Paper 107-2010

Challenge! Reading Mainframe Hex Delimited Flat File Where Each Line Has Different Layout

Anjan Matlapudi and Knapp, J. Daniel
Pharmacy Informatics, PerformRx, The Next Generation PBM,
Philadelphia, PA

ABSTRACT

This paper illustrates how to design an appropriate input program to handle a complex file layout using data collected from pharmacy and health insurance information about individuals. Various INFILE and INPUT options are illustrated in the process, and some related functions are considered.

The input file is the output of a COBOL program pulling data from a DB2 database which is then brought to the PC via FTP. Each record contains 6 types of information, called segments, for a person. The segments and the fields within are divided by unprintable hexadecimal codes, which SAS represents with notations like the hexadecimal numbers 1E (Segment separator) and 1C (Field separator) respectively. A further complication is the use of 1D which is the group separator to separate repeating segments. There are also groups of repeating fields within a segment. Since the fields do not have a fixed length and they may be missing on some records, there is no fixed record layout for the file.

Although the program was written for the PC, the technique is applicable for any system. All the tools discussed are in BASE SAS®. The typical attendee or reader will have some experience in SAS, but not a lot of experience dealing with the input of external data.

INTRODUCTION

The SAS® System has excellent facilities for importing and manipulating data between platforms. This presentation is designed to review how to convert a hierarchical EBCDIC (Extended Binary Coded Decimal Interchange Code) text file from an IBM mainframe source into a set of SAS datasets on an ASCII (American Standard Code for Information Interchange) system. The examples in this paper explain where and how to use the INFILE and INPUT statements. Alternative approaches using character handling functions are also discussed.

THE FILE STRUCTURE

The file begins with a header record and ends with a trailer that we will exclude. The remaining records are divided into segments of related fields. Each record starts with '02' and ends with '03'. Below we have shown two data records.[1]

The first segment, known as G1, is exceptional and will be eliminated from our discussion. The other segments begin with 1E (Dec 30), but may be preceded by 1D (Dec 29) indicating that that segment may repeat within one record (see segment 'AM07'). Segments are further divided into fields that begin with 1C (Dec 28). Each field is divided into a two character identifier followed by the value of the field. For example, 'AM' is the identifier and '04' is the value for segment 'AM04'. The next field in 'AM04' has the identifier, C2 (in dark red) and value, '3424342355', which is the identifier of the person whose information is on this record. CC gives the first name and CD the last.

```
    Header Segment
RULE:    ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+---
1    CHAR  .00T4454831521009339070001  0648558201008020648P11622760 . 58
     ZONE  0335333333333333333333333322333333333333333333353333333320       Segment
     NUMR  2004445483152100933907000100064855820100802064801162276003       Identifiers
     Start of Text
2    CHAR  .G1608191393160076051B1445483152 1011487752192      20090727PAN00104  . .AM04.C23424342355   Line 1
     ZONE  0433333333333333333333343333333333332333333333333322222333333333544333333221144331433333333333
     NUMR  2716081913931600760512144548315201011487752192000002009072701E0010400EC01D0C323424342355
```

