Paper 083-2010

SAS® Code Validation: L.E.T.O Method

Aaron Augustine, Information Resources, Inc., Chicago, IL

ABSTRACT

Coders write abundant amounts of SAS® code for ad hoc analysis. This code is not usually intended for use in a full scale production system, but it is still critical that it be validated. Specifically, the code should meet the necessary business requirements, be syntactically correct and have the output reviewed. What makes this challenging is coders must often do this without the benefit of external testing/peer review and usually in a short time frame. This paper presents a method and checklist for reviewing your work that will ensure each of these points are covered. It is easy to remember and implement. The content presented here is not specific to any particular SAS platform or version. It is intended for beginning and intermediate SAS coders.

INTRODUCTION

In an analytics world a lot of SAS code is written for ad hoc analysis. The challenge is to provide an adequate level of validation for a program within the time constraints of the project. If a person searches the web they would find several prior SAS Global Forum papers related to debugging, log review and validation. Each paper generally focuses on debugging or validation. This paper takes a different approach by covering both aspects. It describes a method to review programs for the correct logic, syntax, record counts and output that is applicable to any platform. The method is easy to remember and implement.

BACKGROUND

Searching the web for such SAS topics as Debugging, Validation, and Error handling a person will find approximately 15 different SAS Global Forum papers. Upon reviewing these, each paper generally would fall into two camps: Debugging/Log review and Code Validation. For Example, in SUGI 28, Lora D. Delwiche & Susan J. Slaughter gives a very practical guide for debugging SAS programs with ERRORS, WARNINGS, AND NOTES (OH MY) A Practical Guide To Debugging SAS Programs. This paper is an excellent source for details on common errors, but it could use more details on program validation. Also in SUGI 28, Neil Howard presented a paper on Beyond Debugging: Program Validation. The paper also gives a good overview of how to validate, but it may be challenging to implement depending on project timelines. In the end what most coders really need is a quick method to verify that their logic, syntax, record counts and output are correct. The L.E.T.O method accomplishes each of these objectives.

L.E.T.O Method

The L.E.T.O method stands for:

L: Logic: Review the code for the correct logic
E: ERROR: Review the log for Errors, Warning, Uninitialized, Etc.
T: Trace: Trace the record counts in the log
O: Output: Check the output file

Anyone working with SAS can use this method to quickly determine if the code they are working with is validated. To illustrate this method this paper provides a sample SAS code at the end along with the log and .LST files. The paper will reference this sample as it describes each step in the method. The amount of time a coder spends on each step will depend on the complexity of the code and what steps are executing. For example, checking for “repeats of BY values” is not necessary unless the code is merging files with the Data Step.

The sample code presented is a simple example that reads in a source file with and ID, FIPSCODE and CENTS field. It then merges up a translation file to calculate total dollars by County Name and write out the results. The presentation of the method starts below with L: LOGIC.

L: LOGIC

New coders sometimes have the tendency to rush programs focusing first on making sure there are no syntax errors. This is an important step but it is more important to make sure that the correct requirements and logic are coded before reviewing a log that may change anyway. Once a coder finishes a program, they should go back and separately list out the requirements of the code. Then they should trace back and make sure each point is covered. An even better approach would be to write out each functional step in the code as a comment. This way they will align to the requirements as they write the code. For example, in the sample code for this paper the brief high level requirements are listed at the top of the program as comments and at each appropriate step.

```
/*HIGH LEVEL CODE REQUIREMENTS AS COMMENTS*/
/*1. Read in sources file*/
/*2. Merge in translation file*/
/*3. Run calculations*/
/*4. Write out Results*/
```

E: ERROR

Some coders seem to focus more on just looking for ERROR, WARNING, and uninitialized messages. Examples are given in the sample code for each of these.

```
NOTE: Variable dummy is uninitialized.



88         proc freq data=sumout; tables _fre_; tile '_freq_ check';
                                                ____
                                                14
ERROR: Variable _FRE_ not found.

WARNING 14-169: Assuming the symbol TITLE was misspelled as tile.
```

These messages are a good place to start, but it is essential that coders extend their review of the log for other key messages such as LOST CARD, new line, truncated, repeats of BY values, missing and Invalid data. Ignoring these messages can be problematic. A more detailed discussion on these messages is given below. A coder should keep in mind that depending on the program editor/environment these items may or may not be case sensitive.

