PharmaSUG2011 - Paper CC13

Beyond the Comma Delimited File for Bringing Data into a SAS Dataset

David Franklin, TheProgrammmersCabin.com, Litchfield, NH

ABSTRACT

"Help! I got this AE data in a Word document and I need it in a SAS® dataset within the hour!"

Getting data from outside SAS is an important task for many who deal with data from different sources. There are a number of procedures that SAS have developed over the years that can get data from sources like Microsoft Excel and Access with PROC IMPORT, but still there is that time when we get data in a flat text file that is not suitable for this approach.

Based on a real-world example of an Adverse Event listing in a text file, this paper introduces several ways the INFILE and INPUT statements inside a datastep, in which SAS can still bring in your data from outside source to a dataset, even though it is not in one of the more commonly recognized electronic formats.

INTRODUCTION

"Help! I got this AE data in a Word document and I need it in a SAS dataset within the hour!" That was the call I received from one of the managers. The CRO just sent a listing in a Word document and had gone home, and AE data was needed in a dataset format before the end of the day.

The usual methods of importing data using PROC IMPORT or similar were not suitable so something quick had to be constructed. Fortunately the listing was in a good format so that it could be saved as a text file, and then this imported into a SAS dataset.

This paper presents several formats that the INFILE and INPUT statements can be used to import the text into a SAS dataset, then uses a couple of examples based on real-life situations to get the data into a SAS dataset.

BASIC WAYS OF USING INFILE/INPUT TO GET DIFFERENT TYPES OF TEXT INTO A SAS DATASET

SAS provides a number of ways for a user to input data using either a INFILE, CARDS or DATALINES statement. For the purposes of this paper, two things must be noted:

- the CARDS statement is exactly the same as the DATALINES statement, and is used instead of the INFILE statement, and
- the CARDS statement is used instead of a filename specified in the INFILE statement for ease of documenting the code used in the paper.

It must be noted that in each example the following statement was used to define the CITY and DISTANCE variables:

```
length city $15 distance 8;
```

The first and most commonly used input style is called the List Input, where the INPUT statement defines the name and order of the variable, and if the variable is character then a '$' sign appears after the name of that variable:

```
input city $ distance;
cards;
Amsterdam 370
Montreal 5200
Auckland 22720
;
```

Note that each value is separated by one or more spaces, and that in this style an INFILE statement is not needed -- it is implied from the CARDS statement. This style is best know for its simplicity.

The Column Input style can trace its origins back to the Punch Card where fixed positions in the column were defined to hold values for variables:

```
input city $ 1-11 distance 13-20;
cards;
Rome        1430
Mexico City 10640
Hong Kong   13200
;
```

Formatted Input style, where pointer positions can be set and data read from that position for a number of columns based on the format described:

```
input @1 city $11. @13 distance 5.;
cards;
Stockholm   1450
Los Angeles 8780
Bangkok     12860
;
```

The next style is the Delimited Input -- this uses the DLM= option which specifies the character to use as a delimiter between values:

```
infile cards dlm='~';
input city $ distance;
cards;
Paris~330
New York~5530
Sydney~215660
;
```

Note that in the example I have used the character '~' as a delimiter, but it is possible to use other characters as the delimiter as well.

The "Double Space" Delimiter Style, with the use of the '&' symbol, is an interesting one as it allows for both for text to be input, as long as there is a single space between words for a text string, and two or more spaces separating variable values:

```
input city $ & distance;
cards;
Frankfurt  640
Rio de Janeiro  11060
Singapore  10810
;
```

The last style that will be looked at here is known as the Named Input style that sets the value from the variable based on the variable name to the left of the value as the following code demonstrates:

```
input city= $ distance=;
cards;
city=Copenhagen distance=953
distance=6800 city=Nairobi
city=Tokyo distance=15260
;
```

Note that the one advantage to this type of style is that the values do not have to be in order.

Based on the SAS documentation, it is possible to mix the methods inside an INPUT statement but caution is needed.

All of these styles are good where the data coming in is either in a structured form, i.e. set to columns, as would appear like in an Excel spreadsheet with no wrapping of lines.

USING THE DIFFERENT INPUT STYLES TO GET OUR DATA

Now using these input styles to get the data into a dataset.

Most listings of our data are usually in the form of columns so methods like the List, Column, Formatted and "Double Space" are the best. In practice, the best of this section is the Formatted or "Double Space" styles as these catch situations where you are bringing in a formatted value and character variables with two or more words. An example of such a situation is given below:

```
DATA
      2009-01-15  HYPERMAGNESEMA    MILD
      2009-01-15  ABDOMINAL CRAMPS  MODERATE
      2009-01-28  NAUSEA            MODERATE

INPUT STATEMENT
      input startdt yymmdd10. event $ & severity $;
```

In the example, the words "ABDOMINAL CRAMPS" are treated as one input value as there is only one space between the words -- the '&' modifier lets SAS know to do this in the INPUT statement.

The Delimited Input style is very similar to the Tab Delimited format that is often used to transfer data between applications. To implement this the DLM option is specified in the INFILE statement. For a Tab character, the syntax used is:

```
INFILE cards DLM='09'x;
```

where '09'x is the hexadecimal code for the ASCII Tab character.

Another form of this input is the CSV file where the delimiter character is the ',' but an issue with this is that while the character can be used as a delimiter, it also may be used in text, so take care when using the ',' as a delimiter.

