Paper 203-25

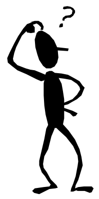
Garbage In, Garbage Out. Does It Have To Be That Way?

Louise Hadden
Abt Associates Inc., Cambridge, MA

ABSTRACT

As computer programming professionals, we are frequently presented with raw data that is, shall we say, less than ideal. This paper illustrates various SAS programming techniques for dealing with two separate "difficult" external files and a group of externally produced SAS data sets. These techniques, unless specifically noted, can be used with SAS under any operating system. In the process of completing the programming tasks described below, SAS version 6.12 for Windows 95/NT and SAS version 6.12 for AIX/UNIX were used.

INTRODUCTION

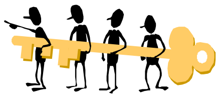


Three separate examples of files which presented a challenge to correctly read into SAS, or process using SAS, are described. Each of these files were delivered to Abt Associates Inc., a social science research company, by several different clients who desired that we use the data in performance of contracted research projects in the area of housing and health economics.

MUNICIPAL RECYCLING

The first 'difficult' file was a flat ASCII file output by a custom-designed software system used by a municipal tax assessor's office. The data contained multiple record types, and within record types, 'corrected' records and original records. Record types, and corrected vs. original records, had a different layout with different fields.

The first task for any SAS programmer presented with such a file is to research infile and input options available in the SAS documentation. The second task is to obtain documentation for the file in question. Some files will be clearly and completely documented, while others will not. In this case, no documentation was originally provided with the file. A request was made for documentation, which, when it arrived, was incomplete, containing layouts for original record types, but no way of determining record type, and no layouts for corrected records. The third task, whether or not a programmer receives a well-documented file, is to 'look at' the data. In this case, the file, while large for a PC, could be viewed using the LIST utility. An ASCII file can be similarly viewed in AIX UNIX systems with the 'pg' command. For EBCDIC files or mainframe files, different techniques must be used to view the file.

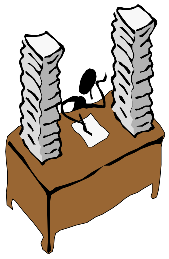


One method is to read the file in using SAS with several long character strings and review the output. I used a character variable length of 100 for simplicity's sake--a length of 100 fits on a single line printed landscape and it makes writing the input statement easy. I use a record length that I know is as long or longer than the longest input record with the infile options missover and pad.

```
options obs=100;

data temp;
infile xx lrecl=600 missover pad;
length string1-string6 $ 100;
input @1    string1 $ebcdic100.
      @101  string2 $ebcdic100.
      @201  string3 $ebcdic100.
      @301  string4 $ebcdic100.
      @401  string5 $ebcdic100.
      @501  string6 $ebcdic100.;
run;
```

Another option is to do a tape dump and/or tape analysis should the file be on tape or cartridge.



After painstakingly reviewing the 'flat' data file and the documentation available, it was possible to determine how to differentiate the different record types, and within each record type, how to differentiate an normal record from an 'edit' record. Edit records contained an offset which turned out to be the number of variables in a particular record type, which was discovered by reviewing printed edit records using the technique described above.

```
%macro runit(key,num);

filename yy&num ".\camb&key..dat";

data _null_;
infile xx lrecl=715 missover pad;
length glop1-glop6 $ 100;
file yy&num lrecl=736 pad;
input
                @1 blockno $5.
                @6 lotno $5.
                @11 unitno $5.
                @16 saleyear $4.
                @20 reckey $3.
                @;
if reckey ne "&key." then delete;
                input
                @24 vhyear $2.
                @26 fill1 $6.
                @33 editdate $6.
                @39 fill2 $1.
                @40 check $1. @;
                if check='*' then
input
                @41 glop1 $100.
                @141 glop2 $100.
                @241 glop3 $100.
                @341 glop4 $100.
                @441 glop5 $100.
                @541 glop6 $100.;

if check ne '*' and reckey eq '000'
then input
                @136 glop1 $100.
                @236 glop2 $100.
                @336 glop3 $100.
                @436 glop4 $100.
                @536 glop5 $100.
                @636 glop6 $100.;

if check ne '*' and reckey eq '040'
then input
                @97 glop1 $100.
                @197 glop2 $100.
                @297 glop3 $100.
                @397 glop4 $100.
                @497 glop5 $100.
                @597 glop6 $100.;

if check ne '*' and reckey eq '050'
then input
                @70 glop1 $100.
                @170 glop2 $100.
                @270 glop3 $100.
                @370 glop4 $100.
                @470 glop5 $100.
                @570 glop6 $100.;
```

```
glop1=trim(compress(glop1,",'.$"));
glop2=trim(compress(glop2,",'.$"));
glop3=trim(compress(glop3,",'.$"));
glop4=trim(compress(glop4,",'.$"));
glop5=trim(compress(glop5,",'.$"));
glop6=trim(compress(glop6,",'.$"));

star='*';
if check='*' then check='Y';
put reckey +(-1) star +(-1) check
+(-1) star +(-1) blockno +(-1) star
+(-1)
lotno +(-1) star +(-1) unitno +(-1)
star +(-1) saleyear +(-1)
star +(-1) vhyear +(-1) star +(-1)
fill1 +(-1) star +(-1) editdate +(-
1)
star +(-1) fill2 +(-1) star +(-1)
glop1 +(-1) glop2 +(-1) glop3 +(-1)
glop4 +(-1) glop5 +(-1) glop6;

run;

%mend;

%runit(000,1);
%runit(030,2);
%runit(040,3);
%runit(050,4);

endsas;
```

