CHAPTER 1
AN INTRODUCTION TO READING RAW DATA WITH SAS

Overview

The goal when reading an external data file is to create a SAS data set or data view that SAS can process to produce meaningful reports and analyses. This book presents examples of reading external data files and instream data that you can adapt to read your own data. This chapter presents the concepts of reading these data sources with SAS.

Collectively, unprocessed data stored in an external data file or included as part of the job stream are termed raw data. DATA steps read raw data and create SAS data sets and views from the raw data. With the features of the INFILE and INPUT SAS language statements, you describe to SAS the structure of your raw data. You can also specify attributes of an external data file in the FILENAME statement.

As considered in this book, external files contain unprocessed data not stored in a SAS data set. These files can transfer information between software applications and the structures of these files can vary. External files are managed by your operating system, not by SAS. Depending on your operating system, you may refer to your external files as flat files, text files, sequential files, DAT files, or ASCII files.

There is an endless variety of ways in which to store raw data. For example, an external file from a clinical study could contain one data line for one patient for one set of lab tests. Another way of representing the same information is to write a series of data lines for one patient: the first data line contains the patient identifiers and is followed by several data lines, each data line containing the patient's results from a lab test. A third way to create this clinical study file is to place information for multiple patients in each data line. A patient identifier is followed by the lab test results. The information for the next patient continues on the same data line.

Most examples in this book show how to read raw data stored in external files. Another way to read raw data is to include the data as part of the job stream. A few examples of reading instream data are presented in this chapter.

Methods of Reading Raw Data

SAS has several ways of reading raw data. These include

- SAS language statements in the DATA step
- SAS functions
- the Import Wizard and the External File Interface (EFI)
- the SAS procedure IMPORT.

The INFILE statement describes the attributes of the external file containing the raw data that you want the DATA step to read. The INFILE statement can either directly name the external file or it can indirectly point to the external file with a fileref defined with the FILENAME statement or window. Typical attributes that you might specify in the INFILE statement are the delimiter between fields and the record length.

With the INPUT statement, you describe to SAS the structure of your data. An INPUT statement that uses simple list input may be able to read your external file by scanning the data lines for data values if your data values are separated by at least one delimiter such as a space. On the other hand, your external file may not have delimiters between data values and the data values may have a specific structure. To read that external file, you may need to use a different INPUT style, such as column input or formatted input.

SAS functions can also read and write external files. These functions either can be coded in a DATA step or can be used outside of the DATA step in the SAS macro language.

If you are running SAS in an interactive windowing environment such as Windows, UNIX, or OpenVMS, you can use the Import Wizard and the EFI to read and write external files. These features use a point-and-click interface that prompt you for information about your external files.

Also available under the SAS windowing environments listed above is the SAS procedure IMPORT. This PROC reads external files as well as tables in database management systems. PROC IMPORT can run in SAS interactive mode as well as in batch mode.

Understanding Data Sources

The way you write your DATA step depends on where your data are stored. Your raw data may be in one of two locations:

- part of the job stream
- stored in an external file.

Most examples in this book show how to read data stored in external files.

Reading Data That is Part of the Job Stream

Raw data that is part of the job stream follows the DATA step code that reads it. You would typically select this style of programming only if you were processing small amounts of data. Either a DATALINES or DATALINES4 statement precedes the raw data.

If you maintain older SAS programs, you may see the CARDS or CARDS4 statement instead of DATALINES or DATALINES4.

In the example that follows, a DATA step reads six variables from five data lines. The data lines are part of the job stream.

Example 1.1 Reading Instream Data Lines

```
data runners;
  input name $ age runtime1 runtime2 runtime3 runtime4;
datalines;
Scott 15 23.3 21.5 22.0 21.9
Mark 13 25.2 24.1 23.5 22.0
Jon 13 25.1 25.7 24.3 25.0
Michael 14 24.6 24.1 24.3 24.6
Matt 14 22.0 21.5 21.4 21.6
;;;;
```

Reading Data from External Files

An external file is managed by your operating system and not by SAS. SAS can read and write many types of external files. Your DATA step code manages the processing of these external files.

If the program above was submitted under Windows and the raw data were stored in the file c:\readdata\runnersapril.dat, the DATA step to read the external file could be written as follows. The INFILE statement identifies the external file containing the raw data.

