

A SAS® Macro System to Automate the Decoding and Validation of Data

Carl R. Haske, STATPROBE, Inc., Ann Arbor, Michigan

Kayhan Najmabadi, STATPROBE, Inc., Ann Arbor, Michigan

ABSTRACT

This paper presents an application that was implemented to solve a problem in transferring data between SAS and Microsoft Access®. The macros and AF application that were developed have gained application beyond their original intent.

When data are transferred between SAS and other platforms, a problem arises if the data variables have associated formats. Typically, when ODBC is used, the translated data contain coded values rather than the formatted values of data. One component of this system is to automate the decoding process by using the formatted values of the data.

There is also a problem of validating the data transfer. When critical data are transformed to a different file format, one must be able to guarantee the accuracy of the data. This validation process typically involves some type of comparison of the translated data to the original data. The second component of the system presented here involves automating this process of comparison and producing a validation report.

INTRODUCTION

Efficient SAS databases typically use numeric codes for variables that have a discrete set of values. Numeric codes use less storage space on disk than long strings of character text. For example, data for a clinical trial may store information relating to adverse effects of a treatment. The adversely affected body system needs to be included in the data, and it is more efficient to store numeric codes for body systems than the actual description of the body system (see table 1).

Table 1. Example of Coding

Code	Body System Description
1	Body as a whole
2	Central nervous system
3	Gastrointestinal system
4	Musculoskeletal system
5	Cardiopulmonary system

The problem arises when source data stored in SAS are transferred to a different file format. A customer, QLT PhotoTherapeutics Inc., requested us to convert multiple SAS databases to Microsoft Access databases and decode all variables that had an associated format. The resulting database was to act as a data dictionary for a clinical trial database.

The prospect of manually scanning through the SAS data for formatted variable and developing custom programs to export each data file was intimidating. Instead, we opted to automate the process and take a data-driven approach.

This paper presents an application that automatically decodes all data in a data library and performs a validation process comparing two databases. The system is macro based and can be either initiated by an external system or accessed via a SAS/AF® interface.

The first section discusses the decoding macros used in the system. The second section describes the validation process. Some utilities that use the SAS dictionary tables are discussed in the third section. Finally, we discuss the AF interface and make some concluding remarks.

This paper discusses a component of a larger system presented in the authors' "Automating Data Transfer from SAS to Microsoft Access," the first paper in the present group of three. Related work can also be found in the authors' "Interfacing Microsoft Access and SAS Using OLE and ODBC," the last paper in the present group.

DECODE MACRO LIBRARY

Two macros described in table 2 are used for the purpose of decoding data.

Table 2. Decoding Macros

Macro Name	Macro Description
DECODEDS	Decodes a single data set
DECODELB	Decodes all data sets in a data library

```
%Macro decodeds (
    inds=inds,
    outds=outds,
    label=,
    varcnt=0,
    codetype=put);
%Local i;
Data &outds
    (label="&label"
    %Do i=1 %to &varcnt;
        drop=&&var&i rename=(__qzx&i=&&var&i)
    %End;
    );
Set &inds;
    %Do i=1 %to &varcnt;
        %If &&code&i ne %then %do;
            __qzx&i=&codetype(&&var&i,&&code&i..);
        %End;
        %Else %do;
            __qzx&i=&&var&i;
        %End;
    %End;
Run;
%Mend decodeds;
```

Listing 1. DECODEDS Macro

The code for the macro DECODEDS is in listing 1. The general purpose of this macro is to derive a new data set from an existing data set, using format or informat values to decode or encode the values of variables where appropriate. To accomplish this task, the macro sequentially checks all variables from the existing data set and detects if the variables have attached coding. When coding is detected for a given variable, the macro translates the variable value and stores the proper translated value in the derived data set that corresponds to the coded value. The macro uses five parameters (see table 3) that must be specified by the calling procedure.

