Paper 225-26

Input File Cleaning and Screening Using a SAS® Macro

Mike Tangedal, US Bank, St. Paul, MN

ABSTRACT

Building a thorough logic table noting all screening and quality checks to be put in place for data input files of a known structure can pay great dividends. Using Base SAS and the logic in the table, this macro serves to help build a detailed quality enhancement tool with minimal effort. The key is to apply generalizations in terms of standard SAS code based on summary category codes. Format tables as well as capturing output from Proc Freq and Proc Means is also used. The result is a input file summary table listing each variable along with the number of missing observations and summary information such as range, list of values, and number of values equal to zero. The result of this macro is a data set containing the input file information plus flags set by the logic table. Also created are reports showing examples of anomalies in the input file and frequency reports of categorical variables.

INTRODUCTION

Before employing any of the wondrous modules of the ever-improving SAS system on the volumes of information facing any programmer, the drudgery of scrubbing the source data must often be addressed, especially when dealing with data from outside sources. This need not be such an arduous task! A logic table noting all checks and requirements of the various elements within the data set can be manipulated into code to perform these checks. Generalizations can be translated into quick and easy solutions. Employing this code in a one-pass data statement can produce valuable summarizations using minimal processing time. This paper serves to describe a SAS macro to perform this function as well as underlying principles involved.

INPUT

This code works best when dealing with the scenario of regular processing of data files of a set defined structure. Whether they be internal or from an external source is not important of itself – external files are just assumed to be outside of control until they are received. The logic and code here assume the input file is a flat file, but logic works with slight modifications if input is a SAS data set.

Employment of initial logic to ensure the cleanest data possible is of course a highly desirable scenario. Initial prevention and detection reaps exponential rewards in the long run. This code also serves as an outline on which further logic constructs can be easily employed.

LOGIC TABLE

The focal point of this data screening process is the logic table. A logic table for each file element can be created separately or added to an existing file structure table. The format for this table is not important as long as it is readable by the SAS macro. Elements in the table translate to applied processing logic in getting this information into a reliable SAS data set structure. The following is a list of all elements in a field definition table. Note that elements deal with both physical layout of the input file as well as logic elements.

Field Name	- SAS construct
Field Label	- initial definition of field labels is beneficial to subsequent reporting
Field Length	- input file structure
Start Position	- input file structure
Informat	- input file structure
Default value	- value set for blank entries
Required value	- Y/N
Maximum	- maximum allowable value (applicable to numeric and date entries)
Minimum	- minimum allowable value (applicable to numeric and date entries)
Validation Logic	- additional logic (SAS statements) to be applied such as intra-variable relationships
Validation Text	- text to be placed in message variable for discrepancies from validation logic
Critical error	- Y/N denotes whether discrepancies result in entire record flagged as unusable
Key field	- Y/N denotes whether field is part of unique identifier
Frequency table	- field or field combination used in showing categorical contents of this field.
Field type	- A, T, U, F, N, R, D, C, S denotes method of summarization

Field type codes can be used to apply generalizations to types of entries in order to standardize and simplify processing. Here is a table listing summary codes and descriptions.

Summary Code	Description
(A)lpha	Variable contains only alpha characters (i.e. name fields)
(T)ext	Variable contains any apha-numeric entry
(U)nique	Variable contains just a few possible entries (i.e. sex of 'M','F', Boolean)
(F)ormat	Contents of variable in lookup list / format table
(N)umeric	Variable should be capable of being used in an arithmetic capacity
(R)ange	Variable composed of numeric codes, normally sequential
(D)ate	Date entries
(C)ategorical	Variable contents unique but not readily definable
(S)pecial	Validation logic and/or additional logic applies to variable

SUMMARIZATION PROCESS

With the application of a summary code as well as the other variables in the logic table to each variable in the source file, much of the coding associated with quality checks becomes almost automated. Most of the rest of the process simply involves snapping in the correct summarization code.

