USE OF MACROS FOR ERROR CHECKING IN RESEARCH DATA MANAGEMENT SYSTEMS

Henk Pechler
Infant Health and Development Program
Stanford University

ABSTRACT
This paper describes a system of SAS* Macros for checking and editing errors in research data. Representative applications involving inter-item consistencies and skip patterns are illustrated. The Macros are contained in a SAS AUTOCALL library and constitute a key component of the File Update System (FUS), a SAS-based data management system developed for the Infant Health and Development Program. Advantages of using the Macro Facility in this context are structured modular code resulting in efficient software maintenance. Potential dangers are compiler overflow problems and SAS programs that are difficult to read.

STRUCTURE AND FUNCTION OF ERROR CHECKING MACROS
When processing research data it is important to ensure the quality of the data gathered. A logical analysis of the data collection instrument (e.g. questionnaire or test form) may yield certain combinations of data values that are mandatory. For example, on a given test form the value "2" for variable "v1" should coincide with the value "9" for variable "v3". In SAS* source statements we can write a checking routine in the Data step that will notify us of any illegal combination of values for these two variables; for instance, a no-frills way is:

```
IF v1 EQ 2 AND v3 NE 9 THEN DO;
   PUT / recordid= "Inter-Item Consistency Error"
   / Condition 1: v1 EQ 2 
   / Condition 2: v3 NE 9 
   / Variables involved: 
   / v1= v3= ;
END;
```

Example 1

We can then provide meat to the bones of this structure with the following invocations of the newly defined macro `iicck`, by providing values to the parameters specified in line 1 of the macro defined in Example 2, such as the following:

```
iicck(condl- v1 EQ 2
   cond.2 - v3 NE 9
   iicvarl- v1
   iicvar2- v3)
```

Example 3

If an inconsistent combination of values is encountered, an error flag such as this will appear in an error report generated by the SAS program:

```
RECORDID=3379 Inter-Item Consistency Error
   Condition 1: v1 EQ 2 
   Condition 2: v3 NE 9 
   Variables involved: 
   V1= v3= 
```

Example 4

IMPORTANCE OF ERROR CHECKING MACROS
This and several other error checking routines form an integral part of a system designed to facilitate the creation of clean data sets. This File Update System (FUS) was developed for the Infant Health and Development Program (IHDP)**, a collaborative, multi-site clinical trial.
This trial was designed to test the efficacy of combining early child development services and family support with pediatric health surveillance in reducing the incidence of health and developmental problems among low birthweight, premature infants.

The end products of the FUS are cumulative datasets, stored as member datasets in SAS libraries. These cumulative datasets form the input for the creation of analysis datasets, which, in turn, feed the research analyses for the main study and ancillary studies. The error checking provides valuable information for cleaning the data, which is usually a process that requires several iterations before a cumulative dataset deserves the label 'clean' (see the Research Information Systems model, highlighted in Constantine, Shing & Pechler, 1987). (See Figure 1.)

Four programs form the backbone to the FUS, performing the following separate functions:

1. *Load-Edit:* (reads the raw data (ASCII), does error checking and creates an interim SAS dataset)

2. *Update:* (creates/updates a cumulative SAS dataset using the interim dataset)

3. *Correction:* (applies corrected variable values, stored in a transaction file, to the cumulative dataset)

4. *Cumulative-Edit:* (performs error checking and other recodes on the most recently created cumulative SAS dataset).

The error checking macros are called in both the Load-Edit and Cumulative-Edit programs from the same AUTOCALL library. The error reports produced by these two programs are used to evaluate whether a correction transaction file needs to be constructed and which corrected variable values this file needs to contain. This is used as input for the FUS Correction program to correct the cumulative dataset.

I would like to make the special note that applying the corrections occurs with the UPDATE statement. A special zap program to correct the cumulative dataset (see Henderson & Rabb, 1987) was provided by SAS Institute and applied to the version (currently V516) in production at the Stanford Data Center, so that UPDATE could handle all special missing value codes for numeric variables (including '.' through '2'), as it could in the 82.4 release. UPDATE provides a much less CPU-intensive method of making changes to a dataset than does MERGE. It would be quite valuable for the purpose of efficient data cleaning to extend this capability for the UPDATE statement to Version 6 of SAS and beyond.

Data acquired via as many as 75 different data collection instruments pass through the FUS. There are some short test forms as well as several fairly long and complex questionnaires. The ten types of errors checked for are the same for each instrument, providing another good reason (besides the occurrence of multiple inter-item consistencies within a single data collection instrument) for selecting a tool in SAS to generate multi-purpose code. These can then be stored as modules that can be called into different programs. The Macro Facility is such a tool.

**HOW SAS PROCESSES MACRO CODE**

Chapter 7, "How the SAS System Processes a Program", in the SAS Guide to Macro Processing (1986), describes in detail what SAS does internally with SAS source statements. All such statements, including macro code, are encountered by the word scanner, a part of the SAS Supervisor (see Henderson & Rabb, 1987, for a tutorial style introduction to this central component of SAS software). In a process called tokenization the word scanner breaks all statements up into tokens. When tokens of type '*' or '@' are encountered, subsequent tokens are directed to the macro processor which will resolve these tokens into regular, that is non-macro, SAS code. Subsequently SAS will attempt to compile and execute that code.

