

Getting Data into SAS®: INFILE and INPUT

Andrew T. Kuligowski, Nielsen Media Research

ABSTRACT / INTRODUCTION

The SAS System has numerous capabilities to store, analyze, report, and present data. However, those features are useless unless that data is stored in, or can be accessed by, the SAS System. This presentation includes an introduction to the INPUT and INFILE statements, which combine to provide a simple, yet powerful, method to pass data into the SAS System. Matters to be addressed are reading fixed and variable length files, .CSV files, in-stream data, informats, and more.

INFILE, INPUT, AND RELATED STATEMENTS

Much of the data that is available in an electronic format is stored in sequential (commonly referred to as "flat") files. There are three steps involved with making sequential file data, or *any* external data, known to the SAS System. The source of the data must be defined to SAS, the format of that data must be defined to SAS, and the data must be subsequently passed to SAS. There are two statements in the DATA step which combine to perform these tasks. The **INFILE** statement will define the data source, while the **INPUT** statement will codify the format and move the data into SAS.

The INFILE Statement

An external file is identified to a DATA step and subsequent INPUT statement(s) by the INFILE statement. There are three different ways to tie an external file to the INFILE statement. **See Figure E for an example of each method .**

The first mechanism is to actually specify the name of the external file, within quotes, in the INFILE statement. This method is useful under certain circumstances, such as one-time-only ad hocs. Its primary disadvantage is that it eliminates some flexibility -- in order to use the SAS routine to process different files, the user must either clone the routine and change the INFILE statement, or they must utilize the macro facility.

The second way is to provide a file reference to an external file. The file reference can be associated to the external file with a command to the host system, such as a JCL "DD" statement under IBM's MVS. An alternate method is to use the **FILENAME** statement under SAS. The FILENAME statement is a global statement which does not need to be included within a DATA step. The syntax for this statement, in this context, is:

```
FILENAME fileref 'external file';
```

The third method is a variation of the second, which is only applicable to "aggregate storage locations", such as an IBM MVS partition dataset. Again, the INFILE statement is provided with a file reference to an external file, along with a specific file or member reference enclosed in parentheses.

```
/* full file name - under Windows */  
INFILE 'c:\sasconf\sample.dat';  
  
/* FILENAME statement - under CMS */  
FILENAME TDATA 'SASCONF TESTDATA A1';  
INFILE tdata;  
  
/* file reference - under MVS */  
//TDATA DD DSN='userid.SASCONF.DATA',  
// DISP=SHR  
INFILE tdata(example1);
```

Figure "E" - INFILE Statement

Note that many of the options for the INFILE and FILENAME statements are dependent on the operating system. Please consult the appropriate "Companion" for your operating systems for details. Also note that it is possible to read from multiple external files from within the same DATA step. To accomplish this, there must be a separate INFILE statement for each external file. The appropriate INFILE statement should immediately proceed its corresponding INPUT statement.

The CARDS Statement

The most basic example of sequential input is the **CARDS** statement. (The statement name can be traced back to the time when the data would be found on actual physical punch cards. The alias **DATALINES** is more reflective of the modern meaning of the statement, and may be preferred by some readers.) When using this option, there is no separate sequential file. Instead, the last executable line of a DATA step will be the CARDS statement. The subsequent lines contain sequential input data. The end of the input data is signified by a single semicolon. (If your data might contain a semicolon, the CARDS4 statement can be substituted -- the end of input data is then signified by a string of 4 consecutive semicolons.) Even the use of the INFILE statement is optional when using CARDS; should the user choose to include it, there is a predefined CARDS file reference. **See Figure F for an example of the CARDS statement.**

```

DATA EXAMPLE;
  INFILE CARDS; /* Optional */
  INPUT W X Y $ Z ;
  CARDS;
4  31.2 emu 141
6  5.18 fry 288
3  29.1 act 671
1  14.4 ion 93
;

