ABSTRACT

The routine SAS® programming techniques for reading raw data files are the List, Column and Formatted modes of the INPUT statement. While the standard use of these modes will usually satisfy many data requirements, the exceptions require that the SAS programmer understand how to go beyond the "default" use of the INFILE and INPUT statements. In addition, version 6.07 of the SAS system offers additional INFILE statement options and INPUT statement features that many SAS programmers have not had the opportunity to discover.

In this paper we will discuss the limitations of the "default" approaches, and present examples of advanced programming techniques using the INFILE and INPUT statements that can be used to overcome these limitations. Included in this tutorial will be explanations of new features such as the use of the DELIMITER option for List input mode, the use of the Tilde and DSD options to read delimited data, and an explanation of the often overlooked "Named" input mode.

INTRODUCTION

The routine SAS programming techniques for reading external data files are the List, Column, and Formatted modes of the INPUT statement. While the standard use of these modes satisfies many requirements, the exceptions require you to understand additional features of the INPUT and INFILE statements. This paper we provide a review of the standard operation of List, Column and Formatted input, and explain the often overlooked Named input mode. Finally, we present exception examples and advanced programming techniques that solve them.

PLAN YOUR INPUT

Before coding an INPUT statement, plan how to read your data. This is especially important when reading external files with many records, or many potential SAS variables, or both. When processing many variables, code potential BY variables early in the INPUT statement. This places those variables near the beginning of the Program Data Vector, which results in more efficient processing in later steps of your program. When reading many records, examine the raw data as they are input, and keep only those observations and variables needed for the application.

Also, code each variable on a separate line. This makes your code easier to read and maintain, and allows space for instream comments or definitions for each variable. For example:

```
TITLE 'EXAMPLE 1';
DATA TEST01;
  INPUT LAST $ /* EMPLOYEE LAST NAME */
  FIRST $ /* EMPLOYEE FIRST NAME */
  MIDDLE $ /* EMPLOYEE MIDDLE INITIAL */
CARDS;
SMITH MARY R
RUN;
```

STANDARD INPUT TECHNIQUES

List Input

The List Input mode requires at least one blank space between each value in the raw data. The standard List INPUT statement creates SAS variables by reading from left to right in the input stream. The variable's data type is assumed to be numeric unless a "$" follows its name. Each variable is stored in eight bytes.

The following data records are suitable for List input. Note that the data do not have to line up in columns because the delimiter is the blank.

```
TITLE 'EXAMPLE 2';
DATA TEST02;
  INPUT LAST $ /* EMPLOYEE LAST NAME */
  FIRST $ /* EMPLOYEE FIRST NAME */
  MIDDLE $ /* EMPLOYEE MIDDLE INITIAL */
  MONTH $ /* HIRE DATE MONTH */
  DAY $ /* HIRE DATE DAY */
  YEAR $ /* HIRE DATE YEAR */
DATALINES; /* NEW INSTREAM IDENTIFIER ALIAS: CARDS */
  SMITH MARY R FEB 25 1992
  LEE-MORE MARYBETH KATHRYNE JULY 20 1992
RUN;
```

List input is the SAS System's simplest input mode. It allows the input record to be relatively free form. Again, the only requirement is a delimiting blank between each field.

Default List input has major limitations: character data cannot contain embedded blanks; character fields longer than eight characters will be truncated to a length of eight; and non-standard data (e.g., "numeric" fields such as $400,000, 25JUN1992, any type of binary or hexadecimal coding) cannot be read properly. For example, date information presented in several "standard" date formats, such as 25JUN1992, or 6/25/92, causes default List input to
generate error messages and convert the data to numeric missing values ("." (default)).

A new name for an old statement was introduced in Example 2. Beginning in SAS Version 6, the DATALINES statement is an alias for the CARDS statement. This statement informs the SAS System that the information following it is raw data. A line containing a ";" (semicolon) follows the last line of the data. Like the CARDS statement, the DATALINES statement has a companion DATALINES4 statement for use with data containing semicolons. Four semicolons, ";;;;", in positions 1-4 signal the end of the input begun with a DATALINES4 statement.