Example 1.2 Reading Data Lines from an External File

```
data runners;
  infile 'c:\readdata\runnersapril.dat';
  input name $ age runtime1 runtime2 runtime3 runtime4;
run;
```

Understanding Data Values

Once you know the structure of the data value that you want to read or write, you must define to SAS whether your data value is numeric or character and you must determine the method that you want SAS to use to process that value. The coding of your DATA step statements requires that you understand the type of data you are processing. Instructions may also be needed to tell SAS how your data values are represented. Informats and formats provide this information.

A numeric data value represents a number. This value may be simply numbers or it may include characters such as a decimal point or a minus sign. A value written in scientific notation is also considered a numeric data value.

A character data value contains a character or sequence of characters. These characters can be letters, numbers, or symbols.

Guidelines for defining numeric data and character data are fully described in *SAS Language Reference: Concepts*, and *SAS Language Reference: Dictionary*.

SAS defines two styles of representing raw data: standard and nonstandard.

Standard data are character or numeric data values that can be read with list, column, formatted, or named input. A number with a decimal point or a preceding minus sign is considered a standard numeric data value as is a value represented in scientific notation.

Nonstandard data include numeric data values that contain nonnumeric characters. Examples include

- numbers with dollar signs or commas or both
- dates and times
- packed decimal and integer binary numbers.

Character data that is considered nonstandard would include data that was stored in EBCDIC but is being read on an ASCII system.

These values can be read only with informats or written only with formats. Informats translate the nonstandard data into a form that can be processed within SAS. Formats write out data values in a specific form that may be different than the SAS internal representation of the data value.

A date in the form mm/dd/yyyy is a nonstandard data value. To have SAS understand this as a date, you must read the value with an informat. SAS then translates this nonstandard data value to a numeric value—the number of days before or since January 1, 1960. When writing it out, if a format was not used, the date would be represented simply as the number of days before or since January 1, 1960. You need to apply a format to that data value to write it out in a form that is easily understood as a date.

This next example reads payment information. The three variables are payer id and two payment dates. The payment dates are read with formatted input with the MMDDYY10. informat. The PROC PRINT report that follows shows a format applied to PAYDATE1 and not to PAYDATE2. Therefore, the values of PAYDATE2 are presented the way SAS stores them—the number of days since January 1, 1960.

Example 1.3 Working with Dates

```
data payments;
  input id $4.   @6 paydate1 mmddyy10.
                 @17 paydate2 mmddyy10.;
datalines;
QDSW 04/15/2002  06/15/2002
JDHA 5-2-02      8-1-2002
MPWZ 12012002    03042003
;;;;
proc print data=payments;
  title 'Payment Dates';
  format paydate1 mmddyy10.;
run;
```

Output 1.3 PROC PRINT of PAYMENTS Data Set

Payment Dates

Obs	id	paydate1	paydate2
1	QDSW	04/15/2002	15506
2	JDHA	05/02/2002	15553
3	MPWZ	12/01/2002	15768

SAS language includes many informats and formats that can process different kinds of data. Refer to *SAS Language Reference: Dictionary* for complete specifications.

Reading Raw Data with the INPUT Statement

Your INPUT statement tells SAS how your raw data are structured. There are several different styles of input and you must determine the style that would best read your raw data. Your INPUT statement can be written in a combination of styles.

The four styles of input in SAS are

- list input and a hybrid of list and formatted input called modified list input
- column input
- formatted input
- named input.

Following are brief descriptions of the input styles. Complete information on coding your INPUT statements is covered in *SAS Language Reference: Dictionary*. The examples in this book use these input styles to read raw data.

Reading Data Lines with List Input

List input scans your input data lines for data values. Data values do not have to be aligned in columns, but they do have to be separated from one another by a space or other delimiter such as a comma (,). List input requires only that you specify the variable names to be assigned to the data values in your input data line. Unless defined elsewhere in your DATA step, such as in a LENGTH or an ATTRIB statement, a dollar sign ($) must follow the name of a character variable.

The type of data that list input can read is restricted to specific structures. The restrictions are as follows:

- Data values must be separated by at least one blank or by another delimiter.
- A real placeholder, not a blank, must represent a missing value. A single period (.) denotes the presence of a missing numeric value in your raw data. For data values separated by a delimiter other than a blank, the delimiter serves as a placeholder for the missing value.
- Processing character data values greater than the default character length of 8 bytes requires additional specifications. One way to do this is to define the length of the character variable prior to the INPUT statement by using a LENGTH, INFORMAT, or ATTRIB statement. Another way is by using modified list input where a colon and informat follow the character variable name. Modified list input is described later in this chapter.
- Specific options must be included if character data values can contain the delimiter.
- Fields must be read in the order they appear in the data line, but they do not have to occupy specific columns.
- Only standard data values can be read with list input. Use modified list input to read nonstandard data values.