LOST CARD/new line/truncated/Invalid data

These SAS notes are often overlooked when reading in an external data file. However coders should double check and verify why the message occurred and resolve them accordingly. For the data source given, these messages could be acceptable, but in general it's safer to resolve these messages since they could be masking other issues in the input files. The sample code provides an example of each one of these messages. (See comments and code below.)

- When reading in the source.txt file the LOST CARD and new line notes are due to missing values and can be resolved by adding a MISSOVER option to the infile statement.
- The truncated message is caused by a LRECL option on the infile for the translation.txt file that is too short. It should be adjusted to 100.
- The Invalid Data is due to a text value in a field that is defined as numeric in source.txt. SAS will treat this as missing. For this example this is sufficient, but a coder may want to investigate their data source further depending on the extent of the Invalid data messages.

```
NOTE: The infile '/ahome/msaja/source.txt' is:
      Filename=/ahome/msaja/source.txt,
      Owner Name=msaja,Group Name=act_dev,
      Access Permission=rw-r--r--,
      Last Modified=Mon Oct 19 20:28:32 2009,

2                            The SAS System
                                        10:08 Tuesday, October 20, 2009

      File Size (bytes)=361

NOTE: Invalid data for cents in line 13 10-13.
RULE:     ----+---1----+---2----+---3----+---4----+---5----+---6
13        14 25027 text 13
id=14 fipscode=25027 cents=. _ERROR_=1 _N_=13
NOTE: LOST CARD.
```

```
id=28 fipscode=25027 cents=. _ERROR_=1 _N_=28
NOTE: 28 records were read from the infile '/ahome/msaja/source.txt'.
      The minimum record length was 9.
      The maximum record length was 13.
NOTE: SAS went to a new line when INPUT statement reached past the
      end of a line.

NOTE: The infile '/ahome/msaja/translation.txt' is:

3                            The SAS System
                                        10:08 Tuesday, October 20, 2009

      Filename=/ahome/msaja/translation.txt,
      Owner Name=msaja,Group Name=act_dev,
      Access Permission=rw-r--r--,
      Last Modified=Mon Oct 19 20:03:34 2009,
      File Size (bytes)=414

NOTE: 28 records were read from the infile
      '/ahome/msaja/translation.txt'.
      The minimum record length was 10.
      The maximum record length was 10.
      One or more lines were truncated.
```

In general truncated and invalid data messages are easier to resolve than Lost Card/New line. SUGI Paper 058-30, In Search of the LOST CARD, gives some good discussion on these topics.

Repeats of BY Values

This message occurs when the merge statement has more than one dataset with repeats of by values. If a coder encounters this message, they should first check the expected structures of the input files and determine if the repeat by values should exist, and then if needed resolve it accordingly. Often new coders might want to simply remove the duplicates rather than figuring out why they exist in the first place. Look below for an example of this message.

```
NOTE: MERGE statement has more than one data set with repeats of BY
      values.
```

In this particular example, there are duplicate records in the translation file that could be removed with a NODUPKEY option on the sort before the merge step. SUGI Paper 194-25, Pruning the SASLOG –Digging into the Roots of NOTEs, WARNINGs, and ERRORs, also describes the repeat values message and a way to identify why the message occurred.

Missing

Missing values are not uncommon and coders may have a tendency to overlook this message. This message however could be an indicator that (a) There is a problem with the dataset processing or (b) There is a data condition that has not been accounted for. Whatever the reason it would be a mistake to ignore the message without first determining if it is acceptable to have missing values or not. See example below.

```
24         data source;
25           set source;
26           dollars=cents/100; /*convert cents to dollars*/
27         run;

NOTE: Missing values were generated as a result of performing an
      operation on missing values.
      Each place is given by: (Number of times) at (Line):(Column).
      1 at 26:16
```

The source data file has missing values for CENTS for some records.

