The Good, The Bad, and The Ugly

Toby Dunn, AMEDDC&S (CASS), San Antonio, Texas
Kirk Paul Lafler, Software Intelligence Corporation, Spring Valley, California

Abstract

The SAS® System has all the tools users need to read data from a variety of external sources. This has been, perhaps, one of the most important and powerful features since its introduction in the mid-1970s. The cornerstone of this power begins with the INFILE and INPUT statements, the use of a single- and double-trailing @ sign, and the ability to read data using a predictable form or pattern. This paper will provide insights into the INFILE statement, the various styles of INPUT statements, and provide numerous examples of how data can be read into SAS with the DATA step.

Introduction

For many SAS users, the process of reading unprocessed "raw" data into a SAS data set is not a new one. For others, it involves a daunting and confusing maze of complicated syntax best left to experienced data analysts, programmers, and users. Often, certain attributes are associated with raw data, including data that is not validated, error prone, and suspect. But, one description about raw data that is generally recognized by most, if not all, SAS users is that it represents unprocessed data that has never been read into the SAS System.

The purpose of this paper is to cut through much of the mumble-jumble and complexities often associated with reading raw data. Our goal is to explain, in simple terms, the process of reading in-stream and external raw data files. Numerous examples will be illustrated and discussed along the way to improve the level of understanding associated with the syntax for successfully turning raw data into meaningful and actionable SAS data sets that all can use.

The Process of Turning Raw Data into Information

Raw data is represented as an unprocessed form of data that has never been manipulated in any way, shape, or form. SAS users refer to raw data as unprocessed non-SAS data that has not been subjected to any type of processing, analysis, or manipulation. In the SAS System, raw data is traditionally read into a SAS data set using the INFILE, DATALINES (or CARDS), and INPUT statements, as well as with SAS functions, and, if your specific operating system supports a graphical user interface, with the Import Wizard or External File Interface (EFI).

An important question, and one this paper attempts to answer, is how raw data is transformed into information? SAS users, including data analysts, will tell you that raw data is transformed into actionable and meaningful information by organizing, analyzing, and formatting data using a combination of facts and metrics. Lisa M. Loftis of Intelligent Solutions, Inc. claims that valuable information exhibit certain characteristics including usefulness, where it is matched to the needs of those who receive it, and availability, where it is received at a time and in a context to enable action.

As defined in BusinessDictionary.com, "information is data that has been verified to be accurate and timely, specific and organized for a purpose, presented within a context that gives it meaning and relevance, and can lead to an increase in understanding and decrease in uncertainty." Ultimately, the value attributed to the transformation of raw data into information is in its ability to affect a behavior, decision, or outcome.

The Power of the INFILE Statement

The INFILE statement identifies an external input data file (or in-stream data) and provides essential information describing the source of the data file to the DATA step. When specified, it opens the external data file and creates an input buffer to hold the input data. More than one INFILE statement can be specified, depending on the number of "input" data files needing to be read. The general syntax of the INFILE statement is:

```
INFILE  file-specification  <option(s)> ;
```

The file-specification designation represents the actual input file name, fileref, alias, or ddname. It directly or indirectly identifies the name of the external input data file or in-stream data. A number of options are available to the INFILE statement for describing the input data and how the data is to be handled. An alphabetical list of INFILE statement options appears below, see Figure 1.