The first step in this SAS macro is to read the input file into its predefined format. This is done through the use of the data definition table also containing the logic table constructs used later. The following is the initial section of the SAS macro.

```
/* Quality check macro used in summarizing
   flat files */
%MACRO EDITFLAG(inset=filein,libin=inlib,
  libout=perm,outset=flags,debug=N);
  /* 'inset' is the location of the flat file
     to be read */
  /* 'libin' is the libname plus the data set
     name for the logic table*/
  /* 'libout' is the libname used to store
     resulting datasets  */
  /* 'outset' is the name of the resulting
     dataset */
  /* 'debug' is used to create testing
     reports throughout the program */

  %* Set debug parameters when testing
     changes in macro;
  %IF &DEBUG=Y %THEN %DO;
    OPTIONS MPRINT NOSYMBOLGEN MTRACE;
  %end;

  TITLE1
  "Dataset Summary Report for: &inset";

  ** This is a temporary file used to store
     the input statement used in reading the
     raw claims;
  filename temp '~/tempdset.dat';
  *** (Unix directory structure);

  ** Read the file definition table to build
     an input statement;
  data _null_;
    set &libin
   (keep = startpos varname varfmt) end=last;
    ** startpos equates to Start Position;
    ** varname equates to Field Name;
    ** varfmt equates to Informat
       from logic table section;
    file temp old;
    if _N_ = 1 then put "input";
    ** building variable inputs;
    put "  " startpos varname varfmt;
    if last then put "   ;";
  run;

  **** Build input dataset;
  data raw;
  ** read file using input statement built
     above;
    infile &inset;
    %include temp;
  run;

  %IF &DEBUG=Y %THEN %DO;
    title2 'Contents of raw dataset';
    proc contents data = raw; run;
  %END;
```

INITIALIZATION

Initialization involves several different elements - created variables, a temporary array used in summarization, temporary files used to store SAS data step statements, and three data sets – a summary data set created from the temporary array, an examples data set, and the input data set with flags. Of course, the need for these temporary files created below will become unnecessary with the changes in System 8 for SAS. Feel free to modify and improve!

```
 ** TEMPORARY FILES TO HOLD VARIABLE LISTS
    ARE NOTED AS:
 * FMTTEXT  - HOLDS CODE FOR DATA STEP
   SUMMARIZATION;
 * FREQTEXT - HOLDS CODE FOR THE PROC FREQ
   TO GET UNIQUE VALUES;
 * FREQLIST - HOLDS RESULTS OF PROC FREQ;
 * MEANTEXT - HOLDS CODE FOR THE PROC MEANS
   TO GET NMISS, MIN, MAX, MEAN, AND STDEV;
 * MEANLIST - HOLDS RESULTS OF PROC MEANS;
FILENAME FMTTEXT  '~/fmttext.dat';
FILENAME FREQTEXT '~/freqtext.dat';
FILENAME FREQLIST '~/freqlist.dat';
FILENAME MEANTEXT '~/meantext.dat';
FILENAME MEANLIST '~/meanlist.dat';

/*****************************************
From looking at logic table for examined
Data set, build lists for further analysis.
Retain variables names for those variables
with unique contents.
Retain a variable names list for those
numeric variables.
Retain a list of variable names and table
type for those categorical variables.
Also, build data step statements for other
forms of analysis.
******************************/
data unique (keep=varname typevar)
  number (keep=varname)
  cat (keep=varname frq);
  set &libin
    (keep = varname summary label frq
            varfmt minn maxx default
            vlogic vtext key)
    end=last;
  ** summary equates to Field Type;
  ** label equates to Field Label;
  ** frq equates to Frequency table;
  ** minn equates to Minimum;
  ** maxx equates to Maximum;
  ** default equates to Default value;
  ** vlogic equates to Validation Logic;
  ** vtest equates to Validation Text;
  ** key equates to Key field
     in Logic Table section;
  counter + 1;  ** used in temporary array;
  length typevar $4 keystr $80;
  retain keystr;
  if substr(varfmt,1,1) = '$' then
    typevar = 'Char';
  else typevar = 'Num';

  **********************;
  * Build field lists for numeric, unique,
  * and categorical variables;
  **********************;
  if summary = 'U' then output unique;
  else if summary = 'N' then output number;
  else if summary = 'C' or
    (summary ne 'F' and frq ne ' ') then
    output cat;

  **********************;
  * Build field list for key identifier
  * by which to sort final dataset;
  **********************;
  if key = 'Y' then keystr =
    trim(keystr) || ' ' || varname;
  if last then
    call symput('keystr',keystr);
```

SUMMARIZATION

The next step is to build as many of the summarization statements as possible based on the summary code variable. The results of the SAS statements will be stored in a temporary array not yet created. Efficiency here is gained by presumptions that variables with a similar summary code can be checked for quality in a similar manner. This process allows for exceptions, but works best when most of the variables follow the pre-described format. Note that only with certain summary codes can generalizations translating into SAS statements be made. Also note that references are made to the data set 'Errors' which contains examples of flagged records. The following is a task list met in the SAS code from the logic table.