**Column Input**

Column input allows you to overcome most of the deficiencies of List input regarding character data; however, Column input mode requires every data record read by one INPUT statement to have the same layout. Each variable's identification and location (beginning position and ending position) is specified in the INPUT statement. By modifying the data in Example 2, we can use Column input to read the same information:

```
DATA TESTD3;
TITLE 'EXAMPLE 3';
INPUT LAST $ 1-8 /* EMPLOYEE LAST NAME */
    FIRST $ 10-17 /* EMPLOYEE FIRST NAME */
    MIDDLE $ 19-26 /* EMPLOYEE MIDDLE NAME */
    MONTH $ 28-35 /* HIRE DATE MONTH */
    DAY 37-38 /* HIRE DATE DAY */
    YEAR 40-43; /* HIRE DATE YEAR */
DATALINES;
SMITH MARY R FEB 25 1992
LEE-MORE MARYBETH KATHRYNE JULY 20 1992
RUN;
```

Column input allows more flexibility than List input: character data can be up to 200 characters in length and contain embedded blanks, and no place holder is required for missing data. The main disadvantage of Column input is that the input must have a fixed structure that allows for the greatest possible width of any data value. Also, Column input, like List input, cannot handle any unusual character data (e.g., hexadecimal), or numeric data containing even so much as a comma.

**Formatted Input**

The most complex standard SAS input mode is Formatted input. Formatted input does not require the fixed layout of Column mode, but does require that you know three facts: (1) either the absolute or the relative position of each variable on the input record; (2) the width of each field; and (3) the nature of the data being read in each field. Formatted input is the only mode of input capable of reading nonstandard numeric and character data. Formatted input mode requires that every variable be formatted in the same way on each record read by an INPUT statement. Example 3 is recoded using the Formatted mode as follows:

```
DATA TESTD4;
TITLE 'EXAMPLE 4';
INPUT @1 LAST $8. /* EMPLOYEE LAST NAME */
    FIRST $8. /* EMPLOYEE FIRST NAME */
    MIDDLE $8. /* EMPLOYEE MIDDLE NAME */
    MONTH $8. /* HIRE DATE MONTH */
    DAY 2. /* HIRE DATE DAY */
    YEAR 4.; /* HIRE DATE YEAR */
DATALINES;
SMITH MARY R FEB 25 1992
LEE-MORE MARYBETH KATHRYNE JULY 20 1992
RUN;
```

This example uses the pointer and field width notations. In Formatted input, the SAS System can use a combination of the absolute column pointer "@", the relative column pointer "+", and the line pointers "/" and "#" to position the input pointer within a record or set of records. In Example 4, we know the absolute position of the LAST field: it starts in position 1 of every input record ("@1"). We need only know the relative positions of the FIRST, MIDDLE, MONTH, DAY, and YEAR variables, if we assume they will be separated from each other by one position ("+1").

For the relative pointer notation to work correctly we must also know the width of each input field. The informat of each field provides this information. In the simplest cases, SAS informats describe each data field's data type (the optional "$" for character type) and field width (a number followed by a period). Implicit decimal places for numeric values are indicated by a number following the informat period (".1"). Special raw data classes such as date values, numeric data containing commas, zoned decimal, binary, and other complex types of numeric and character data are indicated by an informat name before the field width. For example:

```
DATA TESTD5;
INPUT @1 UNITS 4. /* QUANTITY SOLD */
    $6 REGION $2. /* SALES REGION */
    AMOUNT COMMA10. /* AMOUNT OF SALES */
    DATE MMDDYY8.; /* DATE OF SALES */
DATALINES;
1934 NC $10,998.20 08/01/92
1949 NC $1,972.98 08/05/92
RUN;
```

Formatted input is the most flexible and powerful method of reading data. Although it cannot read the free form records that List mode can read because it is dependent on field widths, Formatted input can read almost any other data type or record layout.

Formatted input mode allows a "shortcut" notation for repeated pointer and informat combinations:

```
INPUT (LAST, FIRST, MIDDLE, MONTH) (8. +1)
    NAME, HIRE DATE MONTH */
    DAY 2. /* HIRE DATE DAY */
    YEAR 4.; /* HIRE DATE YEAR */
This shortcut allows you to define the variables to be read in the first list, enclosed in parentheses, followed by the format
list, enclosed in parentheses. The SAS System reads the data items defined in the variable list, "reusing" the format list until it runs out of data.

Named Input

A special input technique used when a raw record contains both variable names and values is called Named Input. When writing an INPUT statement for Named input, each variable name must be followed by an equals sign (=) in both the INPUT statement and the data. Character variables are indicated by a "$" following the equals sign.