Option	Description
COLUMN=variable	Creates a variable that is used to assign the current column location of the input pointer.
DATALINES (or CARDS)	Informs SAS that the input data resides inside the code (in-stream data) and immediately follows the DATALINES (or CARDS) statement. Note: CARDS is an older connotation referencing data residing on 80-column punch cards, known as Hollerith cards. The DATALINES and CARDS options are typically specified when illustrating a concept in an academic setting, because most input data files come from some external source.
DELIMITER=delimiter(s)	Identifies one or more alternative non-blank delimiter(s) for data values when using LIST-style of Input. Note: A list of characters must be enclosed in quotes.
DSD	Refers to delimiter-sensitive data and is used with List-style of Input. Informs SAS to set the default delimiter to a comma, remove quotation marks from character values, and assign a missing value to two consecutive delimiters.
END=variable	Refers to a user-defined variable that SAS uses to indicate whether the current input data record is the last in the input file. SAS sets the user-defined variable to 1 if the current input data record is the last in the input file, otherwise it is set to 0. **Note:** The END= option cannot be used when the DATALINES (or CARDS) statement is specified.
EOF=label	Tells SAS to perform an implicit GO TO using the user-supplied label when an INPUT statement attempts to read from an input file that has no more input data records.
FILEVAR=variable	When the value in the FILEVAR=variable changes, SAS automatically closes the current input data file and opens the new input data file specified in the FILEVAR=variable.
FLOWOVER	As the default behavior of the INPUT statement, the FLOWOVER option tells SAS to continue reading the next input data record when the number of values in the current input line does not match the number of variables specified in the INPUT statement. Note: The behavior of the default FLOWOVER option is the exact opposite of the MISSOVER option (see below).
LINE=variable	SAS set the user-defined variable to the line location of the input buffer's input pointer. The value range of the LINE=variable is 1 to the value of the N=option. **Note:** The LINE=variable is not written to the output SAS data set.
MISSOVER	The MISSOVER option prevents SAS from reading the next input data record when the number of values in the current input line does not match the number of variables specified in the INPUT statement. Variables without any values are automatically assigned missing values. **Note:** The behavior of the MISSOVER option is the exact opposite of the FLOWOVER option (see above).
N=available-lines	Refers to the number of lines that are available to the input pointer at one time.
STOPOVER	The STOPOVER option tells the DATA step to stop processing, set the _ERROR_ system variable to 1, stops writing to the SAS data set, and prints the incomplete data line to the SAS Log when an input line does not contain the expected number of values as specified in the INPUT statement.
TRUNCOVER	The TRUNCOVER option permits SAS to read variable-length records when some records are shorter than what the INPUT statement expects. **Note:** The TRUNCOVER option overrides the default behavior of the FLOWOVER option (see above).
INFILE=variable	A character variable is created to reference the contents of the current input buffer for an INFILE statement. **Note:** The _INFILE_=variable is not written to the SAS data set.

Figure 1. INFILE Statement Options

Reading Data with the INPUT Statement

An INPUT statement is used to tell SAS the unique arrangement of the input data records and the definition of input values associated with the corresponding user-defined SAS variables being read from a raw input data file. An INPUT statement is used for reading raw data from in-stream data files and from external data files. The general syntax of the INPUT statement is:

INPUT <user-specifications> <@ | @@> ;

Styles of INPUT Statements

Four styles of INPUT statement are available to users.

1. ***List Style***

Identifies each variable in the order they appear in the input data file. One or more blanks must separate each field in the input "raw" data file. A dollar sign ($) is specified after the variable name to indicate character data. The dollar sign ($) is omitted for numeric data. Blank fields (missing values) cause the one-to-one matching of data values and variable names specified in the INPUT statement to become out-of-sync. No imbedded blanks are permissible in a data value, (e.g., Silence of the Lambs). The default length of character variables is 8 positions (bytes). The general syntax appears as:

INPUT variable-name {$} ...;

2. ***Column Style***

Tells the SAS System where to find the desired data value in the input data record. It specifies the starting and ending positions of the data value. A dollar sign ($) is specified after the variable name to indicate character data. The dollar sign ($) is omitted for numeric data. Data values can be read in any order. Parts of a data value can be reread, resulting in one or more variables being created. Embedded blanks are permissible. Blank data values are treated as missing values. The length of the variable is derived from the designated starting and ending positions. The general syntax appears as:

INPUT variable-name {$} start-col end-col ...;

3. ***Formatted Style***

Tells the SAS System where to find the desired data value in the input data record. It specifies the starting position and the length of the data value. A dollar sign ($) is specified after the variable name to indicate character data. The dollar sign ($) is omitted for numeric data. Data values can be read in any order desired. Parts of a data value can be reread, resulting in one or more variables being created. Embedded blanks are permissible. Blank data values are treated as missing values. The ending position is derived from the starting position and length of the data value. The general syntax appears as:

INPUT pointer-control variable-name {$} length ...;

Available Pointer Controls include:

Pointer	Description
@n	Tells SAS to start reading a data value in column n.
+n	Positions the input pointer n positions to the right of the current location.
/	Positions the input pointer down one row and places cursor in column 1.
#n	Positions the input pointer to record n.