T: TRACE

Once a coder has reviewed the logic of their program and resolved all messages in the log, it is a good idea to trace the record counts of the log from start to end. Other documents and papers recommend printing out a few observations or using PUT statements to output data to verify logic, but the L.E.T.O. method suggests a coder take it one step further. As they read the log they should ask these questions:

1. What is the expected number of records for this input/output file? For example,

```
NOTE: The data set WORK.SOURCE has 27 observations and 3 variables.
NOTE: DATA statement used (Total process time):
```

   The source file should have had 28 records read in from the raw file rather than 27.

2. How many variables do I expect? For example,

```
67          /*3. Run calculations*/
68          proc summary data=aandb;
69            by county_name;

5                            The SAS System
                                        10:08 Tuesday, October 20, 2009

70            var dollars;
71            output out=sumout (drop= _type_) sum=;
72

NOTE: There were 28 observations read from the data set WORK.AANDB.
NOTE: The data set WORK.SUMOUT has 13 observations and 3 variables.
NOTE: PROCEDURE SUMMARY used (Total process time):
```

   From the summary step a coder would expect to see 3 variables: COUNTY_NAME, _FREQ_, and DOLLARS.

3. When files are merged how many records should be either file or in both? For example,

```
47          data aandb
48               anotb
49               bnota

4                            The SAS System
                                        10:08 Tuesday, October 20, 2009

50                 ;
51            merge source      (in=a)
52                  translation (in=b)
53                  ;
54            by fipscode;
55            testvar=dummy;
56            if a and b then output aandb;
57            else if a and not b then output anotb;
58            else if not a and b then output bnota;
59          run;

NOTE: Variable dummy is uninitialized.
NOTE: MERGE statement has more than one data set with repeats of BY
      values.
NOTE: There were 27 observations read from the data set WORK.SOURCE.
NOTE: There were 28 observations read from the data set
      WORK.TRANSLATION.
NOTE: The data set WORK.AANDB has 28 observations and 7 variables.
NOTE: The data set WORK.ANOTB has 0 observations and 7 variables.
NOTE: The data set WORK.BNOTA has 0 observations and 7 variables.
```

   In this case a coder would expect all the records in the source file to have a translation record. If there were records in ANOTB or BNOTA there could be a problem with the input files or code logic.

A coder comparing their expectations against what they observe in the log is a good way to see when they're short records/variables or have more than anticipated. Both situations often mean that the code logic and data is not lining up and some corrective action is required.

O: OUTPUT

The last step in the method is to check the output. This step applies to both the .LST file and any other output files produced by the program. Specifically, when a coder reviews the .LST and OUTPUT files they should check that:

(a) The output matches the format specified
(b) Look for any missing values
(c) Spot check any difficult calculations or formulas
(d) Output is consistent with general logic

The AANDB merged file in the sample code demonstrates issues (a) through (c).

- (a) The COUNTY_NAME is truncated due to no length statement when reading in the translation file
- (b) The TESTVAR is missing since DUMMY is uninitialized.
- (c) The DOLLARS calculation is correct.

```
                                  aandb                                 3
                                       10:08 Tuesday, October 20, 2009

                                                  county_
     Obs    id    fipscode    cents    dollars     name     testvar    dummy

       1     1      25001      100        1        Barn        .          .
       2    15      25001      100        1        Barn        .          .
       3     2      25003      200        2        Berk        .          .
       4    16      25003      200        2        Berk        .          .
       5     3      25005      300        3        Bris        .          .
```

The final output file sample1.txt demonstrates issue (d) specifically, the starting position for the _FREQ_ variable is incorrect and the COUNTY_NAME is truncated. The coder would also expect the _FREQ_ to be 2 for each variable based on the source file content but Hamp shows 4.

```
Barn             2        2.00
Berk             2        4.00
Bris             2        6.00
Duke             2        8.00
Esse             2       10.00
Fran             2       12.00
Hamp             4       30.00
Midd             2       18.00
Nant             2       20.00
Norf             2       22.00
Plym             2       24.00
Suff             2       26.00
Worc             2          .
```

In short, output checks are often overlooked but are essential since they are not obvious and can have a big impact on the results.