```
TITLE 'EXAMPLE 6';
DATA TESTD6;
INPUT LAST= $ FIRST= $ MIDDLE= $ MDNTH= $ DAY= YEAR=;
DATALINES;
'*/' EMPLOYEE LAST NAME */
'*/' EMPLOYEE FIRST NAME */
'*/' EMPLOYEE MIDDLE NAME */
'*/' HIRE DATE MONTH */
'*/' HIRE DATE DAY */
'*/' HIRE DATE YEAR */

LAST=SMITH FIRST=MARY MIDDLE=R MONTH=FEB DAY=25 YEAR=1992
FIRST=MARYBETH MIDDLE=KATHRYNE LAST=LEE-MORE MONTH=JULY DAY=20 YEAR=1992
FIRST=LEE LAST=WATKINS
RUN;
```

In the first line, the "/" at the end of the line indicates the next data line must be read to complete the observation.

Note that the variables in the second observation are in a different order. Named input allows this, because the SAS System identifies the variables by the variable name and the "=" delimiter. Order is no longer important.

In the third observation, some variables were omitted. Missing values are assigned to variables not included in the input record. In a more extreme case, the SAS System reads all variable names present in the Program Data Vector and sets those that are not present to missing. If named variables are present in the raw data but are not coded in the INPUT statement, the SAS System indicates an error and sets the variable _ERROR_ to 1.

Named input allows an extremely flexible layout of data records with one major limitation: once Named input begins, other input modes cannot be used. However, you may use other input modes before beginning the Named input coding. In Example 7, Column input is used to read the employee number field first, and the remaining fields are read with Named input:

```
TITLE 'EXAMPLE 7';
DATA TEST07;
INPUT EMP_NO 1-5 /* EMPLOYEE NUMBER */
LAST= $ /* EMPLOYEE LAST NAME */
MIDDLE= $ /* EMPLOYEE MIDDLE NAME */
FIRST= $ /* EMPLOYEE FIRST NAME */
MONTH= $ /* HIRE DATE MONTH */
DAY= /* HIRE DATE DAY */
YEAR= /* HIRE DATE YEAR */
DATALINES;
46954 LAST=SMITH FIRST=MARY MIDDLE=R MONTH=FEB / DAY=25 YEAR=1992
63221 FIRST=MARYBETH MIDDLE=KATHRYNE LAST=LEE-MORE / MONTH=JULY DAY=20 YEAR=1992
36823 FIRST=LEE LAST=WATKINS
RUN;
```

Combinations

Before we go beyond the standard uses of the INPUT statement, we note that the above modes can be combined to satisfy a wide variety of input requirements. For example:

```
TITLE 'EXAMPLE 8';
DATA TEST08;
INPUT FIRST $ LAST $ MIDDLE $ +1 YEAR MMDDYY8_ REGION $ 30-31
a33 SALARY COMMA7. , a1 JOB_TITL $30_;
DATALINES;
ALFRED E NEWMAN 01,01,89 SALES REPRESENTATIVE
ALBAN J BARKLEY 09,03,48 PUBLIC RELATIONS
SO 32,000
NC 19,500
RUN;
```

The above example uses List input mode for the three simple character fields FIRST, LAST, and MIDDLE. Formatted input is required to locate, read, and convert YEAR to a SAS date value according to the MMDDYY8. informat. REGION is read using Column input, while SALARY, which contains commas, needs a special informat. This INPUT statement also uses the "/" relative line pointer to read two physical records with one INPUT statement.

Where are the Data?

In all our examples so far, the data have been in the same input stream as the SAS program, and identified by the CARDS or DATALINES statements. When your data are in an external file separate from the program input stream, you need an INFILE statement to tell the SAS System where to find the data.

Using the data from Example 8, the external file DATA09 contains the following:

```
ALFRED E NEWMAN 01/01/89 $32,000
SALES REPRESENTATIVE
ALBAN J BARKLEY 09/03/48 NC 19,500
PUBLIC RELATIONS
RUN;
```
The code now includes an INFILE statement to link the external file to the SAS program for input. The code is as follows:

```
TITLE 'EXAMPLE 9';
DATA TEST09;
INFILE DATA09;
INPUT FIRST $ /* EMP LAST NAME */
   LAST $ /* EMP FIRST NAME */
   MIDDLE $ /* EMP MIDDLE INITIAL */
   +1 YEAR MMDDYY8. /* SALES YEAR */
   REGION $ 30-31 /* SALES REGION */
   @33 SALARY COMMA7. /* YEARLY SALARY */
   @1 JOB_TITL $30. /* JOB TITLE */
RUN;
```

The INPUT and INFILE statements interact to accomplish complicated input tasks.

Limitations of the Standard Input Modes

As Example 9 illustrates, you may combine SAS standard input modes as needed to read most types of records and raw data types. There are, however, situations in which the raw data record or field is unusual enough to require more advanced techniques. Such situations include:

- The starting position of a field is unknown when using Formatted or Column input.
- Some data elements are omitted from some records.
- The field width of a data element varies across records.
- Part of a record must be read differently based on data values in another part of the record.
- A data line must be reread.
- A [Tab] or other special character is used as a delimiter.
- Multiple files are read in the same DATA step.

ADVANCED INPUT TECHNIQUES

Format Modifiers for List Input

List input has the virtue of simplicity, and, if it were not for two character variable limitations (no embedded blanks and a maximum length of 8 bytes), could be used to read many types of data. Both problems have solutions that do not require Column or Formatted input.

The "&" Format Modifier

The embedded blank problem can be cured using the "&" format modifier:

```
TITLE 'EXAMPLE 10';
DATA TEST10;
INPUT NAME $ & AGE;
DATALINES;
John 12
J. J. 11
RUN;
```

Even though "J. J." in the second input line contains an embedded blank, the List INPUT statement reads it properly because the "&" modifier indicates that a field may contain one or more single embedded blanks. The input field will be read beginning from the next nonblank position until either 1) two or more blanks occur in a row, or 2) eight characters have been scanned.

The ":" Format Modifier and the LENGTH Statement

The eight-character constraint can also be solved without losing the relative simplicity of List input. Consider the following DATA step:

```
TITLE 'EXAMPLE 11';
DATA TEST11;
LENGTH NAME $15;
INPUT NAME $ & AGE;
DATALINES;
Jonathan 12
Jerry Jeff 10
RUN;
```

The LENGTH statement at the beginning of the DATA step defines NAME as a character variable of length 15. Using that information and the "&" modifier, the List INPUT statement successfully reads both embedded blanks and character values of more than eight characters.

The ":" Format Modifier

One of the problems with Column and Formatted input modes is that you must know the starting position of each field. This requirement means, in turn, that input records must be rigidly structured, often wasting file space and causing data entry problems. The scanning feature of List input can solve the problem, but only at the cost of losing some flexibility in reading nonstandard data. To have the best of both worlds, consider the following DATA step:

```
TITLE 'EXAMPLE 12';
DATA TEST12;
LENGTH NAME $ 15;
INPUT NAME $ & DATE : DATE7.
   WEIGHT : 3.;
DATALINES;
Jonathan 07JAN92 75
Jerry Jeff 19FEB92 66
RUN;
```

The input records have the free form style of List input, but still contain nonstandard data such as dates. The ":" (colon) format modifiers tell the INPUT statement to scan from the next nonblank column for the next value, while the informats
describe how to read that value. As shown in the Example 12, the informat must follow the ":" format modifier. Note the "::" modifier is also used to allow single embedded blanks, and the LENGTH statement allows the NAME variable to be longer than eight characters. You can write the above code without the LENGTH statement, by using a character informat for the NAME variable:

```
TITLE 'EXAMPLE 13';
DATA TEST13;
  INPUT NAME : & $15.
  DATE : DATE7.
  WEIGHT : 3.;
DATALINES;
  Jonathan 07JAN92 75
  Jerry Jeff 19FEB92 66
RUN;
```

The "::" Format Modifier

The "::" (tilde) format modifier is new with Release 6.07. This format modifier is only valid when you specify the DSD Option of the INFILE statement, which we will discuss in a later section of this paper.

The tilde format modifier is used with character data where single and double quotes are contained in the raw data and must be preserved.

The data for this example are contained in an external file called DATA14:

```
"HOT WORD PROCESSING TIPS" FEB 1993
"CENTERING MULTIPLE LINES IN WORDPERFECT" JAN 1993
"40 SLICK WORD PROCESSING TIPS" SEP 1992
```

The following SAS statements will read these raw data:

```
TITLE 'EXAMPLE 14';
DATA TEST14;
  INFILE DATA14 DSD;
  INPUT ARTICLE - $ 41.
  ISSUE & $ 8.;
RUN;
```

The Tilde format modifier retains double quotes and any other delimiters read into the variable ARTICLE. PROC PRINT illustrates this:

```
EXAMPLE 14
OBS  ARTICLE
  1  "HOT WORD PROCESSING TIPS"
  2  "CENTERING MULTIPLE LINES IN WORDPERFECT"
  3  "40 SLICK WORD PROCESSING TIPS"
```

Format Modifiers for all Types of Input Statements

The "?" and "??" Format Modifiers

The "?" and "??" format modifiers are very useful when dealing with numeric data that may contain invalid values. You may use these modifiers with the List, Column, and Formatted input modes.

When the SAS System encounters invalid date, the "?" format modifier sets the value of the variable to missing, lists the contents of the line in the log, and suppresses the printing of the invalid data message. The "??" format modifier also causes the value of the variable to be set to missing, but neither the invalid data message nor the printing of the line will occur. More importantly, with invalid data, the "?" format modifier causes the SAS variable _ERROR_ to be set to 1, while the "??" modifier prevents _ERROR_ from being set to 1.

Use the "?" and "??" format modifiers with caution. If you want to set invalid numeric data to missing values, these format modifiers will reduce the number of extraneous error messages in the log. If your application requires that invalid numeric data to be investigated and corrected, the error messages should remain.

HINT: Use these format modifiers to investigate the content of variables. For example, consider a field that is supposed to contain a formatted date. If you read the field with a date informat, and it is either blank or an invalid date, the associated SAS variable is set to missing. If it's important to know which condition applies, read the field with a character informat, and test the character field for blanks first. Then, convert the character variable to a date variable

```
[DATEVAR=INPUT(CHARVAR, ?? MMDDYY8.);]
```

and test for a missing, and therefore invalid, date variable.

Special Techniques

Reading a Variable Amount of Data: $VARYING.

Consider the following data records containing information about the creation of SAS data sets:

```
WORK.TEMPDATA09AUG90
OUT.PERMDATA07DEC90
```

The first raw record refers to a working SAS data set called TEMPDATA (WORK.TEMPDATA), the second refers to a permanent data set called PERMDATA. A date follows each data set name. The "::" and "::" modifiers are not helpful here, since there is no blank space after the data set name to tell the input scanner where to stop. One solution is to code the width of the data set name field in the records:

```
13WORK.TEMPDATA09AUG90
12OUT.PERMDATA07DEC90
```

Now the problem becomes how to use the number coded at the beginning of each record to read the file name. The $VARYING. informat provides a solution:
TITLE 'EXAMPLE 15';
DATA TESTIS;
DROP WIDTH;
INPUT WIDTH 1-2
DSNAME $VARYING11-WIDTH
DATE DATE7.;
DATALINES;
13WORK_TEMPDATA09AUG90
12OUT_PERMDATA07DEC90
RUN;
The $VARYINGn. informat uses the numeric variable WIDTH, read from positions 1-2 of the input record to determine how to read the DSNAME field. $VARYING has a width specification of 17, since the maximum length of a fully-specified SAS data set name is 17 characters (an eight-character libref, a period, and an eight-character SAS name). Since the SAS System determines the storage length of a variable from its informat when there is no LENGTH statement or other prior information, the length of DSNAME is 17. In the TESTIS data set created above, the SAS System pads the value of DSNAME with four blanks on the right for the first observation, and five blanks for the second observation.

Example 15 showed that the SAS System can determine how to execute an INPUT statement for a record based on data values within the record. For the varying-width file name, we coded the width of the field in positions 1 and 2 of the input record. This is a powerful technique, but critical information is often not in the input records. Consider the original input records:

WORK_TEMPDATA09AUG90
OUT_PERMDATA07DEC90

How can we obtain the width of the file name? There are really two problems here. First, we need to read the input record and scan it ourselves to determine where the file name ends and the date begins. Second, we need to read each record twice: once to "parse" it, and once to read DSNAME and DATE.

TITLE 'EXAMPLE 16';
DATA TESTIS;
DROP TEXT WIDTH;
INPUT TEXT S24.;
WIDTH=LENGTH(TEXT)-7;
INPUT al DSNAME $VARYING17. WIDTH
DATE DATE7.;
PUT N = WIDTH= DSNAME= DATE=;
DATALINES;
WORK_TEMPDATA09AUG90
OUT_PERMDATA07DEC90
RUN;

Example 16 illustrates a solution to one of the two problems we have described, and one which many beginning SAS users are unaware of: use of the SAS System to examine raw input data treated as "text" to find out what that input data contains. In this case, we've read a raw record as a character variable called TEXT with a width of 24 characters (the maximum size of our input records). Using the SAS LENGTH function, we can determine where the data ends, and subtract 7 to derive the WIDTH variable.

The SAS log for Example 16 shows us we've solved one of our problems (we have calculated WIDTH), but not the other:

1
2
3
4
5
6
7
8
9
NOTE: The data set WORK.TEST16 has 1 observations and 2 variables.
The PROC PRINT confirms that the DATA step did not do what we wanted:

Example 16

When we executed the first INPUT statement in the DATA step, the SAS System assumed we were finished with the first line of the raw data and released it. The second INPUT statement then read the second line of data using the WIDTH determined for the first line. As the log and output show, the WIDTH for the first line was too wide for the second line, and an extra "0" was tacked onto the end of the file name. To reread the first line, we need to recode the first INPUT statement using a trailing "@":

TITLE 'EXAMPLE 17';
DATA TESTIS;
DROP TEXT WIDTH;
INPUT TEXT $24.@;
WIDTH=LENGTH(TEXT)-7;
INPUT al DSNAME $VARYING17. WIDTH
DATE DATE7.;
PUT N = WIDTH= DSNAME= DATE=;
DATALINES;
WORK_TEMPDATA09AUG90
OUT_PERMDATA07DEC90
RUN;

The trailing "@" sign tells the SAS System not to release the input line after executing the first INPUT statement. The second INPUT statement executes on the same line and reads the correct data, as shown by the SAS log below:
TITLE 'EXAMPLE 17';
DATA TEST17;
DROP TEXT WIDTH;
INPUT TEXT $24. a;
WIDTH=LENGTH(TEXT)-7;
INPUT @1 DSNAME SVARYING17. WIDTH DATE DATE7.;
PUT _N_= WIDTH= DSNAME= DATE=;
DATALINES;

NOTE: The data set WORK.TEST17 has 2 observations and 2 variables.

The trailing "@" has a variety of uses. For example, if we needed to discard invalid records from our raw file of SAS data set names, we could use the trailing "@" technique:

TITLE 'EXAMPLE 18';
DATA TEST18;
DROP TEXT WIDTH;
INPUT TEXT $24. a;
YIOTH=INDEX(TEXT,'90')-6;
PUT WIDTH=;
IF WIDTH < 3 THEN DELETE;
INPUT a1 DSNAME $VARYING17. WIDTH DATE DATE7.;
PUT _N_= WIDTH= DSNAME= DATE=;
DATALINES;

WIDTH=13
_N_ =1 WIDTH=13 DSNAME=WORK.TEMPDATA DATE=11178
_N_ =2 WIDTH=12 DSNAME=OUT.PERMDATA DATE=11298
NOTE: The data set WORK.TEST18 has 2 observations and 2 variables.

We altered the WIDTH calculation to check for a valid date string using the SAS INDEX function. When the DELETE statement executes, causing the SAS System to return to the top of the DATA step, the input line that is on "hold" is discarded, as the following SAS log indicates:

TITLE 'EXAMPLE 19';
DATA TEST19;
RETAIN START;
INPUT TEXT $24. a;
WIDTH=INDEX(TEXT,'90')-6;
PUT 'AFTER PARSE: '_N_= TEXT= WIDTH=;
IF WIDTH < 3 THEN DELETE;
IF _N_ = 1 THEN START = 1;
ELSE START = START+WIDTH+7+1;
INPUT aSTART DSNAME $VARYING17. WIDTH DATE DATE7. @@;
PUT 'AFTER READ: '_N_= START= WIDTH= DSNAME= DATE=;
DATALINES;

WIDTH=13
_N_ =1 START=1 WIDTH=13 DSNAME=WORK.TEMPDATA DATE=11178
_N_ =2 START=21 WIDTH=12
_N_ =3 START=12 DSNAME=OUT.PERMDATA DATE=11298
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.

NOTE: The data set WORK.TEST19 has 2 observations and 5 variables.

INFILE Statement Options

The MISSOVER Option

In the best of all possible worlds, every field in an input record is defined, or has an explicit missing value coded for...
absent data points. Unfortunately, SAS INPUT statements often produce the following results:

```plaintext
DATA TEST20;
INPUT NAME $ AGE WEIGHT;
DATALINES;
NOTE: Invalid data for WEIGHT in line 9 1-6.
RULE: ·····1·····2·····3·····4
NAME=Alfred AGE=12 WEIGHT=68 ERROR =1 N =2
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: The data set WORK.TEST20 has 2 observations and 3 variables.