When applying the Macro Facility in the way I have shown in Examples 2 and 3, it is apparent that SAS provides, through the Macro Facility, a means to use shorthand for repetitive and difficult to read code. Good short introductions to simple and sound applications with the Macro Facility can be found in Barnes (1986) and Rosenberg (1987).

**CHECKING FOR SKIP PATTERN ERRORS**

The SAS System has another tool for checking for relatively simple error checks, such as for inter-item consistency (a.k.a. cross-field validation) and for value range violation: SAS/FSF. Data entry, however, must be interactive (as opposed to batch processing). For more complex error checks, or if detailed reporting is needed, the SAS programmer must use the Data Step.

A more complex example of an error checking routine is one that can handle skip patterns. What is a skip pattern? Not uncommonly in more complex data collection instruments, certain questions can be passed over, depending on the answers given to other, usually prior, questions. Consider, for instance, this segment from a questionnaire investigating programmers' family life situations:

```plaintext
545
```
A skip pattern is apparent in this set of variables (Qla and Qlb) most likely have sample questions. When Question 1 was answered with a "No" then Questions 1a and 1b do not need to be asked and can be skipped before proceeding with Question 2.

If the routing condition is satisfied (Q1 EQ 2, i.e. "No"), then the identified skip variables (Qla and Qlb) most likely have missing values. Most researchers will want to make a distinction between two kinds of situations for a particular variable: one where variable values are appropriately missing because the questions should have been skipped; and the other situation in which the value is missing while there should have been one. (In Example 5: if the answer to Question 1 had been "Yes", we would like to see values for the corollary variables Qla and Qlb).

At some point, appropriately missing values (as in the first situation) are going to be recoded to some special value: a number value or one of the special missing value codes (.A through .Z) which SAS allows for, indicating that these are classified missing values. This effort will have the desirable effect of isolating the illegal or 'unclassified' missing values.

The researcher will then be able to better assess whether too many unclassified missing values appear in her data. She can then take appropriate action, such as going back to try to collect more data to bring the number of unclassified missing values down, or, which is perhaps more likely, decide which variables not to include in the analysis.

In order to let SAS help clean the data in this way, this error checking routine should be able to recode validly skipped variables. This makes the skip pattern check more complicated to code than the simpler inter-item consistency check.

Examine this code:

Example 5

Invoking the macro in Example 6 would result in the following for the questionnaire segment from Example 5:

Example 6

When many skip patterns are present in a data collection instrument, it is easily seen that parameterizing the error check into a SAS Macro allows for program code (in the form of Macro invocations) that is much more compact and more easily readable.

If SAS encounters an unexpected value for a skip variable, a message such as the following will appear in the error report:

Example 7

Example 8
SAS AUTOCALL LIBRARIES

We can define these macros in the SAS program that employs them, or we can store them in an AUTOCALL library. They are then called from within a SAS program as needed. For macros that need to be called into more than one program, there is the additional great advantage of having to make changes to only one copy of the macro code when revisions become necessary. The AUTOCALL Facility is described in Technical Report P-146 Changes and Enhancements to the Version 5 SAS System, April 1986, p. 54ff. (See Figure 2.)

ADVANTAGES AND LIMITATIONS OF USING THE MACRO FACILITY

Advantages

Development of error checking code using SAS source statements can be greatly aided by employing the Macro Facility in a judicious way. It allows, through parameterization, for the implementation of a certain degree of modularization of SAS programs. That has clear benefits for maintenance of programs, especially when several programmers collaborate on a project. For alternative ways to employ the Macro Facility for skip pattern checking, see Bolotin (1985).

It is important to note that the power of the Macro Facility can be applied to go far beyond the straightforward symbol substitutions which are implemented in the error checking routines presented in this paper. That power beckons the inventive SAS programmer to create powerful and ultra-compact SAS code; but, not infrequently, that code becomes difficult to read for other programmers (see Kretzman (1986, 1987) for specific suggestions for discerningly employing SAS Macros).

Limitations

For a long and complex data collection instrument one may run into capacity problems when SAS tries to complete compilation of the program containing the many invocations of error checking macros in a single data step.

"ERROR 344" warns us that too many iterative DO loops, subroutine CALLS and/or lengthy literals prevented the compiler from successful completion. It deserves mention that non-iterative DO/END loops contribute significantly too.