```

Figure "F" - CARDS Statement

The primary benefit of the CARDS methodology is simplicity. There is no need to locate and understand external data -- the external data is the user's to control. Conversely, this is also the main drawback. In a batch setting, the user must ensure that the data is included in the DATA step. In an interactive setting, it becomes the user's responsibility to actually *enter* the data at the keyboard, using the copy facilities of their editor, manual entry, or using the mouse to cut and paste. Further, the user is *supposedly* limited to working with one external file per DATA step. In actuality, by using the INFILE CARDS statement, making the CARDS the last input source used in the DATA step, and with some careful coding, it is possible to combine CARDS with other external files in the same DATA step. (Admittedly, the situations where one would *want* to override this alleged limitation are very limited.) See Figure G for an example of multiple INFILE statements with a CARDS statement.

```

FILENAME ADISK 'A:STATECNT.TXT';
DATA TEMP;
  DO UNTIL(LASTFILE);
    INFILE ADISK END=LASTFILE;
    INPUT STATE $ COUNT ;
    OUTPUT;
  END ;
  DO UNTIL(LASTCARD);
    INFILE CARDS END=LASTCARD;
    INPUT STATE $ COUNT ;
    OUTPUT;
  END ;
  STOP ;
CARDS;
TX 2
TN 1
FL 1
IL 1
CA 1
CT 1
;

```

Figure "G" - Multiple INFILES with CARDS

The INPUT Statement

The actual transfer of data from an external file to a SAS DATA step is performed by the INPUT statement. By default, the SAS System assumes all input data to be numeric. This can be overridden by pre-defining a variable to be character, or by

associating either a dollar sign (\$) or a character format with the variable on the INPUT statement.

A full discussion of the INPUT statement is not possible in this limited space - the current *SAS Language: Reference* devotes 30 pages to the statement! However, there are some basic tenets that should be reviewed in this setting.

There are five basic methods to describe a record to an INPUT statement. These methods can be combined on a single INPUT statement. However, it is suggested that users restrict themselves to one method per INPUT statement; this will avoid introducing additional complications in a program which will assist in debugging and in future maintenance on the routine. See Figure H for examples of each input method.

The most basic method is *list input*, in which the INPUT statement simply contains the name of each variable. Each value on a line of input data must be delimited by one or more blank characters (or other delimiter, as defined by the DELIMITER= or DLM= option on the INFILE statement). For this reason, missing data must be explicitly listed in the input file. Further, the standard list input method cannot be used for character variables that may contain embedded blanks.

The weaknesses of this approach can be overcome via *modified list input*, using format modifiers to provide additional flexibility to the INPUT statement. The Ampersand (&) format modifier will permit single embedded blanks in character variables. Similarly, the Colon (:) format modifier will allow the INPUT statement to ignore its default 8-character maximum on character variables. The Question Mark (?) format modifier can be used to bypass error processing should invalid data be expected and acceptable to the application - for example, attempting to read a character field into a numeric variable. A double Question Mark (??) conceals every indication of an error - the "Invalid Data" message and the echoing of the input line containing the bad data are omitted from the SASLOG, and the _ERROR_ variable is not set. A single question mark (?) only suppresses the "Invalid Data" message, allowing the other indications of an error to be dealt with normally.

In *Column input*, the start and end columns are listed after each variable name. Column input does not have the weaknesses described under list input - character variables can contain embedded blanks and can exceed 8 bytes in length, and missing values do not have to be explicitly defined in the input file. However, it is only useful when input data

is formatted consistently on each line; other methods must be employed for free-format data.

Formatted input is similar to the list and column input methods described above. However, each input variable must have an accompanying informat. This is useful for data in a "non-standard" form, such as packed decimal, dollar, or date values.

The least common method is *named input*. With this method, the INPUT statement has an Equal sign (=) following each variable. The input data must also contain the same series of "fieldname=value" pairs. Please note that this method is an exception to the "all methods are interchangeable" rule referenced earlier. Once an INPUT statement begins to reference named input, all remaining data on that line must be in named input format.

There is also a *null input*. An INPUT statement, followed by a semicolon, can be used to move to the next input record without any further processing of the current line.

```
DATA EXAMPLE;
  INPUT X
    Y $ ;          /*list input*/
  INPUT X 1-4
    Y $ 6-8;       /*column input*/
  INPUT X 4. +1
    Y $ 3.;        /*formatted input*/
  INPUT X=
    Y= $ ;         /*named input*/
  INPUT ;          /*null input*/
  CARDS;
14.4 Bob
1492 Sue
1776 Ann
X=28.8 Y=Jay
dummy - ignored by null INPUT
;
```

Figure "H" - INPUT Statement

There are a number of mechanisms to control the column pointer when reading an input file. Defining the start column after each variable name when using the column input method will move the pointer to that column. Similarly, the use of a format after a variable name will move the column pointer.