Example 1.1 presents a DATA step that reads data with list input.

Reading Data Lines with Column Input

Column input reads standard data values that are aligned in specific columns in the data lines. The range in columns for a variable follows the variable name. If the variable is character, place a dollar sign between the variable name and the column range. Additional features of column input include the following:

- Placeholders for missing values are not required.
- Data values can be read in the order you specify; it is not necessary to specify in the INPUT statement the variables in the order they appear in the data lines.

- ❒ Data values must be in the same columns in all the data lines.
- ❒ Data values or parts of data values can be reread.
- ❒ Leading blanks within the field are removed.
- ❒ Values do not have to be separated by blanks or other delimiters.
- ❒ Column input can read only standard character and numeric data values.
- ❒ Character data values can contain embedded delimiters. For example, SAS can read multiple words as one data value.

An example of column input follows. This example demonstrates that columns can be reread.

Example 1.4 Reading Data Lines with Column Input

```
data stores;
  input  storeid $ 1-6 state $ 1-2
         phone 7-16 areacode 7-9
         zipcode 17-25 zip1 17-21 zip2 22-25;
datalines;
WI038192055539455491012 34
WI623560855538235370073 62
WI720071555548205500176 54
WI541241455500875320032 21
;;;;
```

PROC PRINT displays the STORES data set.

Output 1.4 PROC PRINT of STORES Data Set

Stores in Wisconsin

Obs	storeid	state	phone	areacode	zipcode	zip1	zip2
1	WI0381	WI	9205553945	920	549101234	54910	1234
2	WI6235	WI	6085553823	608	537007362	53700	7362
3	WI7200	WI	7155554820	715	550017654	55001	7654
4	WI5412	WI	4145550087	414	532003221	53200	3221

Reading Data Lines with Formatted Input

Formatted input provides you with the most flexibility when reading data lines. You can read both standard and nonstandard data with formatted input. Pointer controls in the INPUT statement can direct where SAS should read data for a specific variable and informats can specify the structure of the data value. Additional features of formatted input include the following:

- ❒ Character data values can contain embedded delimiters.
- ❒ Placeholders for missing values are not required.

- ❒ Data values can be read in the order you specify; it is not necessary to specify the variables in the order they appear in the data lines. Pointer controls can direct where SAS should read data values.
- ❒ Data values or parts of data values can be reread.

An example of formatted input follows. The column pointer controls, the at sign (@) and the plus sign (+), specify the position of the column pointer as SAS reads a data line. The number following the @ tells SAS to move to that column. The number following the + tells SAS to move the pointer that number of columns.

Example 1.5 Reading Data Lines with Formatted Input

```
data patients;

 input @1 id $5.

       @1 initials $3. +3 ssn comma11.

       @19 (test1-test3) (4. +1)  ;
datalines;
AFG03 999-99-0393 381  1.3   5
TEY01 999-99-7362            3
REW17 999-99-4313 25   3     0
;;;;
```

Starting in column 1, read a character variable that is five bytes in length. After reading the variable, the pointer is positioned at column 6.

Move the pointer back to column 1 and reread the data in columns 1-3. Move the pointer to the right three columns and read the next variable.

Move the line pointer to column 19. Read the three test values with the 4. informat. Skip one space between each of the test values.

PROC PRINT displays the PATIENTS data set.

Output 1.5 PROC PRINT of PATIENTS Data Set

Patients in Study

Obs	id	initials	ssn	test1	test2	test3
1	AFG03	AFG	999990393	381	1.3	5
2	TEY01	TEY	999997362	.	.	3
3	REW17	REW	999994313	25	3.0	0

Reading Data Lines with Modified List Input

Modified list input is a hybrid between list input and formatted input. As with simple list input, this style is restricted to reading variables in order. The data values do not have to be aligned in columns, but they do have to be separated from one another by a space or other delimiter such as a comma. Additionally, you can include informats that allow you to read more complex data values than you can read with simple list input. For example, modified list input can read nonstandard numeric data and character data values larger than 8 bytes.