Table 3. Parameters of DECODEDS

Parameter Name	Parameter Description
inds	Input data set
outds	Output data set
label	Input data set label
varcnt	Number of variables in the input data set
codetype	Name of coding function to employ

The default behavior of the macro is to transform coded variables into decoded variables using the PUT function by default. The coded variables are then dropped, and the new decoded variables are renamed with the same name as their corresponding coded variable. Diagram 1 illustrates the process flow of the macro.

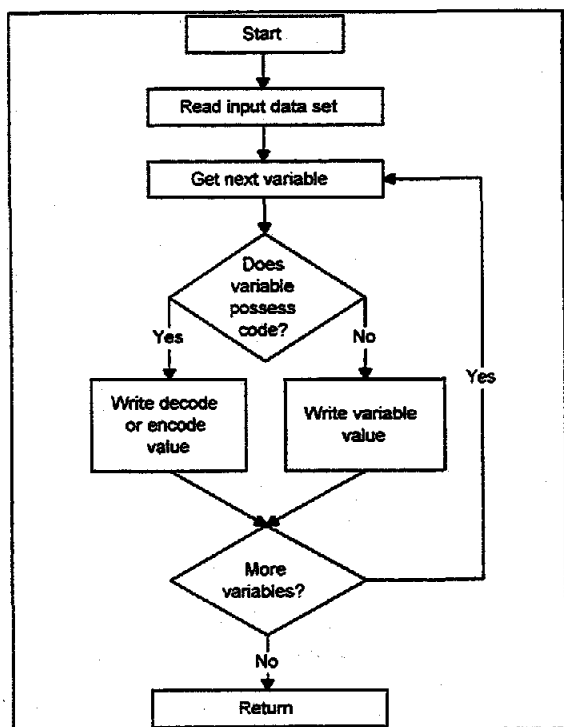


Diagram 1. Process flow of DECODEDS

The macro creates the new data set by reading the existing data set and translating the variables that have an as-

signed code. This macro is used for both decoding variables and validating decoded variable values. The flexibility of passing the coding function, either PUT or INPUT, allows for data that are stored in decoded format to be encoded by the macro and compared with the original data values.

Before the macro is called, the SAS dictionary tables are used to determine how many variables are in the input data set and which variables have assigned formats. This information is stored in global macro variables that DECODEDS references.

The macro DECODELB uses the SAS dictionary tables to create a list of data sets in a given library. Then it loops through all data sets and invokes the macro DECODEDS. The process flow for DECODELB is shown in diagram 2.

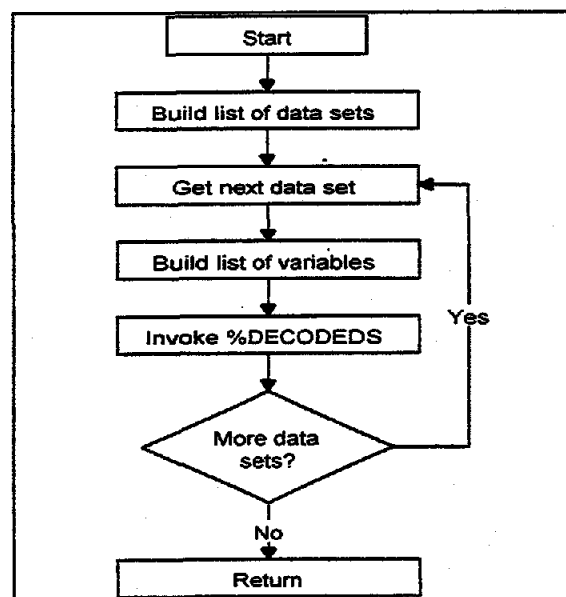


Diagram 2. Process flow of DECODELB

The macro code is in listing 2. The macros DSLIST and VARLIST are explained below in the section "SAS Dictionary Macros."

```

%Macro decodelb(lib=lib,outlib=WORK);
%Local i;
%dslist(lib=&lib)
%Do i = 1 %to &dsCnt;
  %varlist(lib=&lib,
    ds=&ds&i,
    varname=var,
    codename=code,
    codetype=format,
    count=varcnt)
  %decodeds(inds=&lib..&ds&i,
    outds=&outlib...%substr(&ds&i,
      1,%length(&ds&i)-
      %eval(%length(&ds&i)>7)),
    varcnt=&varcnt);
%End;
%Mend decodelb;
  