There are two symbols that move the column pointer within the current record. The Plus sign (+) will move the column pointer relative to its current position. For example, +6 will move the pointer six characters from its current position. It should be noted that the value passed to the relative column pointer (Plus sign) need not be positive. A negative number will move the pointer *backwards* from its current position, up to the first position of the file. Negative numbers must be enclosed in parentheses when using the Plus sign to reposition the column

pointer. Similar to the Plus sign, the At sign (@) provides absolute column position. For example, @6 will move the pointer to the 6th position within the current record. Also note that it is not necessary to hard-code a fixed number for the Plus and At signs; they can be followed by a SAS variable to provide additional flexibility.

It is also possible to use a text string in conjunction the At sign. As an example, @"the" will move the pointer to the first position following the next occurrence of the string "the" within the current record. It should be noted that the string must be typed in *exactly* as desired. The case - upper vs. lower for alphabetic characters is significant, and blanks are considered significant characters for both the text string and the value read. Similar to the numeric At, a character variable can be used to provide greater flexibility. See Figure I for examples of @"string" usage.

```
DATA _NULL_;
  INFILE CARDS;
  INPUT # 1 @"the" NEXT8_1 $CHAR8.
    /* reads "me of th" */
    # 1 @"the " NEXT8_2 $CHAR8.
    /* reads "Thebes t" */
    # 1 @"The" NEXT8_3 $CHAR8.
    /* reads " theme o" */
    # 1 @"The " NEXT8_4 $CHAR8.;
    /* reads "theme of" */
  CARDS;
The theme of the Thebes theater's ...
;
```

Figure "I" - Variable Length File

It is also possible to control the line pointer. The Pound sign (#) will move the line pointer to a specified line number within the input buffer. For example, #4 will move to the 4th line of the input buffer. (By default, the input buffer will contain the maximum number of lines specified by using the Pound sign in an INPUT statement. This can be overridden by the N= option of the INFILE statement.) The At sign (@) can also be used to control the line pointer. The default action is to move to a new input line with each INPUT statement; however, by ending an INPUT statement with "@", the line pointer will remain on the *same* input line. (This is referred to as a "Trailing 'At' sign".) The line pointer will not be moved until it encounters an INPUT statement without a Trailing At sign -- this is the most common reason for a null input statement -- or the next iteration of the DATA step. The latter control can be overcome if necessary by using a Double Trailing At (@@).

There are some special considerations which should be undertaken when reading records from variable length files. The LENGTH= option on the INFILE

statement will assign the length of the current record to a SAS variable. A null input statement with a Trailing @ will permit that variable to be assigned, while keeping the line pointer on the current input line. Finally, the \$VARYING. format will allow for flexibility in length of character variables. See **Figure J** for an examples of this approach.

```
DATA EXAMPLE;
  INFILE varyfil LENGTH=linelen ;
  INPUT @ ;
  INPUT NAME $VARYING40. linelen;
RUN;
```

Figure "J" - Variable Length File

CSV (Comma Separated Value) Files

The CSV, or *Comma Separated Value* File is a special variety of sequential file, typically used for importing or exporting data from a spreadsheet. Data values are separated by commas, as is implied by the name, and character values are typically surrounded by double quotation marks (").

CSV files can be processed by using the DSD parameter on the INFILE statement. This parameter automatically sets the default delimiter to comma, although this can be overridden by use of the DELIMITER= option. The presence of a pair of commas denotes a missing value. The DSD parameter also causes SAS to strip the double quotation marks, if present, from character values before storing them in SAS variables. Please note that character variables are defined with a default length of 8 bytes in this instance. This default length can be overridden by use of the **LENGTH** statement. Do not attempt to specify a format length on the input statement for character variables, as this may cause delimiting commas to be treated as part of the variable's value. See **Figure K** for an example of reading a CSV file.

```
DATA TEMP;
  LENGTH CITY $ 20. STATE $ 15. ;
  INFILE SAMPCSV DSD ;
  INPUT YEAR CONFNAME $
        CITY $ STATE $;
RUN;
```

Figure "K" - CSV File

Informats

A sequential file may contain several types of data. The reader can opt to input this data as ordinary numeric or character data, then programatically transform it according to its characteristics. However, in many cases, this task can be streamlined by reading the data using an *informat*. Informats provide instruction for the reading of data into a SAS variable. They specify whether an input

value is character or numeric, its length, the number of decimal places (if applicable), and any special conditions.