4. ***Named Style***

Tells the SAS System where to find the desired data value in the input data record. It specifies the starting position and the length of the data value. A dollar sign ($) is specified after the variable name to indicate character data. The dollar sign ($) is omitted for numeric data. Data values can be read in any order desired. Data values containing a variable name followed by an equal sign and a value are read. Embedded blanks are permissible. Blank data values are treated as missing values. The ending position is derived from the starting position and length of the data value. The general syntax appears as:

INPUT <pointer-control> variable-name= {$} length <@ | @@> ...;

INPUT <pointer-control> variable-name= <informat-name.> <@ | @@> ...;

INPUT variable-name= {$} start-column <end-column> <@ | @@> ...;

Reading ASCII or EBCDIC Data

Tab-delimited input contains tab characters separating each input record's data value. Unless you're using an ASCII or EBCDIC editor, tab characters are generally invisible. Tabs are represented in ASCII as '09'x and in EBCDIC as '05'x. To allow SAS to read this type of input under Windows and Unix operating environments, you'll want to specify the DSD and DLM='09'x INFILE statement options, as follows. The input data file can then be successfully read with SAS using each embedded tab character as separators between data values.

SAS Code for Windows and Unix Operating Environments

```
DATA MOVIES;
  INFILE CARDS DSD DLM='09'x MISSOVER;
  INPUT TITLE $ RATING $;
  DATALINES;
Jaws PG
Rocky,II PG
Titanic PG-13
;
RUN;
```

To read an input data file with embedded tab characters created in a mainframe operating environment, you'll want to specify the DSD and DLM='05'x INFILE statement options, as the following code illustrates.

SAS Code for Mainframe Operating Environments

```
DATA MOVIES;
  INFILE CARDS DSD DLM='05'x MISSOVER;
  INPUT TITLE $ RATING $;
  DATALINES;
Jaws PG
Rocky,II PG
Titanic PG-13
;
RUN;
```

Techniques on Handling "Bad" Data

When input specifications don't match one or more of the data values in an input data file, the result is often "bad" or missing data. This is an all too familiar scenario for SAS users, and one that some users may be seeking an answer to. The following example illustrates a useful technique that can help put you in control of what to do when "bad" data occurs.

SAS Code

```
DATA MOVIES;
  INFILE CARDS STOPOVER;
  INPUT TITLE : $11. RATING $;
  DATALINES;
Jaws PG
Rocky,II PG
Titanic
;
RUN;
```

By specifying the INFILE statement STOPOVER option, the propagation of "bad" data or missing values can be a thing of the past. When an input line doesn't contain the expected number of values as defined on the INPUT statement, SAS sets the _ERROR_ system variable to 1, stops processing the DATA step, and prints the offending data line to the SAS Log. The error message associated with running this code shows that the INPUT statement exceeded the record length, as illustrated below.

SAS Log

```
102  DATA MOVIES;
103    INFILE CARDS STOPOVER;
104    INPUT TITLE : $11. RATING $;
105    CARDS;

ERROR: INPUT statement exceeded record length. INFILE CARDS OPTION STOPOVER specified.
RULE:       ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+
108         Titanic
TITLE=Titanic RATING=  _ERROR_=1 _N_=3
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.MOVIES may be incomplete.  When this step was stopped there were 2
         observations and 2 variables.
WARNING: Data set WORK.MOVIES was not replaced because this step was stopped.
NOTE: DATA statement used (Total process time):
      real time           0.07 seconds
      cpu time            0.00 seconds

109  ;

110  RUN;
```

Reading "Free-form" Text Files

In this example we'll examine a scenario where data residing in the input data file contain three variables defined as "free-form" text files. By combing the SAS-L archives and LexJansen.com, we found a very interesting technique for handling "free-form" data, as follows.