---

[1] The real data files contained 5 to 10 million data records.

```
                                                                                  Two Character
              1D Group Separator, IE Segment Separators, 1C Field Separators    Name of the Field

     89   .CCKAMBLE.CDRICHARD.C90.C10100...AM07.EM1.D20202029.E103.D750458058701.E70000030033.D300
   ZONE   1444444441445444452143314333331114433144314333333333143331433333333333331433333333333314333
   NUMR   C33B1D2C5C342938124C390C310100DEC1D07C5D1C420202029C5103C4750458058701C570000030033C4300

    177   .D5030.D61.D80.DE20090724.DK09.C802.28EA..AM03.EZ01.DB1063421121..AM05.4C2.5C01.6C99.7C1
   ZONE   1433331433143314433333333314433143331334411443314533144333333333311443313431343313433134
   NUMR   C45030C461C480C4520090724C4B09C3802C2851EC1D03C5A01C421063421121EC1D05C432C5301C6399C731

    265   00933907.E820090727.HB1.HC99.DV0000164H.5E01.6EAF.5C99.E820090727.HB1.HC99.DV0001013D.5E
   ZONE   3333333314333333333144314433145333333341343313444134331433333333314431443314533333333134
   NUMR   00933907C5820090727C821C8399C4600001648C5501C6516C5399C5820090727C821C8399C4600010134C55
                        Overpunch signs in dollar fields
    353   01.6E41..AM11.D90000164H.DX0000000{.DQ0001529I.DU0000164H.DN00. 415
   ZONE   3313433114433143333333341453333333371453333333414533333333414433O
   NUMR   01C6541EC1D11C4900001648C480000000BC4100015299C4500001648C4E003

3  CHAR   .G16116825601600760 51B1445483152 1011487752192     20090727PAN00104   ..AM04.C24545456666   Line 2
   ZONE   0433333333333333333334333333333332333333333333332222233333333354433333221144331433333333333
   NUMR   2716116825601600760512144548315201011487752192000002009072701E0010400EC1D04C324545456666

     89   .CCANJAN.CDMAT.C90.C10100A...AM07.EM1.D233020222.E103.D755111019401.E70000033300.D300.D5
   ZONE   1444444414444514331433333411144331443143333333331433314333333333333314333333333333314333143
   NUMR   C331EA1EC34D14C390C3101001DEC1D07C5D1C4233020222C5103C4755111019401C570000033300C4300C45

    177   030.D61.D80.DE20090729.DK09.C802.28EA..AM03.EZ01.DB1487668901..AM05.4C1.5C01.6C99.7C1009
   ZONE   3331433143314433333333314433143331334411443314533144333333333311443313431343313433134333 3
   NUMR   030C461C480C4520090729C4B09C3802C2851EC1D03C5A01C421487668901EC1D05C431C5301C6399C731009

    265   33907.E820090729.HB1.HC99.DV0000454{.5E01.6EAF..AM11.D90000439{.DC0000025{.DX0000010{.DQ
   ZONE   3333314333333333144314433145333333371343313444114433143333333371443333333714533333337145
   NUMR   33907C5820090729C821C8399C460000454BC5501C6516EC1D11C490000439BC430000025BC480000010BC41

    353   0000845I.DU0000454{.DN00. 377
   ZONE   3333333414533333333714433O
   NUMR   00008459C450000454BC4E003

   CHAR   .990648558000008408 8.                                      49
   ZONE   0333333333333333333330222222222222222222222222222222220    <- End of the Text
   NUMR   2990648558000008408830000000000000000000000000000000003
           ^
           |
 Trailer  Segment
```

MAIN LOGIC OF THE PROGRAM:

How can the file, described in the preceding section, be read? How should the separating fields be used?. Since the file is very hierarchical in structure it is easiest to preprocess the file to a segment structure before trying to read the fields. Without doing this one cannot take advantage of the DELIMITER (DLM) option of the INFILE statement. The variable length of fields and the delimiter, 1C, suggest that LIST INPUT is most appropriate. The problem is that one must first identify the segments which are separated by the hex number 1E. Consequently, we are led to a two-step process – 1) change the physical organization so that each record is a segment and several records are needed to get all the information for one person, and 2) read each record and send the information to the appropriate output dataset.

STEP 1 – CHANGE PHYSICAL STRUCTURE OF THE FILE

Here is the program:

```
filename inp "C:\path\Input files\mainframe.txt";
filename temp "C:\path\Datasets\segments.dat";
```

The FILENAME statement ties the FILEREF, INP, to the input physical file location. It separates this information from the DATA step; hence it is easier to modify. In addition it provides various options to provide the flexibility to handle many different data sources. The second FILENAME statement does the same for the output file.

The following DATA step creates the new file with segment structure. It illustrates handling external file I/O. No SAS datasets are output in this step. One should not create SAS datasets when none are needed because this activity takes up a significant portion of the execution time for a DATA step.

```
*----break up record into segments discarding unused segments----*;
data _null_ ;
  length seg $200 segnm $4 ;
  infile inp dlm='1E'x truncover lrecl=400 firstobs=2 ;
  do i = 1 to 6 ;
    input seg :$char200. @ ;
    segnm = substr (seg , 2, 4) ;
*----segments are skipped to simplify the code for this paper.----*;
    if segnm in: ("G1" "99" "AM07" "AM11") then ;
    else
    do ;
       seg = translate(seg, " " , "1D"x ) ;
       file temp ;
       put seg ;
    end ;
  end ;
run;
```

The INFILE statement associates the FILEREF, INP, with an input buffer. In addition, it provides options for flexibility. The DLM option, which specifies the "field" separator[2], is most important to our task, since the segments are separated by 1E[3]. Otherwise one must use character handling functions to manipulate the data.

TRUNCOVER is the contemporary way to handle variable length files. It accepts that data that is there without going to the next record looking for a field that is short. The default value is FLOWOVER.[4]

When records are longer than the default length, 256 bytes, then the LRECL option is required. It determines the longest length that the buffer should need to hold a complete record.

FIRSTOBS was set to 2 in order to skip over the header record.

The FILE statement does for output what the INFILE statement does for input.

In addition to providing a place to specify options, the INFILE and FILE statements are executable. They set the current buffer. In the above program, INPUT applies to FILEREF INP and its associated buffer instead of the default FILEREF DATALINES. Similarly PUT applies to TEMP instead of the default LOG. The concept is important because it means that you can read from multiple input buffers and write to multiple output buffers.