The NOTES occurred because the input data looked like this:

John 12 75
James 11
Alfred 12 68

When a value was not found for WEIGHT on the second record, the SAS System took its default action. The default action is to go to the next input record and read the first value, "Alfred", for the numeric variable WEIGHT. This causes a missing value for WEIGHT, an invalid data message, and erroneous analyses if this data error were unnoticed. (You can explicitly request this default option by coding FLOWOVER on the INFILE statement.)

Fortunately, the SAS System provides a simple solution to the problem:

```plaintext
TITLE 'EXAMPLE 21';
DATA TEST21;
INFILE DATALINES MISSOVER;
INPUT NAME $ AGE WEIGHT;
DATALINES;
John 12 75
James 11
Alfred 12 68
RUN;
```

MISSOVER on the INFILE statement tells the List INPUT statement not to go to a new line if expected values are not found in an input record. Instead, the SAS System sets any variable that would occur in the blank portion at the end of the current input record to missing, and reads the next record properly.

Note that we have used an INFILE statement option with instream data in Example 21. Not all INFILE options can be used with DATALINES input, but many useful INFILE options can be applied.

The TRUNCOVER Option

The TRUNCOVER Option (an enhancement in Release 6.07) is similar to the MISSOVER option, except when reading variable length records. In Example 22, the MISSOVER option sets the variables which are shorter than the specified informat to missing. The TRUNCOVER option will keep the variables as they are read from the raw data, regardless of the length of the record (up to the length specified in the informat).

The raw data for this series of examples is contained in an external file of variable length records, DATA22:

```
52
5512
15243
67
```

Using the MISSOVER Option:

```plaintext
TITLE 'EXAMPLE 22 WITH MISSOVER';
DATA TEST22;
INFILE DATA22 MISSOVER;
INPUT PROD_TON 5.; /* PRODUCTION TONS */
RUN;
```

EXAMPLE 22 WITH MISSOVER

```
OBS    PROD_TON
  1          52
  2
  3    15243
  4
```

By using the TRUNCOVER option instead of the MISSOVER option the following PROC PRINT would result:

```
EXAMPLE 22 WITH TRUNCOVER

OBS    PROD_TON
  1    52
  2
  3    15243
  4    67
```

The STOPOVER Option

The MISSOVER option allows the SAS System to handle missing information at the end of an input record by skipping the unrecorded data and setting the corresponding SAS variable(s) to missing. This automatic missing data is not always desirable, since it will obscure other problems on the input record. In such cases you must identify and deal with the underlying conditions. The STOPOVER option may be more appropriate:

```plaintext
TITLE 'EXAMPLE 23';
DATA TEST23;
INFILE DATALINES STOPOVER;
INPUT NAME $ AGE WEIGHT;
DATALINES;
John 12 75
James 11
Alfred 12 68
RUN;
```

The STOPOVER option on the INFILE statement causes the SAS System to react to missing information at the end of an
input record by setting the automatic DATA step variable _ERROR_ to 1, printing the current raw record and contents of the Program Data Vector, and stopping the DATA step.

1  TITLE 'EXAMPLE 23';
2  DATA TEST23;
3     INFILE DATALINES STOPOVER;
4     INPUT NAME $ 
5     AGE 
6     WEIGHT;
7     DATALINES;

ERROR: INPUT statement exceeded record length.
INFILE CARDS OPTION STOPOVER specified.
RULE: ---------------1---------------2---------------3---------------4
8 James 11 NAME=James AGE=11 WEIGHT_. _ERROR_=1 _N_=2
NOTE: The SAS System stopped processing this step because of errors.
NOTE: SAS set option ODS=0 and will continue to check statements. This may cause NOTE: No observations in data set.
WARNING: The data set WORK.TEST23 may be incomplete. When this step was stopped there were 1 observations and 3 variables.