SAS Code

```
Data Test ;
  Array Var[ 3 ] ;
  Do _N_ = 1 To 3 ;
     Input @1 @( CatS( 'var' , _N_ , ':' ) ) Var[ _N_ ] @ ;
  End ;
  Datalines ;
var1:11 var2:12 var3:13
var3:23 var2:22 var1:21
var3:33 var1:31 var2:32
;
Run ;
```

The resulting SAS data set from processing the preceding code appears below.

Obs	Var1	Var2	Var3
1	11	12	13
2	21	22	23
3	31	32	33

By inspecting the raw data in the DATALINES in-stream data, you will notice that the each data element is prefixed with the variable name it reads with an ':'. We exploit this feature to our benefit by specifying the @(text) feature that tells SAS where to start reading data for a variable. Since we have three variables to read data for, we wrap this into a DO-loop and build the INPUT statement on the fly.

By dissecting the INPUT statement, Input @**1** @(CatS('var' , _N_ , ':')) Var[_N_] @ ;, we see the CatS('var' , _N_ , ':') builds the text that denotes when to start reading data for a variable. The _N_ = 1 creates var1 and _N_ = 2 creates var2. The second part Var[_N_] @ creates the variable name. Since the text to start reading data is the same text that makes up the variable name this makes the task easier. Lastly, the INPUT statement builds one variable at a time using the DO-Loop, while the trailing-@ is there to ensure that SAS holds the line until it reads in all three values.

Handling Missing Values

Traditionally we use the INFILE and INPUT statements for reading data either from an external file or in-stream data using a DATALINES (or CARDS) statement. However, it can, if you want, be used slightly differently to solve some other problems. In this example the poster wanted to read and parse a text string, "aa*bb**dd*ee****ii". Where the * are the delimiters between the text. Consecutive delimiters need to be treated as missing.

SAS Code

```
Data Missing_Data ;
  Str = "aa*bb**dd*ee****ii" ;
  Infile Cards DSD DLM = '*' MissOver ;
  Input @ ;
  _Infile_ = Str ;
  Input @1 ( S1-S10 )( :$8. ) ;
  Output ;
  Put (S1-S10)(=) ;
  Datalines ;
Necessary Evil
;
Run ;
```

Variables to be Parsed

```
S1=aa S2=bb S3= S4=dd S5=ee S6= S7= S8= S9=ii S10=
```

Here we see the text to be parsed in the variable Str. An Infile statement is then specified to process the in-stream raw data stream. This may seem odd when there is no data from the in-stream data stream, but it is specified to set the so we can use the Input statement to perform the parsing for us. Next the Input @ is used to create an _Infile_ buffer that is loaded with the data from variable Str. Once this is done, the Input statement is specified to parse the text into individual variables. You will notice the "Necessary Evil" after the cards statement. While it doesn't necessarily have to be this, something has to be there even if it isn't read into the data set so SAS actually creates the _Infile_ input buffer. This example is pretty slick in that it uses SAS's ability to read and parse data into a data set even when the data doesn't come from a traditional source.

In this next example, a person needed to read a "blob" of data into SAS while breaking it into its data element parts. A "blob" is a data construct that contains everything thrown together. What makes this particularly interesting is that SAS was used to grab the "blob" of data, by reading it, and creating a temporary SAS data set.