The INPUT statement does the actual reading, in this case, from the INP file because of the preceding INFILE statement. There are three types of input, LIST INPUT, COLUMN INPUT, and FORMATTED INPUT. Usually formatted input is the most appropriate choice for complex reading, but in this case we have a modified form of list input because the separator 1E determines the length of the segment. The use of a format is a direct command to read the specified number of byte, i.e. ignoring separators when required. The colon in front of $CHAR200. says to respect the separator, i.e. stop reading when the separator is encountered. It provides a handy way to have the correct combination of formatted input with list input.

The trailing @-signs says to hold the input record until either the next INPUT statement applied to that buffer or the bottom of the implied loop of the DATA step (whichever comes first). Consequently the same record is read in the DO-loop 6 times. The limit is 6 because there are always exactly 6 segments in the files that this program reads.

---

[2] For the purposes of this step, the whole segment is a field.

[3] SAS uses an “X” to refer to hexadecimal numbers. For example, 1CX indicates the decimal 28. In character form this is “1C”X.

[4] FLOWOVER goes to the next record when a field is short. MISSOVER throws away short the short field and does not go to the next record. It is important for historical reasons, but usually TRUNOVER is the option of choice.

There is also a double trailing @ which holds a record over the iteration of the DATA step. In all but the simplest of situations it is probably best to avoid the double trailing @ because it leaves dirty messages in the log.

The special segments G1 and '99' are skipped in this program. G1 occurs at the beginning of every data record as a sort of header to a record, and '99' is the trailer at the end of the file.

The TRANSLATE function is used to eliminate the code 1D. No segment was repeated within an input record, so the code was just a nuisance and this was a good time to get rid of it since this simplifies the next step.

In this step both the reading and writing are done in an explicit loop. The common idiom in SAS is one record in one out with no explicit loop. Why is this different? We want to change the structure of the data. Although one could force the SAS idiom, it would make the code more error prone and harder to read. One should learn to write DATA steps that differ from the natural rhythm of SAS when the data structure calls for it.

STEP2 – OUTPUT A SAS DATA SET FOR EACH SEGMENT

The DATA step is organized by subroutines – one for each segment. This helps to make the step easier to read and modify since the code associated with a segment is in one place. The function of the main routine is to control the whole process.

The main routine consists of the DATA statement with the list of output datasets, a LENGTH statement including all the variables, and a SELECT block to decide which subroutine to call for current segment. If an unknown code is encountered, the program aborts.

The subroutines consist of INPUT and OUTPUT statements for that segment plus any special code needed to handle that segment. For example, segment AM05 can have repeating fields. The INPUT statement assumes one repeat and a CHECK field to prove there are no more. When CHECK is not missing, the step aborts. The TRUNCOVER option allows including the repeat variables in the AM05 routine, since they will be made blank if they aren’t there and will be filled in if they are. This subroutine structure makes the program easy to read and modify.

The two preceding paragraphs illustrate one technique for checking the input assumptions to preserve the integrity of the program and the programmer.