1. Build summarization statements to count number of blank entries for each variable.
2. If field is blank and default value exists, set default value
3. For field types of 'Alpha', count number of entries with non-alpha characters and set flag for these records.
4. For field types of 'Format', count number of entries not in format lookup table and set flag for these records.
5. For field types of 'Range', update minimum and maximum values accordingly and set flag for entries outside range of allowable values.
6. For field types of 'Date, update minimum and maximum values accordingly, format range to display in text format, and set flag for entries outside range of allowable values.

```
** build data step summarization code
   statements;
file FMTTEXT old;

***********************;
* Build summarization statements to count
* number of missing entries;
* and to set default values;
***********************;
**** Get number of missing observations
     for character variables;
if typevar = "Char" then do;
  put "if " varname " = ' ' then do;";
  put "  summ{" counter ",1} + 1;";
  if default ne ‘ ‘ then
    put "  if " varname "= ' ' then "
      varname "= '" default "';";
  put "  end;";
  end;
else do;  **** numerics;
  **** get number of observations equal
       to zero for numerics;
  if summary = 'N' then
    put "if " varname " = 0
      then summ{" counter ",1} + 1;";
  **** get number of missing observations
       for other numeric variables;
  else put "if " varname " =.
    then summ{" counter ",1} + 1;";
  **** set default value for numeric if
       applicable;
  if default ne ‘ ‘ then
    put "if " varname " =. then "
      varname "=" default ";";
  end;

***********************;
* For field type = Alpha, count number of
* observations with a non-alpha character
* and set flag for this record;
***********************;
if summary = 'A' then do;
  put "if " varname " ne ' ' and ";
  put "  verify(" varname “,”;
  put
   ",.-abcdefghijklmnopqrstuvwxyz
    ABCDEFGHIJKLMNOPQRSTUVWXYZ''')
   >0 ";
  put "  then do;";
  put "  summ{" counter ",4} + 1;";
  ** retain up to five examples of obs
  not meeting criteria for this variable;
  put "  if summ{" counter ",5} < 5 then
    do;";
  put "    summ{" counter ",5} + 1;";
  put "    message = '" varname
    " not alpha';";
  put "    output errors; end;";
  put "  end;";
  end;

***********************;
* For field type = Format, count number
* of observations not in format lookup
* table.;
***********************;
if summary = 'F' then do;
  put "if " varname " ne ' ' “;
  put “and put(" varname "," frq ") =
    'Invalid Code' then do;";
  ** assumption here is that format table
     contains formats where OTHER is
     always defined as Invalid Code;
  put "  summ{" counter ",4} + 1;";
  put "  if summ{" counter ",5} < 5 then
    do;";
  put "    summ{" counter ",5} + 1;";
  put "    message = '" varname
    " :Invalid Code';";
  put "    output errors; end;";
  put "  end;";
  end;

***********************
For field type = Range, update minimum
and maximum values accordingly;
***********************;
if summary = 'R' then do;
  *** Get range of character variable;
  if typevar = 'Char' then do;
    put "summ{" counter ",2} =
      min(summ{" counter ",2},
      input(" varname ",15.));";
    put "summ{" counter ",3} =
      max(summ{" counter ",3},
      input(" varname ",15.));";
    ****  check against minimum value;
    put "if " minn " ne. then do;";
    put "  if input(" varname ",15.) < "
      minn;
    put "    and summ(" counter ",4} < 5
      then do;";
    put "    summ{" counter ",4} + 1;";
    put "    message = '" varname
      " :below min value';";
    put "    output errors; end;";
    put "  end;";
    ****  check against maximum value;
    put "if " maxx " ne. then do;";
    put "  if input(" varname ",15.) >
     " maxx;
```

```
      put "    and summ(" counter ",5} < 5
        then do;";
      put "    summ{" counter ",5} + 1;";
      put "    message = '" varname
        " :above max value';";
      put "    output errors; end;";
      put "  end;";
      end;
    end; **** range variables;

  ***********************
  For field type = Date or numeric ranges,
  update minimum and maximum values
  accordingly;
  ***********************;
  if summary = 'D' or (summary='R' and
    typevar='Num') then do;
    put "summ{" counter ",2} =
      min(summ{" counter ",2},
      " varname ");";
    put "summ{" counter ",3} =
      max(summ{" counter ",3},
      " varname ");";
    ****  check against minimum value;
    put "if " minn " ne. then do;";
    put "  if " varname "< " minn "
      and summ(" counter ",4} < 5 then
      do;";
    put "    summ{" counter ",4} + 1;";
    put "    message = '" varname
      " :below min value';";
    put "    output errors; end;";
    put "  end;";
    ****  check against maximum value;
    put "if " maxx " ne. then do;";
    put "  if " varname "> " maxx "
      and summ(" counter ",5} < 5 then
      do;";
    put "    summ{" counter ",5} + 1;";
    put "    message = '" varname
      " :above max value';";
    put "    output errors; end;";
    put "  end;";
    end;

  *************************
  For field type = Special, apply SAS code
  stored in logic table;
  *************************;
  if summary = 'S' then do;
    put "if " vlogic " and
      summ(" counter ",4) < 5 then do;";
    put "    summ{" counter ",4} + 1;";
    put "    message = '" vtext "';";
    put "    output errors; end;";
    put "  end;";

  if last then call symput('dimm',counter);
run;

%IF &DEBUG=Y %THEN %DO;
  TITLE2
  "listing of data sets containing unique,
   cat, and number";
  proc print data = unique;
  proc print data=cat;
  proc print data=number; run;
%END;
```