The Named Input Style is not often used, but some say its origins start way back as a form of transporting data from one system to another. Here is an example of some code that produces this form of output for the first five records from the CLASS dataset in the SASHELP library:

```
1    data _null_;
2       set sashelp.class (obs=5);
3       put (_all_) (=);
4    run;

Name=Alfred Sex=M Age=14 Height=69 Weight=112.5
Name=Alice Sex=F Age=13 Height=56.5 Weight=84
Name=Barbara Sex=F Age=13 Height=65.3 Weight=98
Name=Carol Sex=F Age=14 Height=62.8 Weight=102.5
Name=Henry Sex=M Age=14 Height=63.5 Weight=102.5
```

Using this output and the Named Input style, it is possible to bring the data back into a dataset using the following code:

```
data class;
   length Name $8 Sex $1 Age Height Weight 8;
   input Name= $ Sex= $ Age= Height= Weight=;
cards;
Name=Alfred Sex=M Age=14 Height=69 Weight=112.5
Name=Alice Sex=F Age=13 Height=56.5 Weight=84
Name=Barbara Sex=F Age=13 Height=65.3 Weight=98
Name=Carol Sex=F Age=14 Height=62.8 Weight=102.5
Name=Henry Sex=M Age=14 Height=63.5 Weight=102.5
;
```

A MORE COMPLEX EXAMPLE

The following is an example of a table that was originally saved in RTF, but was saved as a text file using Microsoft Word without page breaks -- this can be easily done within Word and other word processors, but the procedure does depend on your word processor so refer to the relevant documentation.

```
Cook and Eat (Yum Yum), Inc.                                        Page 1 of 19
New Sauce
                                  Listing 4
                                Adverse Events

Subject Number: 1.001

--------------------------------------------------------------------------------
#001  Event: HYPERMAGNESEMA
      Preferred Term: Hypermagnesaemia
      Body System: Metabolism and nutrition disorders
      Start Date: 2009-01-15  Stop Date: 2009-01-25
      Severity: MILD        Relationship: NONE        Action: NONE
--------------------------------------------------------------------------------
#002  Event: ABDOMINAL CRAMPS
      Preferred Term: Abdominal pain
      Body System: Gastrointestinal disorders
      Start Date: 2009-01-15  Stop Date: 2009-01-27
      Severity: MODERATE    Relationship: NONE        Action: NONE
--------------------------------------------------------------------------------
#003  Event: NAUSEA
      Preferred Term: Nausea
      Body System: Gastrointestinal disorders
      Start Date: 2009-01-28  Stop Date: 2009-02-02
      Severity: MODERATE    Relationship: POSSIBLE    Action: STUDY DRUG HELD
--------------------------------------------------------------------------------
```

Fortunately the text coming in was well structured. Note that several lines make up the one record, hence the code was much easier than it could have been.

```
data ae_data;

   ** Define our data variables **;
   attrib ptnum length=$5 label='Subject Number'
          aenum length=$3 label='AE Number'
          event length=$100 label='AE Event'
          pt length=$100 label='Preferred Term'
          soc length=$100 label='Body System'
          startdt length=$10 label='Start Date'
          stopdt length=$10 label='Stop Date'
          severity length=$10 label='Severity'
          relation length=$10 label='Relationship to Study Drug'
          action length=$20 label='Action Taken';

   ** Retain the Suject Number across records **;
   retain ptnum '';

   ** Location and name of text input file **;
   infile 'filename and location' length=len;

   ** Get the line of code being processed **;
   input flg $varying200. len @;

   ** If subject number, then get that number **;
   if flg=:'Subject Number:' then do;
       input @17 ptnum $5.;
   end;

   ** If first character is '#' then start of a record to input.  Note    **;
   ** that the character '/' is used to go to the next line of input.     **;
   else if flg=:'#' then do;
      input @2 aenum $3. @14 event $ & /
            @23 pt $ & /
            @20 soc $ & /
            @19 startdt $  @42 stopdt $ /
            @17 severity $ & @43 relation $ & @63 action $ &;
      output;
      call missing(aenum,event,pt,soc,startdt,stopdt,severity,relation,action);
   end;
run;
```

In the example, the file was well structured -- if the text started with the text 'Subject Number' then it got the subject number, but if the text started with a '#' then this was an indication that it was the start of an event.

If the example was not well structured, as shown in the following example, then this approach would not be possible and some other form of either pre-processing, or processing each line as a text string which itself would have to be processed individually, would have to be done -- this approach is beyond this paper:

```
Cook and Eat (Yum Yum), Inc.                                    Page 1 of 19
New Sauce
                                  Listing 4
                                Adverse Events

Subject Number: 1.001

--------------------------------------------------------------------------------
#001  Event:    HYPERMAGNESEMA
  Preferred Term:                 Hypermagnesaemia
      Body System:Metabolism and nutrition disorders
   Start Date: 2009-01-15Stop Date: 2009-01-25
           Severity: MILD Relationship: NONE Action: NONE
--------------------------------------------------------------------------------
```

CONCLUSION

As can be seen from the examples given in this paper, there are a number of ways that the INFILE and INPUT statements can be used to get data that is in a structured from into a SAS dataset. The fact that the original request was to work on a Word file did not hinder our approach to the problem after the file was saved in a text file format.

CONTACT INFORMATION

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

David Franklin
TheProgrammersCabin.com
16 Roberts Road
Litchfield, NH 03052
Cell: 603-275-6809
Email: dfranklin@theprogrammerscabin.com
Web: http://www.TheProgrammersCabin.com