The macro above uses the following SAS tools to read in the raw data file and convert it to separate raw data files based on the record type. The first tool used is the trailing @ in the input statement, which allows you to select records of a particular type, and continue inputting data based on the values of variables already input. In this case, different record types and within record types, error vs. regular records, had different starting points following the header portion of the record and had to be input separately. The second tool used is the compress function, which rids the 100 column character strings of unwanted characters. In this case, part of each record was delimited with the '*' character, so all other delimiters were removed and the '*' character was inserted to delimit the parts of the records that had no delimiters. The third tool is to use the '+' in the put statement to move the pointer so that extra blanks were not inserted between the 100 character strings. Note that to move the pointer backwards a minus 1 (or however many spaces) must be enclosed in parentheses as the + sign indicates pointer movement, while the - sign by itself does not. The fourth tool used is to output a flat file(s) while reading in a flat file in the same data step. SAS also allows you to modify a file in place by using SHAREBUFFERS, described below.

"The SHAREBUFFERS|SHAREBUFS infile specifies that the FILE statement and the INFILE statements the same buffer. When using the INFILE, FILE, and PUT statements to update an external file in place, the SHAREBUFFERS option saves CPU time by preventing what is being written with the PUT statement from being copied to an output buffer. Instead, the PUT statement output is written straight from the INPUT buffer. SHAREBUFFERS is useful when you want to update specific fields, not an entire record." (SAS Online Documentation).

This technique was not used in this case because of the need to update entire records, and the desire to maintain a well-defined record of changes to the file.

Following the use of this macro, the resulting flat files were then possible to input in a 'normal' manner using the '*' delimited records and appropriate length statements for the variables in each file. The SAS data sets were processed and analyzed for the client, who received a Microsoft Excel spreadsheet and SAS reports as a final product. Without using the various infile and file tools that SAS provides, making sense of the original data set would not have been possible.

UNLIMITED DELIMITERS

Frequently SAS programmers are presented with flat files produced by such packages as Microsoft Access, Microsoft Excel, Lotus 1-2-3, and other data base, spreadsheet and/or word processing packages. Difficulties may arise with these files when they are presented as delimited files, as each package has their own standard way of delimiting fields. Since SAS programmers deal with so many different file types, SAS has the capactiy to recognize many different delimiters. Problems may arise when text fields include characters that SAS recognizes as delimiters. Many programmers are familiar with the use of the ampersand ("&") to indicate that a text field may have embedded blanks. The "&" allows programmers to specify that the field is not complete until the pointer reaches two consecutive blanks or the end of the line, whichever comes first. The usage of the "&" format modifier follows below. Note that two blanks follow the name field.

```
   data one;
      input name & $40. Age;
      cards;
      first1 mi1 last1  14
      firstfirst2 mi2 lastlast2  38
      first3 mimi3 last3  21
      ;
    run;

    proc print data=one;
      var name age;
    title 'Test Use of & in Input
Statement';
    run;
```

```
  RESULT:

   OBS  NAME                              AGE
     1 first1 mi1 last1                    14
     2 firstfirst2 mi2 lastlast2           38
     3 first3 mimi3 last3                  21
```

The SAS Institute has also provided another infile option which helps deal with both embedded blanks and other embedded delimiters. The DSD option is documented in the online documentation and in some "Changes and Enhancements" manuals, but not in the SAS Language Version 6 Manual.