At this point in the execution of the SAS macro, SAS statements derived from summary code generalizations would have been stored in a temporary file. These statements are to be applied in the next data step along with SAS statements stored as special validation in the logic table.

These statements are applied to the input data set. Created in this data step are three different data sets – one holding the summary results from the temporary array, another consisting of examples of flagged records, and the third data set consists of the original input data set with various flags attached.

```
 /*******************************************
Apply these data summarization statements
to the data set.
VARLIST is built from a temporary array
storing summarization information from each
variable in the data set.  This information
is used in a later report.
ERRORS contains examples of obs where
values of variables in the data set were
outside the accepted criteria.  These obs
are listed at the end of this program.
**************************************/
data varlist
  (keep = nmiss zero min max sumnum flag)
     errors
  (drop = nmiss zero min max sumnum flag i)
     &libout..&outset
  (drop=nmiss zero min max sumnum flag i);
  array summ{&dimm,5} _temporary_;
  length message $40 flag $1;
  set raw end=last;
  *** pull in created statements from
      processing logic table;
  %include fmttext;

  /****************************************
  Code is placed here setting various flags
  based on field values.
  For example, a flag might be useful to
  note all records where a certain anomaly
  occurs such as sex codes not equal to 'M'
  or 'F'.
  *********************************/
  output &libout..&outset;

  *** upon processing of full data set,
      create summarization from array
      values;
  if last then do;
    do i = 1 to &dimm;
      nmiss = summ{i,1};
      zero = summ{i,1};
      min=summ{i,2}; max=summ{i,3};
      sumnum=summ{i,4};
      if summ(I,4) > 0 or
        summ(I,5) > 0 then flag = 'Y';
      output varlist;
      end;
    end;
  run;

%IF &DEBUG=Y %THEN %DO;
  TITLE2 "example of flags data set";
  proc print data = flags (obs=10); run;
  TITLE2 "example of varlist data set";
  proc print data = varlist (obs=10); run;
%END;

/***  transform summary listing  *****/
```

```
data ctent
  (drop=sumnum min max flag required);
  merge varlist &libin
  (keep= varname summary varfmt label crit
         required startpos);
  ** crit equates to Critical error;
  ** required equates to Required value
     from Logic Table section;

  length sumtext $40 mintext maxtext $15;
  if summary = 'N' then nmiss=.;
  else if nmiss = . then nmiss = 0;
  if summary ne 'N' then zero=.;
  else if zero =. then zero = 0;
  format nmiss zero comma8.;
  if summary = 'A' then do;
    if sumnum > 0 then sumtext =
      'Not alpha:' ||
      left(put(sumnum,comma8.));
    else sumtext = '';
    end;
  if summary = 'F' then do;
    if sumnum > 0 then sumtext =
      'Not in list:' ||
      left(put(sumnum,comma8.));
    else sumtext = '';
    end;
  if summary = 'D' then
    sumtext = trim(left(put(min,date9.)))
      || '-' || left(put(max,date9.));
  mintext = right(put(min,best15.));
  maxtext = right(put(max,best15.));
  if crit = ‘Y’ and
    ((required=’Y’ and nmiss > 0) or
    flag=’Y’) then crit=’Y’;
  else crit = ‘N’;
  run;
proc sort data = ctent; by varname; run;

%IF &DEBUG=Y %THEN %DO;
  proc print data = ctent;
    format label $char30.;
    var varname label summary  nmiss zero
     mintext maxtext sumtext crit;
    TITLE2 "Contents of dataset &dset after
      matching with summarization dataset";
    run;
%END;
```