Format modifiers and informats added to the INPUT statement enable you to read more complex data values than you can read with simple list input. The three format modifiers are:

- The ampersand (&) format modifier following the variable name tells SAS to read character data values that contain embedded delimiters. SAS stops reading the character data value when it encounters more than one consecutive delimiter.
- The colon (:) format modifier and an informat following the variable name tell SAS to read the data value with the informat and to read until it encounters the specified delimiter or reaches the width specified by the informat, whichever comes first.
- The tilde (~) format modifier following the variable name tells SAS to treat single quotation marks, double quotation marks, and delimiters within the data value as part of the data value.

An example of a DATA step that uses modified list input follows.

Example 1.6. Reading Data Lines with Modified List Input

```
data survey;

  infile datalines

          delimiter=',';

  input name : $15.

         comments ~ $50.;
datalines;
Mary Ann,More restrictions on emails
Scott,Did not like slogan "Our Team is Tops"
Luke,Would like to have comp time
Rosa,Would like manager's input on reports
;;;;
```

Indicate that raw data is part of the job stream and follows the DATALINES statement.

Specify that commas separate the two fields in the data lines.

Read to the delimiter or read 15 bytes, whichever occurs first.

Read up to 50 bytes. Treat single quotation marks, double quotation marks, and delimiters within the data value as part of the data value.

PROC PRINT displays the SURVEY data set.

Output 1.6 PROC PRINT of SURVEY SAS Data Set

Survey Results

Obs	name	comments
1	Mary Ann	More restrictions on emails
2	Scott	Did not like slogan "Our Team is Tops"
3	Luke	Would like to have comp time
4	Rosa	Would like manager's input on reports

Reading Data Lines with Named Input

Named input requires that the variable name be part of the data line. The variable name followed by an equal sign precedes the data value. Features of named input include:

- Named input can be used in combination with other input styles. Once you start named input, you must stay in that style to read the remaining variables.
- Only the variables named in the INPUT statement are read. The named variables that exist in data lines but not in the INPUT statement are not included in the output data set. They are, however, identified in the SAS log as not being defined. SAS sets the automatic error variable, _ERROR_, to **1** when named variables appear in the data lines but SAS does not find them in the INPUT statement.
- Named input can read only standard data values.
- As with simple list input, SAS defaults to assigning the lengths of character variables to 8 bytes. If your character data value is longer than 8 bytes, specify a LENGTH or ATTRIB statement before the INPUT statement.

An example of reading data lines with named input follows. This example demonstrates that only the variables specified in the INPUT statement are written to the output data set.

Example 1.7 Reading Data Lines with Named Input

```
data grades;

  length name $ 15;

  input name=$ math=;
datalines;
name=Linda english=95 math=94 science=90
name=Susan math=88 english=91 science=90
name=Mary Louise math=90 english=84 science=81
;;;;
```

Since the length of NAME is 15 and greater than the default of 8, place a LENGTH statement before the INPUT statement.

For named input, follow each variable name with an equal sign (=). If the variable is character, follow the equal sign with a dollar sign ($). Read two of the four variables.

PROC PRINT displays the GRADES data set.

Output 1.7 PROC PRINT of GRADES Data Set

Math Grades

Obs	name	math
1	Linda	94
2	Susan	88
3	Mary Louise	90

A Checklist for Specifying Your External File

SAS follows the instructions you specify in your statements when reading external files. With so much flexibility in the SAS language, you may have to specify several items to successfully read your external file.

The following list presents some of the items you may need to consider when coding your program.

How are the data values arranged in the data lines?

- ✔ Not column aligned, with delimiters separating the values
- ✔ Column aligned

What types of data values are you reading?

- ✔ Character
- ✔ Numeric
- ✔ Nonstandard numeric data

How are missing values represented?

- ✔ Blanks
- ✔ A character such as a period (.)
- ✔ A delimiter, if delimiters separate data values

Are the data values fixed or variable in length? If variable, what is the maximum length? If no delimiters separate data values, how do you determine the length of your data value that is variable in length?

How many data lines contain the information for one observation?

- ✔ One data line per observation
- ✔ Multiple data lines per observation
- ✔ Multiple observations per data line

What are the attributes of your external file?

- ✔ Variable-length records
- ✔ Fixed-length records
- ✔ Record length and block size

Are all your data lines structured the same way? Do you need to examine the data line to determine what type of data line it is before you completely read it?

Are all your data lines in one file? Are they in multiple files?

Is your external file on your local system or do you need to connect to a remote system to read the file?