"DSD changes the way delimiters are treated when using list input and enables you to read delimiters as characters within quoted strings. When the DSD option is in effect, the delimiter is assumed to be a comma. If the data contain another delimiter, you must specify it with the DELIMITER= option. DELIMITER|DLM= delimiters specifies a delimiter other than a blank (the default) for list input. Delimiters can be expressed as a list of delimiting characters or as a character variable. " (SAS Online Documentation)

The DSD option is invaluable when processing large text files which include fields such as a physician's name (i.e. John Jones, M.D.) Without the DSD infile option, this text field would be read as John Jones, and the subsequent field in the input statement would be incorrectly read as M.D. if the field were defined as character, and a missing value if the field were defined as numeric. Below follows an example of the use of the DSD option to convert a text claims file into a SAS data set. Note that as always with delimited files, it is important to include a length statement defining the length of character variables over 8 columns long. Because the records were variable length, with the longest record 300 columns, the LRECL and MISSOVER options were used as a precaution. Unless you are sure that the default logical record length on your host system is longer than the record length of the file, it is wise to specify both an LRECL which is at least as long as the longest record, and MISSOVER.

```
data dd.clms9699;
  length last first $ 12 manumber
  claimno provid cdob cthrudt
  cfromdt cclmrcdt $ 10
  provname dx1desc px1desc posdesc
  $ 50;

  infile xx delimiter=',' dsd
  missover lrecl=300;

  input claimno $ manumber $ last $
  first $ gender $ cdob $
  provid $ provname & $ dx1 $ dx2 $
  dx3 $ dx4 $ dx1desc & $
  px1 $ px1desc & $ poscode $
  posdesc & $ cfromdt $ cthrudt $
  ctotchg $ camtpaid $
  cclmrcdt $;

run;
```

In this case, the DSD option allowed several descriptive variables (provider name, primary diagnosis description, procedure description, and place of service description), which contained embedded ","s, to be correctly read.

Even with the DSD infile option, it is important to carefully reviewing the resulting variables. In this case, 99.9% of the records were read in correctly, but character fields with a single embedded double quote were not (SAS correctly identified the double quotes surrounding text fields as characters to ignore, but the single embedded double quote caused problems).

It turned out to be a quoted brand name beginning a field, which was possible to correct by preprocessing the file using the process described above and the TRANWRD function. TRANWRD(source,target,text) replaces or removes all occurrences of a word. The TRANSLATE, COMPRESS and DEQUOTE functions were not appropriate in this case, because they would remove the double quotes in the records that were valid because they were surrounding character fields.

MORE FUN WITH FUNCTIONS

The two examples above describe some techniques for dealing with some types of external files to the SAS system. There are other instances when SAS data files, libraries or catalogs may be problematic. In the example that follows, the task at hand was to link several SAS data sets containing names, various identifiers, and addresses in various formats. The purpose of the file linking was to identify households among a certain population. Functions were used in conjunction with reviewing of output to standardize the variables so that a match was possible.

In the abbreviated section of code below, a file containing SSI data was processed to extract separate city, state, street address and zip code variables from a single field, FULLADDR. Once again the length statement is important because SAS will give the length of the full variable to each of the variables parts if not otherwise specified. A listing of the full address field was reviewed before writing and running the program.

```
data dd.ssidata3;
    length city1 $ 20 state1 $ 2
    revzip zip1 $ 5 address1 $ 50
    ssn $ 11;
    set dd.ssidata2;

/* standardize names & addresses */

    first=upcase(fname);
    last=upcase(lname);

    fulladdr=upcase(fulladdr);
    address8=substr(fulladdr,1,8);

    if index(fulladdr,"WASH")>0
    then do;
        city1="WASHINGTON";
        state1="DC";
    end;

      . . .

    if index(fulladdr,"ARLI")>0
    then do;
        city1="ARLINGTON";
        state1="VA";
    end;


    revaddr=reverse(fulladdr);
    revzip=scan(revaddr,1);
    zip1=reverse(revzip);

    if substr(zip1,1,3)='200' then
    do;
        city1="WASHINGTON";
        state1="DC";
    end;

citystrt=index(fulladdr,"WAS")-1;
       . . .

address1=substr(fulladdr,1,citystrt
);
drop citystrt;
address1=upcase(address1);
address1=compress(address1,'-.,');
address8=substr(left(address1),1,8)
;
address8=upcase(address8);
city1=upcase(city1);
state1=upcase(state1);

run;
```