```

Listing 2. DECODELB Macro

In order for this system to work properly, macros that use the SAS dictionary tables are invoked to create lists of data sets, data set labels, and data set variables. These lists are stored in *macro arrays*. Macro arrays are a common informal programming structure in SAS that consists of a prefix for the array and a macro variable that stores the number of elements in the array. Macro arrays allow for loops to perform like processing on all elements of the array. Table 4 describes the various lists used by this system. The macros that create these lists are described below in the section "SAS Dictionary Macros."

Table 4. Lists Used in Macro Arrays

Item	Description
ds	Prefix for the data set list macro array.
dscnt	Macro variable to store the number of data sets.
var	Prefix for the variable list macro arrays.
varcnt	Macro variable to store the number of variables in a given data set.

VALIDATION PROCESS

A typical data transfer process needs to be validated by comparing derived data with source data and scanning for differences. In our system, we transfer the source data from SAS to MS Access using ODBC, and then transfer the data back to SAS and compare the results with the source data.

The macro we use for the transfer is built around PROC COMPARE and is used to compare imported data with original decoded SAS data. The macro first builds a list of all data sets in a library and then sequentially validates each data set. The validation procedure used is a basic test using PROC COMPARE. The resulting report shows any differences between the structure and content of the two databases. If differences are found, they are investigated to determine if there is impact to the accuracy of the data transfer. The code for the macro is in listing 3.

```
%Macro validate(base=base,compare=compare);
%Local i;
%dslist(lib=&base)
%Do i = 1 %to &dscnt;
  Proc compare base=&base..&&ds&i
    compare=&compare..&&ds&i;
  Run;
%End;
%Mend validate;
```

Listing 3. VALIDATE Macro

The parameters to this macro, *base* and *compare*, coincide with the PROC COMPARE nomenclature. *Base* designates the source library and *compare* is the library of derived data sets.

SAS DICTIONARY MACROS

Two general-purpose macros that use the SAS dictionary tables are employed in this system. The first macro, DSLIST, builds a list of data sets in a given library and stores the list as a macro array. This list is used by a

calling routine with the intent to process the data set sequentially in a loop. The second macro, VARLIST, builds a list of variables in a given data set and stores the list as a macro array. The code for DSLIST is in listing 4.

The DSLIST macro first scans a given library and creates a table of names of the data sets in the library. The macro creates global macro variables to hold the names of the data sets; it then executes a data procedure and uses the SYMPUT function to store the names.

```
%Macro dslist(
  lib=WORK,
  memtype=DATA,
  name=ds,
  label=lbl,
  count=dscnt);
%Local i;
%Global &count;

%Let lib=%upcase(&lib);
%Let memtype=%upcase(&memtype);

%* Error check *;
%If &memtype ne DATA or
  &memtype ne VIEW %then
  %Let memtype=DATA;

%* Count the number of members *;
Proc sql noprint;
  Select count(memname) into: &count
    From dictionary.tables
    where libname in("&lib") and
      memtype="&memtype";
Quit;
%If &&&count=0 %then %goto FINISH;
%* Create a view of members *;
Proc sql noprint;
  Create view ___tb as
    Select memname, memlabel
    From dictionary.tables
    where libname in("&lib") and
      memtype="&memtype";
Quit;

%Do i=1 %to &&&count;
  %Global &name&i;
  %Global &label&i;
%End;

%* Store the member names and labels *;
Data _null_;
  Set ___tb;
  call symput("&name"||trim(left(_n_)),memname);
  call symput("&label"||trim(left(_n_)),memlabel);
Run;

Proc datasets mt=VIEW nolist;
  delete ___tb;
Quit;

%FINISH;
%Mend dslist;
```

Listing 4. DSLIST Macro

The general purpose of the VARLIST macro is to build a macro array of variable and format names of a given data set. This macro also uses the SAS dictionary tables, and the structure of its program flow is similar to that of the DSLIST macro. The code for the VARLIST macro appears in listing 5.