The START Option

Another handy INFILE option is the "START=variable" option. This option is used primarily when copying raw data from one file to another if the first portion of the data is not required in the new file. Once the value of the variable is identified in the program, the SAS System copies from that position to the end of the record.

This option could be used to remove sequence numbers at the start of each record. In Example 24, the variable BEGIN contains the value of the position within the record where the data to be copied start. When the new file is created the characters will be shifted 7 positions to the left. The data from position 8 in the original file will be in position 1 in the new file.

1  TITLE 'EXAMPLE 24';
2  DATA TEST24;
3     INFILE DATALINES START=BEGIN;
4     INPUT;
5     BEGIN = 8;
6     FILE DATA24 NOTITLES;
7     PUT _INFILE_;
8     RUN;

The FILEVAR Option

The FILEVAR option lets you dynamically close one input file and open another file for input. With this option you can read similar data from many files, without concatenating the files in your command language (JCL, DCL, UNIX). This also implies that the record format, logical record length and the block size can vary. This option eliminates the overhead of having multiple files open at the same time.

1  TITLE 'EXAMPLE 25';
2  DATA TEST25;
3     INPUT NAMEFILE $ 1-8;
4     INFILE MONTHLY FILEVAR = NAMEFILE END = EOF;
5     DO UNTIL(EOF);
6     INPUT REJ_CODE $ 10-12
7     REJ=TOTALS 15-25;
8     END;
9     CARDS;
10    userid.JAN1992
11    userid.FEB1992
12    userid.MAR1992
13    RUN;

In this example, NAMEFILE contains the list of files to be read. Each iteration of the DATA step reads a different MONTHLY file. You can combine the data in many ways, without recoding the INFILE statement. For example, you could process a single month, or three months for a quarterly report.

The FILEVAR option can also be used on FILE statements, to provide a similar flexibility for file output.

The DELIMITER Option

The DELIMITER option allows you to change the SAS System default delimiter (space) to another value. The new value can be one or many characters up to a maximum of 200 characters. By using the DELIMITER Option, an input record whose values are separated by commas could be read using a simple List INPUT statement.

1  TITLE 'EXAMPLE 26';
2  DATA TEST26;
3     INFILE DATALINES DELIMITER = ',';
4     INPUT LAST $ /* EMPLOYEE LAST NAME */
5     FIRST $ /* EMPLOYEE FIRST NAME */
6     MIDDLE $ /* EMPLOYEE MIDDLE INITIAL */
7     MONTH $ /* HIRE DATE MONTH */
8     DAY /* HIRE DATE DAY */
9     YEAR; /* HIRE DATE YEAR */
10    DATALINES;
11    SMITH,MARY,R,FEB,25,1992
12    RUN;

In the above example, the "&" format modifier is not required because a space no longer signals the end of a variable. With the DELIMITER option you can read raw data from many different software packages and on many different platforms.

The DSD Option

The DSD option is used with List input to read raw data containing delimiters. The DSD option uses the comma as a delimiter instead of a blank. If your data include commas, use the DELIMITER option to specify another character as a delimiter.

The DSD option generates a missing value when the raw data contain two consecutive delimiters.

1357
**CONCLUSION**

The use of List, Column, Formatted, and Named input modes, with the default values of the INFILE statement, allows you to read most forms of raw data. A more thorough understanding of the features of the INPUT statement, as well as the INFILE statement, will enable you to read more exotic forms of raw data. This paper only touches on some of the rich and powerful features that are available for the INPUT and INFILE statements. To acquaint yourself with all the power of these two statements, make sure you review the latest documentation from the SAS Institute.

**REFERENCES**


**ACKNOWLEDGEMENTS**

The idea for this paper came from a paper presented at NESUG '80, "Advanced Input Techniques," by Judith H. Mopsik and Jeff Philipe of Atlantic Research Corporation. We would like to thank Judith H. Mopsik, Robert Ormsby, Warren Repole, Scott Rampola and Elizabeth Henrickson, who reviewed this paper and offered many constructive suggestions.

The authors can be reached at:

- ARC Professional Services Group
  1301 Piccard Drive, Second Floor
  Rockville, MD 20850
  - Federal Reserve Board
    21st and C Streets, N.W.
    Washington, DC 20551

SAS is a registered trademark of the SAS Institute Inc. in the USA and other countries. ® indicates USA registration.