Outside of data step processing, other SAS procedures are utilized to obtain summary information for the input data set. PROC FREQ is used to build a text string containing the contents of unique variable types and PROC MEANS is used to build summarization strings of numeric variable types. Note that PROC PRINTTO is used here to obtain the results from these procedures. This will become an easier task when utilizing version 8.

```
  *** THESE OPTIONS ARE IMPORTANT FOR PROC
      PRINTTO;
  OPTIONS NOCENTER NODATE NOLABEL NONUMBER
    ps=2000;
  TITLE '';

*** Build the frequency table for variables
    with unique entries;
proc format;
  value
    $MISS '                  ' = 'MISSING';
  value  MISS . = 'MISSING';
  run;
data _null_;
  set unique end=last;
  FILE FREQTEXT old;
  length charlist numlist $80;
  retain charlist numlist;

  **** THE FIRST TIME FOR EACH TYPE OF
   VARIABLE IN THE DATASET, WRITE THE
   HEADER CODE TO A TEMPORARY FILE;
  IF _N_ = 1 THEN DO;
    PUT
   "PROC PRINTTO NEW PRINT=FREQLIST; RUN;";
    PUT "PROC FREQ DATA = raw;";
    put "tables ";
    END;

  put varname;
  if typevar = "Char" then
    charlist = trim(left(charlist)) || ' '
    || varname;
  else numlist = trim(left(numlist)) || ' '
    || varname;

  if last then do;
    put "/missing;";
    if charlist ne ' ' then
      put "format " charlist "$MISS.;";
    if numlist ne ' ' then
      put "format " numlist "MISS.;";
    put "run; proc printto; run;";
    end;
  run;
%include freqtext /nosource;
data fsumm (keep = varname missing unique);
  infile freqlist;
  length varname vartemp $8 flag $1
    unique $40;
  retain varname flag unique;
  retain missing 0;
  format missing comma9.;
  input vartemp $ 1-8 @;
  vartemp = left(vartemp);
  select (vartemp);
    when(' ') do;
      if flag = 'Y' then do;
        output;
        varname = ''; flag='N';
        missing=0; unique = ' ';
        end;
      else input;
      end;
    when('--------') do;
      flag = 'Y'; input; end;
    when('MISSING') input missing;
    otherwise do;
      if flag='Y' then do;
        input count;
        if unique = ' ' then
          unique =
          trim(left(vartemp)) || '(' ||
          trim(left(put(count,comma10.)))
          || ')';
        else unique = trim(unique) || ' '
          || trim(left(vartemp)) || '(' ||
          trim(left(put(count,comma10.)))
          || ')';
        end;
      else do;
        varname=scan(vartemp,1);
```

```
          input; end;
        end;
      end;
    run;
  proc sort data = fsumm; by varname; run;

%IF &DEBUG=Y %THEN %DO;
  TITLE2
  'Results of compiling unique data set';
  proc print data = fsumm; run;
%END;


  /****************************************
  If any of the variables in the analyzed
  Data set are of the type used in arithmetic
  equations, then the results of a proc means
  on these variables is read and summarized
  in various variables.  This information is
  then later added to the summary report.
  ***************************************/
  *** Build the proc means for numeric
      variables;
  data _null_;
    set number end=last;
    FILE MEANTEXT OLD;

    ** THE FIRST TIME FOR EACH TYPE OF
       VARIABLE IN THE DATASET, WRITE THE
       HEADER CODE TO A TEMPORARY FILE;
    IF _N_ = 1 THEN DO;
       PUT
     "PROC PRINTTO NEW PRINT=MEANLIST; RUN;";
       PUT
     "PROC MEANS DATA = raw NWAY NMISS MIN
      MAX MEAN STD;";
       put "  VAR ";
       END;

    put varname;

    if last then
      put "; run; proc printto; run;";
    run;
  %include meantext /nosource;

  data MINMAX
    (KEEP = varname missing textmin textmax
      textsum);
    INFILE MEANLIST MISSOVER;
    LENGTH FLAG $1 textsum $40
      textmin textmax $15 varname $8;
    RETAIN FLAG;
    INPUT @1 varname $8. @;
    IF varname = '--------' AND FLAG NE 'Y'
      THEN FLAG = 'Y';
    else IF varname = '--------'
      and FLAG = ‘Y’ THEN stop;
    ELSE IF varname = ‘Y’ THEN DO;
      input missing min max mean std;
      textsum = 'Mean: ' || put(mean,10.2) ||
      '  Stdev: ' || put(std,10.2);
      textmin = right(put(min,comma10.2));
      textmax = right(put(max,comma10.2));
      OUTPUT;
      END;
  PROC SORT DATA = MINMAX; BY varname; RUN;

  %IF &DEBUG=Y %THEN %DO;
    TITLE2
    'Results of compiling means data set';
```

```
  proc print data = minmax; run;
%END;


/****************************************
Put contents table together for summary
report
***************************************/
data ctent (drop=unique);
  merge ctent
    fsumm(in=f keep = varname unique);
  by varname;
  if f then sumtext = unique;
  run;
data ctent
  (drop=missing textmin textmax textsum);
  merge ctent minmax(in=m);
  by varname;
  if m then do;
    nmiss = missing;
    mintext = textmin;
    maxtext = textmax;
    sumtext = textsum;
    end;
  run;
```