Check List

This section provides a checklist that the coder could use to implement this method. In general as the coder gains confidence with this method, these checks will become almost second nature.

1. Logic: Check the code logic against requirements
2. Error: Review the logic for the following items and resolve accordingly:
    a. ERROR, WARNING, uninitialized,
    b. LOST CARD, new line, truncated, Invalid data
    c. repeats of BY values
    d. missing
3. Trace: Trace the record counts of the log
    a. Expected number of records and variables
    b. Record counts for datasets resulting from merges (A and B, A not B, B not A)
    c. Expected number of duplicates from a PROC SORT NODUPKEY
4. Output: Check the .LST and output files for
    a. The output matches the format specified (variables/formats)
    b. Look for any missing values.
    c. Spot check any difficult calculations or formulas
    d. Output is consistent with general logic.

Conclusion

Full scale production SAS code should have all the necessary review and testing plans executed before implementation. For ad hoc SAS code, coders often do not have the benefit of external testing or peer review and are usually working in short time periods. The L.E.T.O method provides a checklist to coders to verify their work in a way that is effective and quick to implement. It can help verify the code meets the necessary business requirements, is syntactically correct and has the output validated.

References

Andrew T. Kuligowski, In Search of the LOST CARD, Thirtieth Annual SAS Users Group International Conference, Philadelphia, PA

Andrew T. Kuligowski, Pruning the SASLOG –Digging into the Roots of NOTEs, WARNINGs, and ERRORs, Twenty Fifth Annual SAS Users Group International Conference, Indianapolis, IN

Lora D. Delwiche & Susan J. Slaughter, ERRORS, WARNINGS, AND NOTES (OH MY) A Practical Guide To Debugging SAS Programs, Proceedings of the Twenty Eighth Annual SAS Users Group International Conference, Seattle, WA

Neil Howard, Beyond Debugging: Program Validation, Proceedings of the Twenty Eighth Annual SAS Users Group International Conference, Seattle, WA

Contact Information

Comments and questions are valued and encouraged. Contact the author at:

Aaron Augustine
Director, Analytics Research & Development
Information Resources, Inc.
Tel: +1 312 474 2159
aaron.augustine@infores.com or augustine17@sbcglobal.net
www.infores.com

Sample Code Source Files

source.txt	translation.txt
1 25001 100	25001 Barnstable
2 25003 200	25003 Berkshire
3 25005 300	25005 Bristol
4 25007 400	25007 Dukes
5 25009 500	25009 Essex
6 25011 600	25011 Franklin
7 25013 700	25013 Hampden
8 25015 800	25015 Hampshire
9 25017 900	25017 Middlesex
10 25019 1000	25019 Nantucket
11 25021 1100	25021 Norfolk
12 25023 1200	25023 Plymouth
14 25027 text	25025 Suffolk
13 25025 1300	25027 Worcester
15 25001 100	25001 Barnstable
16 25003 200	25003 Berkshire
17 25005 300	25005 Bristol
18 25007 400	25007 Dukes
19 25009 500	25009 Essex
20 25011 600	25011 Franklin
21 25013 700	25013 Hampden
22 25015 800	25015 Hampshire
23 25017 900	25017 Middlesex
24 25019 1000	25019 Nantucket
25 25021 1100	25021 Norfolk
26 25023 1200	25023 Plymouth
27 25025 1300	25025 Suffolk
28 25027	25027 Worcester