The first function used is the UPCASE function, which puts all characters in upper case. Since the SAS data sets provided were inconsistent, a decision was made to retain all character fields as upper case for the purposes of linking. The second function used was the SUBSTR function. Its purpose was to provide a shortened version of the street address field to identify possible matches. The third function used was the INDEX function. The INDEX function returns a number indicating the starting column of a specified character string (as opposed to INDEXC which identifies the column of the first instance of a specified character). The INDEX function was used in two ways, first to identify cities (a discreet number in the greater Washington DC area) and secondly to identify the end of the street address field. The next function used in this example was the REVERSE function, which produces a mirror image of a character field. First, the entire address field was reversed, and the first 'word' (in this case a reversed version of ZIP Code) was extracted using the SCAN function. The resulting 'word' was reversed again to form ZIP Code. All resulting variables from the extraction process were standardized using the UPCASE, LEFT and COMPRESS functions, to convert to upper case, remove any starting blanks, and remove any unwanted punctuation from the street address. Similar routines were performed on all the SAS data sets to be linked based on which variables were present in the files.

```
data enrms112;
    set enrms112;

ssn=compress(ssn1,'-');

first=upcase(first);
last=upcase(last);
mi=upcase(mi);

/* standardize addresses */

address1=upcase(address1);
address1=compress(address1,'-.,');
address8=substr(left(address1),1,8)
;
city1=upcase(city1);
state1=upcase(state1);
address8=upcase(address8);

run;

      . . .


data prosp112 (keep=address1
address8 gender link manumber city1
state1 zip1 first last mi pseq11
ssn);
    length address1 $ 50;
    set prosp112;

    first=upcase(first);
    last=upcase(last);
    mi=upcase(mi);

/* standardize addresses */

    address1=trim(house)||'
'||trim(street)||' '||trim(apt);
    address1=upcase(address1);
    address1=compress(address1,'-
      .,');
    address8=upcase(address8);
    city1=upcase(city);
    state1=upcase(state);
    zip1=upcase(zip);

run;
```

When creating an address field (or any other field containing more than one character variable) it is important to remember the TRIM function, which removes any extra trailing blanks from a variable. Then the number of blanks between 'words' in a character string can be controlled using the concatenation operator. However, if you mistakenly create or are given a variable with extra blanks between 'words', the function COMPBL will remove the extra blanks leaving only one blank between words. To remove ALL blanks between 'words', the COMPRESS function can be used.

Once all the input files had been standardized, they were run through an iterative linking routine which identified households based on such fields as street address, phone numbers, city, state, zip, adult and child last names.

CONCLUSION

The SAS system provides many useful tools and interfaces for reading 'difficult' external files. It also provides tools for extracting desired information from SAS data sets which are not set up in consistent ways. These tools make it possible to prevent "garbage" data from becoming "garbage" results, whether the result is a new SAS data set or analysis of existing SAS data sets.

REFERENCES

SAS Language, Version 6, First Edition
SAS Companion for the UNIX environment: Language, Version 6
SAS Online Documentation (PC SAS V612, AIX UNIX SAS V612)

ACKNOWLEDGMENTS

The author wishes to acknowledge the author of the LIST utility, Vernon D. Buerg.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Louise Hadden
Abt Associates Inc.
55 Wheeler St.
Cambridge, MA 02138
Work Phone: 617-349-2385
Fax: 617-349-2675

Email: louise_hadden@abtassoc.com

KEYWORDS

SAS, UNIX, INFILE, INPUT, PUT, POINTER CONTROL, LENGTH, MISSOVER, PAD,TRAILING @, SHAREBUFFERS, &, DSD, DELIMITER, LRECL, TRANWRD, TRANSLATE, COMPRESS, DEQUOTE, UPCASE, SUBSTR, INDEX, INDEXC, REVERSE, SCAN,TRIM, COMPBL, LIST, PG