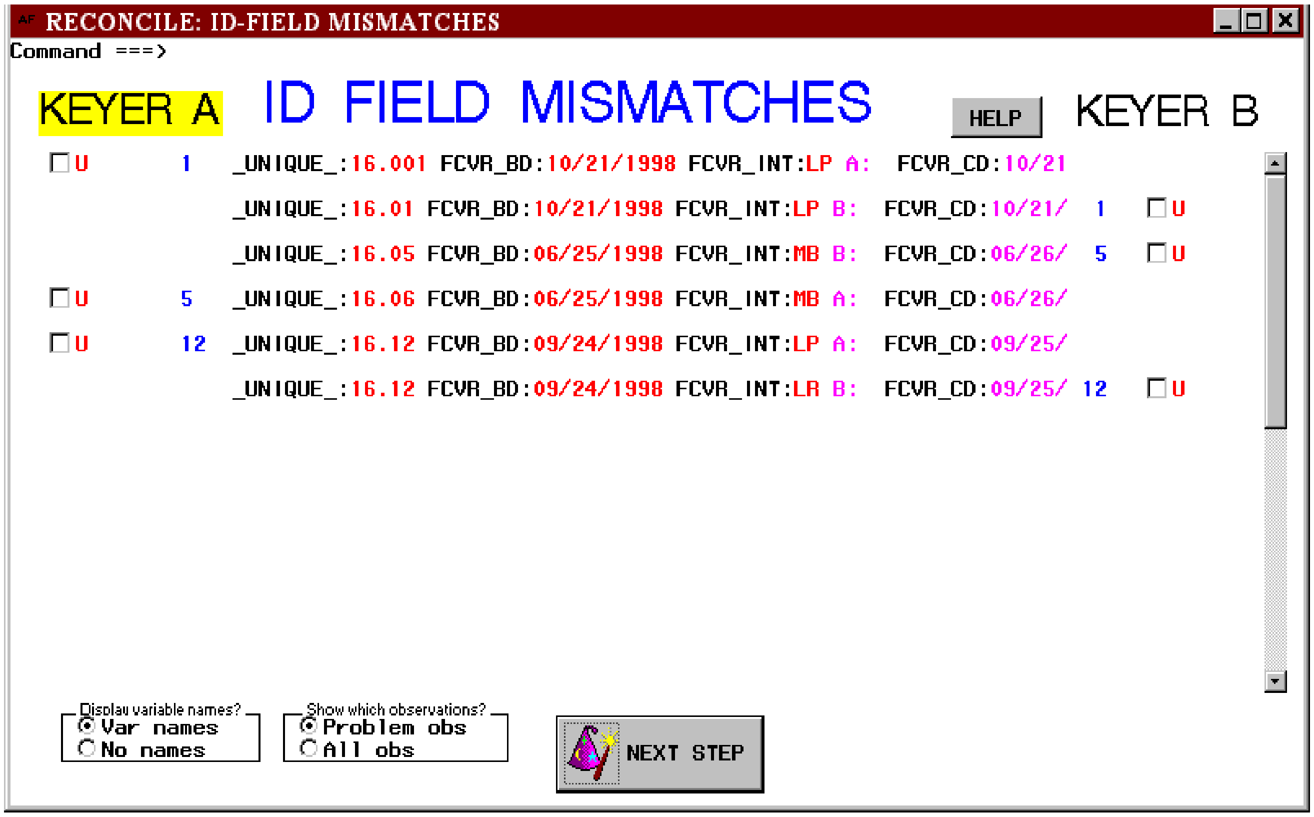
```
Sas for Windows Version 6.12
File  RECONCILE-OPTIONS  RECONCILE-EXIT  Globals  Options  Window  Help
RECONCILE
Command ===>

                               _UNIQUE_: 16.07
FCVR_CD
 F: Cover-2. Completion Date
A:    14217                    12/04/1998
B:    14234                    12/21/1998
R:    Format MMDDYY10.
                                                                 8
      Num Conversion: 1. STD DATES: 02JAN1999=01/02/1999 2. Informat MMDDYY10.
```

Figure 2: Before and After: Main reconciliation screen. *Top*: Macro version. This screen was defined on a WINDOW statement in a DATA STEP. *Bottom*: AF version. This FRAME entry contains more than 50 widgets on a TAB control. Many of the widgets are hidden at any time.

From MACRO to SAS/AF® Software: A Case Study of the Evolution of an Application



```
<---------------------------------- RECONCILIATION ---------------------------------->

       1.  Variables in TT17_A (Keyer 'A') identical to those in TT17_B (Keyer 'B').

       2.  Variable names cannot begin with '_WI_'.

       3.  Observations in TT17_A uniquely identified by the keys.

       4.  Observations in TT17_B uniquely identified by the keys.

       5.  Each observation in TT17_A matches one in TT17_B (& vice versa).
                  LOOK IN LOG WINDOW TO SEE OBSERVATIONS WITH KEYS THAT DON'T MATCH.

   ******************************  FATAL ERROR  ******************************

   PLEASE CORRECT THE PROBLEM(S) LISTED ABOVE AND THEN RE-RUN THIS PROGRAM.
```



```
RECONCILE: ID-FIELD MISMATCHES
Command ===>

KEYER A   ID FIELD MISMATCHES   HELP   KEYER B

☐U     1  _UNIQUE_:16.001 FCVR_BD:10/21/1998 FCVR_INT:LP A:  FCVR_CD:10/21
          _UNIQUE_:16.01 FCVR_BD:10/21/1998 FCVR_INT:LP B:  FCVR_CD:10/21/   1   ☐U
          _UNIQUE_:16.05 FCVR_BD:06/25/1998 FCVR_INT:MB B:  FCVR_CD:06/26/   5   ☐U
☐U     5  _UNIQUE_:16.06 FCVR_BD:06/25/1998 FCVR_INT:MB A:  FCVR_CD:06/26/
☐U    12  _UNIQUE_:16.12 FCVR_BD:09/24/1998 FCVR_INT:LP A:  FCVR_CD:09/25/
          _UNIQUE_:16.12 FCVR_BD:09/24/1998 FCVR_INT:LR B:  FCVR_CD:09/25/  12   ☐U

Display variable names?    Show which observations?
[✓ Var names]              [✓ Problem obs]
No names                   All obs
                                                      NEXT STEP
```

Figure 3: Before and after: "Matching". *Top:* Macro-based version. The user is notified about mistyped ID values. The application will terminate as soon as the user presses the ENTER key. *Bottom*: AF version. User may invoke an FSEDIT session to correct mistyped ID values. A click on a check box next to an observation takes one there.