There are two methods to specify an informat for a variable. The first is to specify the informat along with the variable on the INPUT statement. (**Figures I and J, above, used this approach.**) The other method is to use either the **INFORMAT** or **ATTRIB** statement. Either of these statements can be specified in the DATA step to permanently assign an informat to variables within a SAS dataset. The syntax for each of these statements is:

INFORMAT variables <informat>

<DEFAULT=default informat>;

ATTRIB variables INFORMAT=informat;

(Please note that the DEFAULT= informat specified on the INFORMAT statement is only valid for the DATA step in which it was coded.) In the event both methods are used in a DATA step, the informat specified on the INPUT statement will take precedence over one specified via either an INFORMAT or ATTRIB statement.

There are five categories of informats available in the SAS System : *numeric, character, date/time, column-binary, and user-defined.* Lists and descriptions of the first four categories can be found in the *SAS Language: Reference* manual. The fifth category, user-defined, is a catch-all grouping; the SAS System provides a utility to create and use informats that may be unique to the needs of the individual. The details of the process to create user-defined informats is beyond the scope of this presentation; the interested reader is encouraged to refer to the section on **PROC FORMAT** in the *SAS Procedures Guide*.

SAS Data Views

Until this point, we have been treating a SAS data set as the automatic output of a SAS DATA step. This is actually no longer true. There are *two* file structures that can be created by the DATA step : the traditional SAS data set, and the relatively newer *SAS data view*.

The SAS data view does not contain actual data; rather, it contains a description of data which may be stored in external databases, sequential files, or even other SAS data sets. There are three types of SAS data views. SAS/ACCESS views and SQL views are beyond the scope of this presentation (although an example of an SQL view is provided) and will not be discussed at this time. The third, the DATA step view, is structured very similarly to the traditional SAS DATA step. The only difference in coding is that the option **VIEW=viewname** must be coded on the DATA statement, separated from the rest of the statement by a slash (/). Please note

that the view name must be the same as the traditional dataset name as coded on the DATA statement.

There are several advantages to using a SAS data view over a traditional DATA step. The primary advantage is that data is not stored within a SAS data set. This eliminates redundant data storage and ensures that the routine always uses the most current version of the data. A second advantage from a security standpoint is that the source code is

stored within a SAS data library; it is not echoed back to the SAS log when actually invoked and it is not accessible to the end user. (This may also be a disadvantage for those who like their SAS logs to be a complete history of their actions - or if the original source code for the data view is lost.) **See Figure L for an example of the creation and use of a SAS DATA step view (along with an SQL view).**

```
93  FILENAME loctxt 'c:\sasconf\confloc.txt';
94
95  /*  Define a DATA Step View.  */
96  DATA LOC / VIEW=LOC ;
97      INFILE loctxt;
98      INPUT @ 1 CONFNAME $CHAR8.
99             @ 10 CONFYEAR 4.
100             @ 15 CONF_CITY $CHAR20.
101             @ 36 CONFST  $CHAR15.;
102  RUN;

NOTE: DATA STEP view saved on file WORK.LOC.
NOTE: The original source statements cannot be retrieved from a stored DATA STEP view
nor will a stored DATA STEP view run under a different release of the SAS system
or under a different operating system.
Please be sure to save the source statements for this DATA STEP view.
NOTE: The DATA statement used 0.66 seconds.

103
104  /*  Use the DATA Step View  */
105  /*  to define an SQL View.  */
106  PROC SQL ;
107      CREATE VIEW SAMPSQL AS
108      SELECT CONFYEAR, CONF_CITY, CONFST
109      FROM loc
110      WHERE CONFNAME='SUGI';
NOTE: SQL view WORK.SAMPSQL has been defined.
111  RUN;
NOTE: PROC SQL statements are executed immediately; The RUN statement has no effect.
112
113  /*  Use the SQL View, which  */
114  /*  uses the DATA Step View  */
NOTE: The PROCEDURE SQL used 0.6 seconds.

115  PROC PRINT DATA=SAMPSQL; RUN;

NOTE: The infile LOCTXT is:
      FILENAME=c:\sasconf\confloc.txt,
      RECFM=V, LRECL=256

NOTE: 12 records were read from the infile LOCTXT.
      The minimum record length was 37.
      The maximum record length was 47.
NOTE: The view WORK.LOC.VIEW used 0.77 seconds.

NOTE: The PROCEDURE PRINT used 1.14 seconds.
```