Code

NOTE: This source code is provided for the purpose of illustrating the points made in this paper. Readers should be encouraged to evaluate and test this source code thoroughly, before deciding to use it in their own SAS programs.

```
/*CODE EXAMPLE FOR L.E.T.O METHOD*/
/*20091018*/
/*Aaron Augustine*/


/*HIGH LEVEL CODE REQUIREMENTS AS
COMMENTS*/
/*1. Read in sources file*/
/*2. Merge in translation file*/
/*3. Run calculations*/
/*4. Write out Results*/

options ls=70;

/*1. Read in sources file*/
data source;
  infile '/ahome/msaja/source.txt';
  input
    id
    fipscode
    cents
    ;
run;
```

```
data source;
  set source;
  dollars=cents/100; /*convert cents
to dollars*/
run;

proc print data=source(obs=5); title
'source file'; run;


data translation;
  infile
'/ahome/msaja/translation.txt'
lrecl=10;
  input
    fipscode
    county_name $
    ;
run;
proc print data=translation(obs=5);
title 'translation file'; run;



/*2. Merge in translation file*/
proc sort data=source;      by
fipscode; run;
proc sort data=translation; by
fipscode; run;

data aandb
```

```
      anotb
      bnota
      ;
  merge source      (in=a)
        translation (in=b)
        ;
  by fipscode;
  testvar=dummy;
  if a and b then output aandb;
  else if a and not b then output
anotb;
  else if not a and b then output
bnota;
run;

proc print data=aandb(obs=5); title
'aandb'; run;
proc print data=anotb(obs=5); title
'anotb'; run;
proc print data=bnota(obs=5); title
'bnota'; run;



/*3. Run calculations*/
proc summary data=aandb;
  by county_name;
  var dollars;
  output out=sumout (drop= _type_)
sum=;

proc print data=sumout; title
'sumout file'; run;


/*4. Write out Results*/
filename sumout
'/ahome/msaja/sample1.txt';
data _null_;
  set sumout;
  file sumout;
  put
    @1  county_name $10.
    @9 _freq_       8.
    @20 dollars     8.2
    ;
run;

proc freq data=sumout; tables _fre_;
tile '_freq_ check'; run;

/*End of program*/
endsas;
```

