Tutorial - Variable Length Records
Peter L. Rikard, Virginia Commonwealth University

Reading variable length records is a pain in most computer languages. Most programmers background is in dealing with fixed length records and the features in most languages provide no help in working with strange length records.

Definition

Variable length records are records in non SAS datasets whose ACTUAL record length is not constant.

A fairly circular definition, but it does EXCLUDE SAS datasets, and variable length record datasets where the records are really all the same length.

Notes

Records that are all the same length in a variable length file may be read as if they are fixed.

SAS datasets are not variable length records.

A SAS dataset has "fixed" length data records.

Examples:

SOME SMF records are really variable in length:
- 4 - step record, is
- 5 - JOB record, is sort of
- 6 - output writer, is
- 7 - data lost, isn’t
- 8 - i/o config, is

Hierarchical Files
Some is
Some ain’t

SAS on the other hand gives you a number of tools to easily handle some of the most difficult records. If you learn the capabilities of each tool, you can read any file.

Tools
Pointers
- trailing @
- trailing @@
- @variablename

INFILE parameters
- MISSOVER/STOPOVER
- LENGTH=
- COLUMN=
- END=

DO - END groups

Trailing @
The single trailing @, holds a record within the current execution of the DATA step.

The double trailing @@, holds a record until either you input past the end of the record OR until YOU release it.

Examples:

DATA;
  INPUT x9999999 @@ ;
  (read from first record)
  INPUT yyyyy @ ;
  (still on first record)
  INPUT zzzzz ;
  (STILL on first record but not held anymore)
  INPUT qqqqq ,
  (read next record)

Pointer - @variablename

INPUT 37 ggggg ;
  moves the pointer to column 37, so the next input item is read beginning there.

INPUT @variablename gggggg ;
  moves the pointer to the column of the current value of the variable.

The variable may get its value from assignment or from being INPUT.

INFILE parameters

LENGTH=variablename
The SAS variable will contain the value of the length of the last record input.

INFILE ccc LENGTH=LEN ;
INPUT vvvvvvvv @ ;
LEN now has value of the length of the current record.
The variable will contain the current position of the pointer.

EX: Records normally contain 20 variables. Some records have missing variables at the end and the record is shorter.

You only want COMPLETE records.

INFILE CCC LENGTH=LEN ;
INPUT @ ;
(LEN is now set)
IF LEN=200 ;
(subsetting if)
INPUT (X1-X20) (10.0) ;

Only records of length 200 get past the subsetting if, so only complete records are read.

***********************************************************************
DO - END

Use of various kinds of DO groups allows easy coding of reading the multiple segments of variable length records.

Instead of an input statement for each multiple segment, a DO-END group may be executed many times.

EX:
DO J = 1 TO COUNT,
    INPUT ccccc @;
    other statements
END,

COUNT may be calculated from the length of the record or be an input variable.

Note: An older method is to use LINK/RETURN statements to do this task. This is marginally more expensive than DO/END.

***********************************************************************

SAS datasets are not variable length. Each data record is fixed in length and has room for every variable kept in the DATA statement.

If you read datasets with "repeated" data blocks where many of the blocks are normally missing, you may waste lots of space in your SAS dataset.

Instead, create an observation only for each non-missing segment.

EX: A record on the input dataset contains 20 bytes of demographic data on the family and up to 15 segments, each 10 bytes long, one for each family member. Create multiple observations from a single input.

INFILE ggg LENGTH=LEN;
INPUT 20 bytes of demo @;
NUMBER = (LEN-20)/10;
   calculate number of 10 byte segments.
DO J = 1 TO NUMBER ;
    INPUT 10 bytes @; (a person)
    OUTPUT;
END;

***********************************************************************

System Management Facility (SMF) records are old friends to anyone doing billing, performance analysis or anything else with an IBM system. These records are stored in a single variable length file. Most of them are really fixed length records, but some aren't. IBM however has given the person reading these records a few "handles" for accessing the records. SAS makes the most of these with its tools.

The type 4 SMF record

100 bytes of fixed info
2 byte OFFSET - this is the location of the last item
2 byte COUNT
   COUNT number of data fields each 8 bytes long, total is variable in length.
1 byte LENGTH
   Accounting data, variable in length, total of LENGTH bytes
4 bytes of last junk

***********************************************************************

EX: Reading the Type 4 record, only the "fixed" data.

INFILE SMF ;
INPUT @2 REC PIB1. @;
   (select the type 4 records)
INPUT ..first 100 bytes...
   OFFSET PIB2.
   @OFFSET +1
   ..last 4 bytes...

variable OFFSET is used to move the pointer over the variable portion of the record.
EX: Read the type 4 records and form two datasets.

(1) A SAS dataset with one record for each type 4 record read.
(2) A SAS dataset for each tape device used.

DATA SMF4 DROP=DEVICE UNITTYPE CHA ADDR UNITADDR EXCP)
  TAPE(KEEP=DEVICE CHA ADDR UNITADDR EXCP);

The DATA statement allows you to create several datasets in one step and control the variables in each.

INPUT •• 100 bytes ••
  (OFFSET COUNT) (PIB2.) @. EXCP=0.
  DO INDEX = 1 to COUNT,
    INPUT (DEVICE UNITTYPE CHA ADDR UNITADDR) (PIS1.) EXCP PIB4. @,
    (input each segment)
    EXCP=EXCP+EXCP,
    (accumulate the excp count)
    IF UNIT=3 THEN OUTPUT TAPE
    (output tape use stats)
  END:
  INPUT accounting data...;
  OUTPUT SMF4;
  (output single record for the step)

***************************************

INFILE options - MISSOVER/STOPOVER

An input that reaches beyond the end of the current record has probably been coded wrong or the record is too short.

EX: Record should have 20 variables but sometimes has fewer. (prior example)

INPUT (X1-X20) (10.),
  reads 20 items, if the record is too short, then SAS reads data items from beginning of next record.

********************

MISSOVER and STOPOVER give you control when the record is shorter than it should be.

INFILE ccc MISSOVER;

INPUT (X1-X20) (10.);
  If the record contains 140 bytes then only X1-X14 will have values and X15-X20 will be set to MISSING.

INFILE ccc STOPOVER;
  If the input record is too short the data step will stop running immediately. (consider a short record an error and STOP)

******************************************************************************

Trailing @@

A simple use of the trailing @@, many observations on one record.

DATA;
  INPUT X @@;
  CARDS;
  1 2 3 4 5 66 77 777 ...

A more complex example is below. Many hierarchical files contain variable records. With one type for one level and different types of records for other levels. If you wish to "capture" higher level information at lower level records, one way is in the input process. (another is a merge)

EX:
Hierarchical file
  Long family demographic record followed by a short record for each member.

First byte indicates type.

INFILE nnn END=EOF;
  (the END= gives a truth variable being true when the end of file is reached)

INPUT demographic record;

LOOP: IF EOF THEN GOTO ENDIT;
  INPUT @1 TYPE @@,
    (Hold record - for next pass of the data step)
    IF TYPE=1 THEN RETURN;
      (no more person records for this family)
    IF TYPE=2 THEN DO;
      INPUT person data;
      OUTPUT;
      (see if next record is another family member)
    END;
  ENDIT: STOP,

******************************************************

Conclusion: SAS lets you read a messy variable length record in segments.
It gives you tools that allow you to easily control the execution within the data step. It allows you to perform your own tests to insure you are reading the record correctly.

Practise helps.