Figure "L" - SAS Data View, Creation and Use

TRANSPORT FILES

There is one potential source of data that might not be thought of as an "external source" at first glance - the SAS System itself, licensed on another computer. However, it is not possible to simply move SAS datasets from one operating system to another. Instead, the data must be converted to a format that is consistent on both machines. The "brute force" method of accomplishing this task would be to use PROC PRINT or the PUT statement to output the contents of the SAS dataset to a sequential file on the first machine, then to input that file into the SAS System on the second machine using the techniques described previously in this paper. However, an easier method exists; the SAS System permits the transfer of SAS datasets across operating systems as *SAS Transport Files*.

Transport files are used to move one or more SAS data sets from one host system to another. The transport file is a sequential file which is independent of the host operating system. This file can be readily transferred electronically or on permanent media to the destination host system. There are three basic steps involved:

- *Export* - creating the transport file on the original host system,
- *Transport* - moving the file via network protocols, tapes, or floppy media, and
- *Import* - reading the file back into SAS with the format of the destination host system.

Due to space limitations, only the procedure for moving a single data file will be discussed here. Further documentation on moving entire data libraries or catalogs can be found in the assorted Operating System Companion manuals and in *SAS Technical Report P-195*.

Export : Creating the Transport File

The first step in the transport process is to *export*, or create the SAS transfer data set on the host system. Typically, this process starts by the allocation of the transport file on the host system. The SAS System has strict requirements on the allocation of a transport file. A SAS transport file **must** have a fixed record length of 80. In addition, it is highly recommended that it have a block size of 8000. (Note that block size is a meaningless concept under OS/2.) **See Figure Z for examples of tape allocations.**

```
Under VMS:
DEFINE TRANFILE REEL
ALLOCATE TRANFILE
MOUNT/FOREIGN/BLOCKSIZE=8000 TRANFILE
```

```
Under MVS:
//TRANFILE DD DSN=mvs.data.set.name,
//  DISP=(NEW,CATLG,DELETE),
//  UNIT=TAPE,VOL=SER=volser,LABEL=(1,NL),
//  DCB=(RECFM=FB,LRECL=80,
//      BLKSIZE=8000,DEN=density)
```

Figure "Z" - Allocating a tape

Two LIBNAME statements are required. One LIBNAME statement will identify the location of the file to be transported. The other LIBNAME statement specifies the name that will be given to the transport file, and defines the XPORT engine to identify the destination file as a transport file. **See Figure AA for examples of LIBNAME statements.**

```
Under VMS:
LIBNAME outxp XPORT
' [directory]filename.dat';
LIBNAME outxp XPORT;
LIBNAME libref b '[directory]';
```

```
Under MVS:
LIBNAME ddname XPORT;
LIBNAME alibref 'data.set.name.';
```

Figure "AA" - Allocating a tape

At this point, PROC COPY is used to actually create the SAS transport file. PROC COPY will read in the host system formatted file that is named in the SELECT statement, and create the transport file. **See Figure BB for an example of PROC COPY.**

```
PROC COPY IN=sasdata OUT=xprtdata;
SELECT sas-datasetname;
RUN;
```

Figure "BB" - PROC COPY

Transport : Moving the Transport File

The middle step in the transport process is to actually *transport* the SAS transfer data set on from the original host system to the new operating system. In the early days of the SAS System, this usually required the allocation of a round tape which could be then mounted on another machine. This process may still involve the physical transfer of a tape or floppy disk between two machines. However, the technology of today permits easy electronic data transfer between different machines and operating systems. Note that SAS transport files should be treated as BINARY data when using network commands such as ftp.