Log

```
1           /*CODE EXAMPLE FOR
L.E.T.O METHOD*/
2           /*20091018*/
3           /*Aaron Augustine*/
4
5
6           /*HIGH LEVEL CODE
REQUIREMENTS AS COMMENTS*/
7           /*1. Read in sources
file*/
8           /*2. Merge in translation
file*/
9           /*3. Run calculations*/
10          /*4. Write out Results*/
11
12          options ls=70;
13
14          /*1. Read in sources
file*/
15          data source;
16            infile
'/ahome/msaja/source.txt';
17            input
18              id
19              fipscode
20              cents
21              ;
22          run;

NOTE: The infile
'/ahome/msaja/source.txt' is:

Filename=/ahome/msaja/source.txt,
      Owner Name=msaja,Group
Name=act_dev,
      Access Permission=rw-r--r--,
      Last Modified=Mon Oct 19
20:28:32 2009,

2                            The SAS
System

10:08 Tuesday, October 20, 2009

      File Size (bytes)=361

NOTE: Invalid data for cents in line
13 10-13.
RULE:     ----+---1----+---2----+---
3----+---4----+---5----+---6
13        14 25027 text 13
id=14 fipscode=25027 cents=.
_ERROR_=1 _N_=13
NOTE: LOST CARD.
id=28 fipscode=25027 cents=.
_ERROR_=1 _N_=28
NOTE: 28 records were read from the
infile '/ahome/msaja/source.txt'.
      The minimum record length was
9.
      The maximum record length was
13.
NOTE: SAS went to a new line when
INPUT statement reached past the
      end of a line.
NOTE: The data set WORK.SOURCE has
27 observations and 3 variables.
NOTE: DATA statement used (Total
process time):
      real time           0.02
seconds
      cpu time            0.02
seconds


23
24          data source;
25            set source;
26            dollars=cents/100;
/*convert cents to dollars*/
27          run;

NOTE: Missing values were generated
as a result of performing an
      operation on missing values.
      Each place is given by:
(Number of times) at
(Line):(Column).
      1 at 26:16
NOTE: There were 27 observations
read from the data set WORK.SOURCE.
NOTE: The data set WORK.SOURCE has
27 observations and 4 variables.
```

```
NOTE: DATA statement used (Total
process time):
      real time           0.01
seconds
      cpu time            0.00
seconds


28
29          proc print
data=source(obs=5); title 'source
file'; run;

NOTE: There were 5 observations read
from the data set WORK.SOURCE.
NOTE: The PROCEDURE PRINT printed
page 1.
NOTE: PROCEDURE PRINT used (Total
process time):
      real time           0.09
seconds
      cpu time            0.05
seconds


30
31
32          data translation;
33            infile
'/ahome/msaja/translation.txt'
lrecl=10;
34            input
35              fipscode
36              county_name $
37              ;
38          run;

NOTE: The infile
'/ahome/msaja/translation.txt' is:

3                            The SAS
System

10:08 Tuesday, October 20, 2009


Filename=/ahome/msaja/translation.tx
t,
      Owner Name=msaja,Group
Name=act_dev,
      Access Permission=rw-r--r--,
      Last Modified=Mon Oct 19
20:03:34 2009,
      File Size (bytes)=414

NOTE: 28 records were read from the
infile

'/ahome/msaja/translation.txt'.
      The minimum record length was
10.
      The maximum record length was
10.
      One or more lines were
truncated.
NOTE: The data set WORK.TRANSLATION
has 28 observations and 2
      variables.
NOTE: DATA statement used (Total
process time):
      real time           0.01
seconds
      cpu time            0.01
seconds
```

```
39          proc print
data=translation(obs=5); title
'translation
39        ! file'; run;

NOTE: There were 5 observations read
from the data set
      WORK.TRANSLATION.
NOTE: The PROCEDURE PRINT printed
page 2.
NOTE: PROCEDURE PRINT used (Total
process time):
      real time           0.00
seconds
      cpu time            0.00
seconds


40
41
42
43          /*2. Merge in translation
file*/
44          proc sort data=source;
by fipscode; run;

NOTE: There were 27 observations
read from the data set WORK.SOURCE.
NOTE: The data set WORK.SOURCE has
27 observations and 4 variables.
NOTE: PROCEDURE SORT used (Total
process time):
      real time           0.00
seconds
      cpu time            0.00
seconds


45          proc sort
data=translation; by fipscode; run;

NOTE: There were 28 observations
read from the data set
      WORK.TRANSLATION.
NOTE: The data set WORK.TRANSLATION
has 28 observations and 2
      variables.
NOTE: PROCEDURE SORT used (Total
process time):
      real time           0.00
seconds
      cpu time            0.00
seconds


46
47          data aandb
48               anotb
49               bnota

4                            The SAS
System

10:08 Tuesday, October 20, 2009

50               ;
51            merge source
(in=a)
52                  translation
(in=b)
53                  ;
54            by fipscode;
55            testvar=dummy;
```

```
56            if a and b then output
aandb;
57            else if a and not b
then output anotb;
58            else if not a and b
then output bnota;
59          run;

NOTE: Variable dummy is
uninitialized.
NOTE: MERGE statement has more than
one data set with repeats of BY
      values.
NOTE: There were 27 observations
read from the data set WORK.SOURCE.
NOTE: There were 28 observations
read from the data set
      WORK.TRANSLATION.
NOTE: The data set WORK.AANDB has 28
observations and 7 variables.
NOTE: The data set WORK.ANOTB has 0
observations and 7 variables.
NOTE: The data set WORK.BNOTA has 0
observations and 7 variables.
NOTE: DATA statement used (Total
process time):
      real time             0.01
seconds
      cpu time              0.02
seconds


60
61          proc print
data=aandb(obs=5); title 'aandb';
run;

NOTE: There were 5 observations read
from the data set WORK.AANDB.
NOTE: The PROCEDURE PRINT printed
page 3.
NOTE: PROCEDURE PRINT used (Total
process time):
      real time             0.00
seconds
      cpu time              0.00
seconds


62          proc print
data=anotb(obs=5); title 'anotb';
run;

NOTE: No observations in data set
WORK.ANOTB.
NOTE: PROCEDURE PRINT used (Total
process time):
      real time             0.00
seconds
      cpu time              0.00
seconds


63          proc print
data=bnota(obs=5); title 'bnota';
run;

NOTE: No observations in data set
WORK.BNOTA.
NOTE: PROCEDURE PRINT used (Total
process time):
      real time             0.00
seconds
      cpu time              0.00
seconds
```

```
64
65
66
67          /*3. Run calculations*/
68          proc summary data=aandb;
69            by county_name;

5                            The SAS
System

10:08 Tuesday, October 20, 2009

70            var dollars;
71            output out=sumout
(drop= _type_) sum=;
72

NOTE: There were 28 observations
read from the data set WORK.AANDB.
NOTE: The data set WORK.SUMOUT has
13 observations and 3 variables.
NOTE: PROCEDURE SUMMARY used (Total
process time):
      real time             0.01
seconds
      cpu time              0.02
seconds


73          proc print data=sumout;
title 'sumout file'; run;

NOTE: There were 13 observations
read from the data set WORK.SUMOUT.
NOTE: The PROCEDURE PRINT printed
page 4.
NOTE: PROCEDURE PRINT used (Total
process time):
      real time             0.00
seconds
      cpu time              0.00
seconds


74
75
76          /*4. Write out Results*/
77          filename sumout
'/ahome/msaja/sample1.txt';
78          data _null_;
79            set sumout;
80            file sumout;
81            put
82              @1  county_name $10.
83              @9  _freq_        8.
84              @20 dollars      8.2
85              ;
86          run;

NOTE: The file SUMOUT is:

Filename=/ahome/msaja/sample1.txt,
      Owner Name=msaja,Group
Name=act_dev,
      Access Permission=rw-r--r--,
      Last Modified=Tue Oct 20
10:08:33 2009

NOTE: 13 records were written to the
file SUMOUT.
      The minimum record length was
27.
```

```
      The maximum record length was
27.
NOTE: There were 13 observations
read from the data set WORK.SUMOUT.
NOTE: DATA statement used (Total
process time):
      real time           0.01
seconds
      cpu time            0.00
seconds


87
88           proc freq data=sumout;
tables _fre_; tile '_freq_ check';

____

14
ERROR: Variable _FRE_ not found.

WARNING 14-169: Assuming the symbol
TITLE was misspelled as tile.

6                             The SAS
System

10:08 Tuesday, October 20, 2009
```

```
NOTE: The SAS System stopped
processing this step because of
errors.
NOTE: PROCEDURE FREQ used (Total
process time):
      real time           0.00
seconds
      cpu time            0.00
seconds

89
90           /*End of program*/
91           endsas;


ERROR: Errors printed on page 5.

NOTE: SAS Institute Inc., SAS Campus
Drive, Cary, NC USA 27513-2414
NOTE: The SAS System used:
      real time           0.44
seconds
      cpu time            0.23
seconds
```