The code for VARLIST first counts the number of variables, then stores the variables in macro arrays using a temporary view and the SYMPUT function.

```
%Macro varlist(
  lib=lib,
  ds=ds,
  varname=var,
  codename=code,
  codetype=format,
  count=varcnt);
%Local i;
%Global &count;

Proc sql noprint;
Select count(name) into: &count
From dictionary.columns
where libname=upcase("&lib") and
      memname=upcase("&ds") and
      memtype="DATA";
Create view __col as
Select name, &codetype
From dictionary.columns
where libname=upcase("&lib") and
      memname=upcase("&ds") and
      memtype="DATA";
Quit;

%Do i=1 %to &&count;
  %Global &varname&i;
  %Global &codename&i;
%End;

Data _null_;
Set __col;
call symput("&varname"&i||trim(left(_n_)),name);
call symput("&codename"&i||trim(left(_n_)),&codetype);
Run;

Proc datasets nolist;
delete __col;
Quit;
%Mend varlist;
```

Listing 5. VARLIST Macro

AF INTERFACE

This system was designed to execute either remotely from an external application such as MS Access, using OLE, or within the SAS system, by use of an AF interface. Figure 1 shows the AF interface to the decode macro that gets the input and output data libraries. This part of the application allows the user to specify input and output data libraries for the decoding process. The user input is passed to the DECODELB macro to execute the decoding process.

Figure 2 shows the facility that interfaces to the macro that decodes a single data set. The user selects a library, a

specific data set to decode, and a target library. The application then builds a list of variables, using VARLIST, and passes the user input to the DECODEDS macro for processing.

Figure 1. Decode facility

Figure 2. Decoding a single data set

CONCLUSION

The system presented here was developed as a component of a larger system to share data between SAS and MS Access. During design of the SAS components of the larger system, it was determined that the components had independent functionality of their own merit. Developing the AF interface came later as a tool to provide end users access to the macro system.

REFERENCES

Haske, Carl R. (1997), "Application Development Templates in SAS/AF," *Proceedings of the Twenty-Second Annual SAS® Users Group International Conference*, 7-12, Cary, NC: SAS Institute Inc.

Najmabadi, Kayhan, and Haske, Carl R. (1997), "Automating Data Transfer from SAS® to Microsoft Access®," *Proceedings of the Eighth Annual Conference of the Midwest SAS® Users Group* (this volume).

Najmabadi, Kayhan; Haske, Carl R.; and Xu, Raymond (1997), "Interfacing Microsoft Access® and SAS® Using OLE and ODBC," *Proceedings of the Eighth Annual Conference of the Midwest SAS® Users Group* (this volume).

SAS Institute, Inc. (1993), *SAS/AF® Software: FRAME Entry, Usage and Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute, Inc. (1994), *SAS® Screen Control Language: Reference, Version 6, Second Edition*, Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

The authors thank QLT PhotoTherapeutics Inc. for providing specifications and review of Microsoft Access databases.

In addition, the authors thank Paul Schwankl for assistance in the preparation of this paper.

SAS and SAS/AF are registered trademarks of SAS Institute Inc. in the USA and other countries. Windows 95 and Microsoft Access are registered trademarks of Microsoft Corporation. * indicates USA registration.

AUTHORS' ADDRESSES

Carl R. Haske, Ph.D.
STATPROBE, Inc.
3885 Research Park Drive
Ann Arbor, MI 48108
(313) 769-5000 x115
E-mail: chaske@statprobe.com

Kayhan Najmabadi, Ph.D.
STATPROBE, Inc.
3885 Research Park Drive
Ann Arbor, MI 48108
(313) 769-5000 x161
E-mail: knajmabadi@statprobe.com