Import : Reading the Transport File

The final step in the transport process is to *import* the SAS transfer data set into the SAS System on the destination machine. This also involves the use of LIBNAME statements and PROC COPY. This time, however, the process is reversed; the LIBNAME for the transport file is used for the IN= parameter of PROC COPY, while the LIBNAME of the SAS dataset is used for the OUT= parameter. Again, a SELECT statement appears in the PROC COPY procedure. Figure 2 shows the syntax for importing the transport file on the new host system. (Due to space limitations, there is no example of using PROC COPY to import a transport file. The process is almost identical to the transport process described earlier in this section.)

It is also possible to use PROC CPORT and PROC CIMPORT to transfer SAS datasets between different machines and operating systems. However, this topic will not be explored in detail due to space limitations.

CONCLUSION

There are a number of methods to introduce external data into the SAS System. It would be impossible to provide in-depth information on all of them in the limited space of this presentation, in fact a ½ day class could not cover them all! It is hoped that the material contained in this paper will serve to stimulate the curiosity of the reader, and that they will continue their education by researching the appropriate manuals and technical papers devoted to the specific topics discussed within this paper. Ultimately, however, it will be through real-life trial and error that true comprehension and retention of this knowledge will be attained.

REFERENCES / FOR FURTHER INFORMATION

Beatrous, Steve, and Clifford, Billy. (1998). "Sometimes You Get What You Want: I/O Enhancements for Version 7". *Proceedings of the Twenty-Third Annual SAS Users Group International Conference*. Cary, NC: SAS Institute, Inc.

Boling, John C. (1997). "SAS Data Views: A Virtual View of Data". *Proceedings of the Twenty-Second Annual SAS Users Group International Conference*. Cary, NC: SAS Institute, Inc.

Carey, Helen and Carey, Ginger (1996). *SAS Today! A Year of Terrific Tips*. Cary, NC: SAS Institute, Inc.

Cody, Ronald (1998). "The INPUT Statement: Where It's @". *Proceedings of the Twenty-Third Annual SAS Users Group International Conference*. Cary, NC: SAS Institute, Inc.

Dickson, Alan, and Pass, Ray (1996). "SELECT ITEMS FROM PROC.SQL Where ITEMS > BASICS". *Proceedings of the Twenty-First Annual SAS Users Group International Conference*. Cary, NC: SAS Institute, Inc.

Heffner, William F. (1998). "DATA Step in Version 7: What's New?". *Proceedings of the Twenty-Third Annual SAS Users Group International Conference*. Cary, NC: SAS Institute, Inc.

Kuligowski, Andrew T., and Roberts, Nancy (1997). "From There to Here: Getting Your Data Into the SAS System". *Proceedings of the Twenty-Second Annual SAS Users Group International Conference*. Cary, NC: SAS Institute, Inc.

Levine, Allison. (1997). "The What, When, Why, and How of PROC FORMAT". *Proceedings of the Tenth Annual NorthEast SAS Users Group Conference*. USA.

Mason, Phil. (1996). *In the Know ... SAS Tips and Techniques from Around the Globe..* Cary, NC: SAS Institute, Inc.

Riba, S. David (1996), *Course Notes: Connecting With Your Data*. Clearwater, FL: JADE Tech, Inc.

SAS Institute, Inc. (1993), *SAS Companion for the Microsoft Windows Environment*. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1990), *SAS Language: Reference, Version 6, First Edition*. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1996). *SAS Online Documentation*. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1990). *SAS Procedures Guide, Version 6, Third Edition*. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1994), *SAS Software: Abridged Reference, Version 6, First Edition*. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1989). *SAS Technical Report P-195, Transporting SAS Files between Host Systems*. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1991). *SAS Technical Report P-222, Changes and Enhancements to Base SAS Software, Release 6.07*. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1997). *Window by Window: Capture Your Data Using the SAS System*. Cary, NC: SAS Institute, Inc.

SAS is a registered trademark or trademark of SAS Institute, Inc. in the USA and other countries. MVS, JCL, DB2, and Lotus are registered trademarks or trademarks of International Business Machines Corporation. ORACLE is a registered trademark or trademark of Oracle Corporation.

® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

The author can be contacted via e-mail as follows:
A_Kuligowski@msn.com

ACKNOWLEDGMENTS

I would like to acknowledge and thank Ms. Nancy Roberts. Nancy was the co-author of the ancestor of this paper, *From There to Here: Getting Your Data Into the SAS® System*, which can be found in the SUGI 22 Proceedings. Some portions of this presentation can be traced back to Nancy's work, and are used with her permission.