LST file

```
                          source file                          1
                                    10:08 Tuesday, October 20, 2009

           Obs    id    fipscode    cents    dollars

             1     1     25001       100        1
             2     2     25003       200        2
             3     3     25005       300        3
             4     4     25007       400        4
             5     5     25009       500        5

                        translation file                      2
                                    10:08 Tuesday, October 20, 2009

                                   county_
                   Obs    fipscode    name

                     1      25001     Barn
                     2      25003     Berk
                     3      25005     Bris
                     4      25007     Duke
                     5      25009     Esse

                             aandb                             3
                                    10:08 Tuesday, October 20, 2009

                                   county_
Obs   id   fipscode   cents   dollars    name     testvar   dummy

  1    1    25001      100       1       Barn        .        .
  2   15    25001      100       1       Barn        .        .
  3    2    25003      200       2       Berk        .        .
  4   16    25003      200       2       Berk        .        .
  5    3    25005      300       3       Bris        .        .

                          sumout file                          4
                                    10:08 Tuesday, October 20, 2009

                       county_
              Obs       name      _FREQ_    dollars

                1       Barn         2          2
                2       Berk         2          4
                3       Bris         2          6
  4       Duke          2          8
  5       Esse          2         10
  6       Fran          2         12
  7       Hamp          4         30
  8       Midd          2         18
  9       Nant          2         20
 10       Norf          2         22
 11       Plym          2         24
 12       Suff          2         26
 13       Worc          2          .
```