Here is an example in which the limited capacity of the SAS data step may force the SAS programmer to make rather unpleasant adjustments. Take this variation on the questionnaire segment introduced in Example 5:

Example 9

Coding invocations for this pattern would save time, if done in this way:

Example 10

The macro "$skparray" in Example 11 could accommodate this shorthand invocation (just 1 instead of a lengthy cluster of 26 invocations of the earlier ' $skipck ' to accomplish the same result):

Example 11

One may run into a situation where series of clusters of multiple invocations (in Example 10: 26 times) of the ' simpler' skip macro (' $skipck ' from Example 6) will give a successful completion. However, a series of single invocations of the macro with the ARRAYed skip variables (' $skparray' from Example 11) presents the SAS compiler (at least the one implemented for the MVS/XA operating system) with an insurmountable problem. The implementation for batch under MVS/XA contains an ADCON table with an address space not

Example 11

One may run into a situation where series of clusters of multiple invocations (in Example 10: 26 times) of the 'simpler' skip macro ('$skipck' from Example 6) will give a successful completion. However, a series of single invocations of the macro with the ARRAYed skip variables ('$skparray' from Example 11) presents the SAS compiler (at least the one implemented for the MVS/XA operating system) with an insurmountable problem. The implementation for batch under MVS/XA contains an ADCON table with an address space not
The following is only applicable if one deals with a long and complex data collection instrument for which a single integral error report is needed. SAS code that consists of a series of single invocations of 'tskparray' has the decided advantage of quicker coding and easier maintenance. One has to consider, though, the consequence of having to split up the data steps to avoid "ERROR 344". Next, some merging operation on the separate reports is required to obtain the integral error report. This may prove to be more costly in programmer time. The capacity problem was encountered for the Version 5 implementation of SAS for OS and should be remedied for mainframe Version 6.

IN CLOSING

1. I hope to have made a convincing argument for using the Macro Facility to write modular, compact and easily maintainable SAS code. This is especially true in production systems in which a lot of different data are processed by at least two different, but similar programs.

2. Maximizing the Macro Facility's potential for compact SAS code has two hazards: one is obscure code that is hard for other programmers to maintain and the other is a compiler limitation encountered in Version 5 under OS.

3. There is an important advantage to being able to use in Version 5 and beyond the cost-efficient UPDATE statement for handling all special missing value codes (including '?A' through '?Z'). This is extremely helpful for data cleaning purposes in complex research data management systems. There is a zap available for Version 5 for OS (#Z5162697). If you share this concern, contact SAS Institute and express your support for keeping UPDATE fully functional.

ACKNOWLEDGMENTS

Thanks to Norm Constantine for his insistence on professional data systems and for the timelines to help make the FUS a feasible project. Thanks, also, to Mark Branagh, June Genis, Janet Lasher, John Rollin and Alex Stagner for their insightful comments on an earlier version of this paper. And special thanks to Nancy Greenwood, Marilyn Hart and Cindy Duenas for their tremendous administrative support.

The Infant Health and Development Program is supported by grants to the Department of Pediatrics, Stanford University, the Frank Porter Graham Child Development Center, University of North Carolina, and the eight participating universities by the Robert Wood Johnson Foundation, with additional support to the Department of Pediatrics, Stanford University, from the Division of Maternal and Child Health of the U.S.P.H.S., the National Institute of Child Health and Human Development, the Pew Trusts, and Stanford University Center for the Study of Families, Children and Youth.

For more information or copies of the documented SAS code contact the author at:

Infant Health and Development Program
Building 460, Stanford University
Stanford, California 94305
BITNET: MN.ZP4@STANFORD

NOTES

*SAS is a registered trademark of SAS Institute Inc., Cary, NC, USA.

**The Infant Health and Development is a collaborative effort. The following are the senior staff involved in the study:

National Study Office:
Ruth T. Gross, M.D., Director; Donna Spiker, Ph.D., Deputy Director; Norman A. Constantine, Ph.D., Director of Data Analysis; Wendy L. Kreitman, Director of Field Operations (The Stanford University Center for the Study of Families, Children and Youth.)

Program Development Office:
Craig T. Ramsey, Ph.D., Director; Donna Bryant, Ph.D., Associate Director; Joseph Spangler, Ph.D. and Barbara Wasik, Ph.D., Co-Directors of Curriculum Development; Isabelle Lewis and Claudia Lyons, Curriculum Development Specialists; Kaye Pendt, M.S.P.H., Director of Data Management and Statistical Computing (The Frank Porter Graham Child Development Center, University of North Carolina at Chapel Hill).

Participating Universities:
University of Arkansas for Medical Sciences (Arkansas); Albert Einstein College of Medicine (New York); Harvard Medical School (Harvard); University of Miami School of Medicine (Miami); University of Pennsylvania School of Medicine (Pennsylvania); University of Texas Health Science Center at Dallas (Texas); University of Washington School of Medicine (Washington); Yale University School of Medicine (Yale).

Site Directors:
Patrick R. Casey, M.D., Arkansas; Cecelia M. McCarten, M.D., Einstein; Michael W. Yogan, M.D., Harvard; Charles R. Hauck, M.D. and Keith G. Scott, Ph.D., Miami; Judith Bernbaum, M.D., Pennsylvania; Jon R. Tyson, M.D. and Mark Swanson, M.D., Texas; Clifford J. Sells, M.D. and Forrest C. Bennett, M.D., Washington; David T. Scott, Ph.D., Yale.

Education Directors:

Settye Caldwell, Ph.D., Educational Consultant, Arkansas; Ruth Turner, Ed.D., Dallas Independent School District Liaison, Texas.
REFERENCES