The hex number 1C is the field separator and the first two bytes are the field identifier.
We use +2 to skip over this portion of the field and the colon modified INFORMAT to preserve the LIST INPUT nature of the file as explained in the previous DATA step.

```
*----write out wanted segments ----*;
   data am04 (keep = seq_num CardH_Id fname lname Ec1_code Group_num)
        am07 (keep = seq_num Rx_Service_RefNum_Quali Rx_Service_refNum
                     Product_Service_ID_Qualifier Product_Service
                     Quantity_Despensed  Fill_Number Days_Supply
                     Compound_Code DAW_Code Date_Prescription_Written
                     Submission_Clarif_Code Submission_Clarif_Code
                     Other_Coverage_Code  Unit_of_Measure)
        am03 (keep = seq_num Prescriber_ID_Qualifier Prescriber_ID)
        am05 (keep = seq_num COBCount CoverageType IDQualifier PayerID
                     PayerDate CoverageType PaidQual AmountPaid1 RejectCount
                     RejectCode PaidCount PayerDate2 PaidCount2
                     PaidQual2 AmountPaid2 RejectCount2 RejectCode2)
        am11 (keep = seq_num Ingredient_CostDispensing_Fee Patient_Paid_Amount
                     Sales_Tax_Amount Usual_Customary_Charge
                     Basis_Cost_Determination)

      ;

           Length segnm                            $ 4
                  seq_num                            8
               /* am04 */
                  CardH_Id                         $ 20
                  fname                            $ 12
                  lname                            $ 15
                  Ec1_code                       8
                  Group_num                    $ 8
               /* am03 */
                  Prescriber_ID_Qualifier      $ 2
                  Prescriber_ID                $ 15
               /* am05 */
                  COBCount                     $ 1
                  CoverageType                 $ 2
                  IDQualifier                  $ 2
                  PayerID                      $ 10
                  PayerDate                    $ 8
                  PaidCount                      8
                  PaidQual                       8
                  Amount_Paid1                 $ 8
                  RejectCount                    8
                  RejectCode                   $ 2
               /* other repeated fields */
                  CoverageType2                $ 2
                  PayerDate2                   $ 8
                  PaidCount2                     8
                  PaidQual2                      8
                  Amount_Paid2                 $ 8
                  RejectCount2                   8
                  RejectCode2                  $ 2
        ;
        infile temp dlm = "021C"x truncover ;
        input segnm :$char4. @ ;
        select (segnm) ;
           when ("AM04")do; link am04 ;end ;
           when ("AM03")do; link am03 ;end ;
           when ("AM05")do; link am05 ;end ;
           when ("AM04", "AM03", "AM05") /* skip for now */
           otherwise error "Bad segment name" ;
        end ;
      return ;

    *----routine for each segment----*;

      AM04:
     input
        +2 CardH_Id                              :$char20.
        +2 fname                                 :$char12.
        +2 lname                                 :$char15.
        +2 Ec1_code                              :1.
        +2 Group_num                             :$char8.
     ;
     output am04 ;
        seq_num + 1 ;
   return ;

   AM03:
     Input
        +2 Prescriber_ID_Qualifier               :$char2.
        +2 Prescriber_ID                         :$char15.
      ;

     Output am03 ;
              seq_num + 1 ;
   return ;

   AM05:
        input
            +2 COBCount                          :$Char1.
```

```
           +2 CoverageType                    :$char2.
           +2 IDQualifier                     :$char20.
           +2 PayerID                         :$char10.
           +2 PayerDate                       :$char8.
           +2 PaidCount                       :1.
           +2 PaidQual                        :1.
           +2 Amount_Paid1                    :$char8.
           +2 RejectCount                     :1.
           +2 RejectCode                      :$char2.
     /* possible repeat */
           +2 CoverageType2                   :$char2.
           +2 PayerDate2                      :$char8.
           +2 PaidCount2                      :1.
           +2 PaidQual2                       :1.
           +2 Amount_Paid2                    :$char8.
           +2 RejectCount2                    :1.
           +2 RejectCode2                     :$char2.
    /*Convert Over punch sings to dollar ammount*/;
    AmountPaid1 = put(input(Amount_Paid1, ZD8.2), 8.2);
    AmountPaid2 = put(input(Amount_Paid2, ZD8.2), 8.2)
    ;
    if missing(COBCount) then abort ;
    output am05 ;
    seq_num + 1 ;
  return ;
     ;

     *** routines for AM07 and AM11 are similar but not shown here ***;
  run ;
```

CONCLUSION

We have illustrated how to make the reading of fairly complex external data into a simple program by separating it into two well structured steps. The key here is to recognize the structure of the separators requires introducing two steps instead of trying to do the reading in one step. Steps not mentioned carry out analysis of the data and move it into other systems for use by doctors and pharmacists.

REFERENCES

Zirbel, Doug. Functioning JCL Into a SAS® Relational Database Table: Your Portable Tutorial On Character Functions, Plus an MVS™ Batch Bonus, *Coders' Corner* Paper 96-25.

Kuligowski, T. Andew. Datalines, Sequential Files, CVS, HTML and More – Using INFILE and INPUT Statements to Introduce External Data into the SAS System, *SUGI 31 Tutorials* Paper 228-31.

ACKNOWLEDGMENTS

We like to acknowledge Ian Whitlock for helping us in preparing this paper. We also like to thank Marian Whitlock for sharing her ideas to put all things together.

We did like to acknowledge Mr.Shimels Afework, The Sr. Director of our company. The PerformRx was formed in 1999 as a division of the AmeriHealth Mercy Family of Companies. We provide specialized pharmacy benefit management services (PBM) through proactively managing escalating pharmacy costs while focusing on clinical and financial results.

CONTACT INFORMATION:

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

Name Anjan Matlapudi
Senior Pharmacy Analyst
Address PerformRx, The Next Generation PBM
200 Stevens Drive
Philadelphia, PA 19113
Work Phone: (215)937-7252
Fax: (215)863-5100
E-mail: anjan.matlapudi@performrx.com
matanjan@hotmail.com

Name Knapp, J. Daniel
Senior Manager
Address PerformRx, The Next Generation PBM
200 Stevens Drive
Philadelphia, PA 19113
Work Phone: (215-937-7251
Fax: (215)863-5100
E-mail: Daniel.Knapp@performrx.com