REPORT RESULTS

A summary report is created listing all variables in the input file and the results of the various summarizations. Noted in addition to the field name and label are the number of missing values, the number of zero entries, minimum, and maximum for numeric fields, the range of values, and whether critical errors were present for that field. Also created could be a separate report just for records with critical errors or a separation of records which are duplicates according to key fields, but this involves removal of information from the input file. I will leave that up to the discretion of the programmer.

```
**** Print the summary report;
TITLE1
"Data set Summary Report for: &inset";
PROC SORT DATA = CTENT; BY start;
OPTIONS CENTER DATE LABEL ps =60
  pageno=1;
PROC PRINT DATA = CTENT NOOBS DOUBLE
  UNIFORM SPLIT='*';
  label varname = 'Name' label='Label'
        nmiss = '#*Missing'
        zero = '# =*Zero'
        crit = ‘Critical*Entries’
        mintext = 'Min'
        maxtext = 'Max'
        sumtext =
          'Summary*/ Out of Range';
  format label $char20. ;
  var varname label nmiss zero mintext
    maxtext sumtext crit;
  run;
  /***********************************
  If any of the variables in the
  analyzed data set are of the
  categorical type, then proc freq
  reports are generated.
  *************************************/
    title2
    'Frequency tables for categorical
     variables';
    *** Create a frequency report for
        variables with unique entries;
    data _null_;
      length person $1; retain person;
      set cat end=last;
      FILE FREQTEXT OLD;

      **** THE FIRST TIME FOR EACH TYPE
           OF VARIABLE IN THE DATASET,
           WRITE THE HEADER CODE TO A
           TEMPORARY FILE;
      IF _N_ = 1 THEN DO;
        PUT "PROC FREQ DATA = raw;";
        put "tables ";
        END;
      if frq = ' ' then put varname;
        else put frq;

      if last then do;
        put "/list missing; run;";
        end;
      run;
    %include freqtext /nosource;

  /************************************
  print flagged observations which did
  not meet allowable criteria.
  *************************************/
  proc sort data = errors nodupkey;
    BY &keystr; run;
  proc sort data = errors; by message;
  title2 'Example of flagged records';
  options ps=60;
  proc print data = errors noobs;
    BY message;
    run;
  *** clear temporary files;
  data _null_;
    file FMTTEXT old;
    file FREQTEXT old;
    file FREQLIST old;
    file MEANTEXT old;
    file MEANLIST old;
    run;
%mend;
```

CONCLUSIONS

Execution of this rather involved front-end process for incoming data files does not carry the burden of excessive processing time. This is due to making one pass of the input file through the processing logic and applying summary procedures only when necessary. The general summarization report produced by this macro can be extremely useful by itself in determining validity of the source file, but the report can also be easily customized through use of some simple added Base SAS code. As noted throughout, the field definition / logic table is the key to this process and provides much of the summarization logic without having to write separate code for each field. This macro could be a useful tool for those with a keen interest in quality control of source file information before further processing is to occur.

REFERENCES

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

Aster, Rick and Seidman, Rhena (1991), Professional SAS Programming Secrets, New York: McGraw-Hill, Inc.

ACKNOWLEDGEMENTS

SAS is a registered trademark or trademark of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.

CONTACT INFORMATION

Mike Tangedal
1204 Harmon Pl #19
Minneapolis MN, 55403
ph: 612-332-4235 e-mail: miamike@mtn.org