Blob Data Set

```
Data Blob ;
  Input ;
  Blob = _Infile_ ;
  Datalines ;
field1 10/10/2010
field2 10/10/2010
field3 10/10/2010
field4 10/10/2010
;
Run ;
```

SAS Code

```
Data Need ;
  Format Date Date9. ;
  Infile Cards ;
  Input @ ;
  Do Until( EOF ) ;
     Set Blob End = EOF ;
     _Infile_ = Blob ;
     Input @1 Field $6. @8 Date MMDDYY10. @ ;
     Output ;
  End ;
  Datalines ;
Necessary Evil
;
Run ;
```

This example is almost exactly like the previous example in that it's "faking" the DATALINES statement to obtain an _INFILE_ buffer so SAS can parse it. The difference between the two examples is the current example uses a SET statement to bring data from a SAS data set for parsing purposes. Since the data being parsed is coming from a SAS data set, the set and input statement need to be wrapped in a DO-loop. Specifically, a DO-Until loop which iterates through the DO-loop until the Blob data set has run out of data.

While the SET statement reads observations, the variable Blob has its values copied to the _INFILE_ buffer. The INPUT statement then parses and reads the data. The important thing to remember is that by forcing the DATALINES statement to use an INFILE statement, it allows the DATA step to have access to the _INFILE_ and INPUT statements.

Reading Delimited Files

A question came into the Listserve (SAS-L) not too long ago that dealt with an input data file that contained pipes (vertical bars) as the delimiter, as illustrated in the example data below.

Input Data File

```
apple|red|1
pear|green|2
banana|yellow|3
badapple||purple|4
```

The last observation has a double pipe "||" which indicated that the first of the two pipes should be included in the field. IF one simply uses the DLM option with a "|" symbol, then the DATA Step wouldn't be able to read it correctly, as shown below.

SAS Code

```
 Data Delimited_Data ;
    Length Fruit $ 10 Color $ 10 ;
    Infile Cards DLM = '|' TruncOver DSD ;
    Input Fruit Color Count ;
 Cards ;
```

The corresponding SAS Log appears below.

SAS Log Results

```
NOTE: Invalid data for Count in line 1658 11-16.
RULE: ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8---
      badapple||purple|4
Fruit=badapple Color=  Count=. _ERROR_=1 _N_=4
NOTE: The data set WORK.DELIMITED_DATA has 4 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.29 seconds
      cpu time            0.03 seconds
```

One SAS-L poster came up with an ingenious solution to this intriguing problem, as is shown below.

```
Data Delimited_Data ;
  Length Fruit $ 10 Color $ 10 ;
  Infile Cards DLM = '|' TruncOver DSD ;
  Input @ ;
  _Infile_ = TranWrd( _Infile_ , '||' , '~|' ) ;
  Input Fruit Color Count ;
  Array _C[ * ] _Char_ ;

  Do _N_ = 1 To Dim( _c ) ;
     _C[ _N_ ] = Translate( _C[ _N_ ] , '|' , '~' ) ;
  End ;
  Cards ;
Run;
```

The results are displayed in the SAS Log below.

```
NOTE: The data set WORK.Delimited_Data has 4 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

This solution first reads the input data record and holds the input line using the trailing @-sign. It then proceeds to use a little _Infile_ magic by converting any double-pipes into '~|' . This way SAS reads the data correctly. Using an array and translate function, it then converts the '~' back into a single |. The ability to change the input buffer with the _Infile_ is an important technique when your data is ugly. **Note:** You may need to do some preprocessing before your input statement reads the input data record.

Concatenating Multiple, but "Similar" Input Data Files

A typical problem SAS users frequently experience is one where multiple, but "similar" input data files having the same data structures need to be read. Although some users may read each input data file separately (one-by-one), the result is typically less than ideal, as illustrated in the code that follows.

SAS Code

```
Data Multiple_Files ;
  Length File $ 40. ;
  File = "C:\Users\sas\Desktop\Paper\Male.csv" ;
  Output ;
  File = "C:\Users\sas\Desktop\Paper\Female.csv" ;
  Output ;
Run ;
```

To streamline the preceding code you'll first want to capture all the directory and input data file names needing to be read in. The first DATA step has two observations with the path and file names combined in one text string. This is then fed to the second data step that uses this information in the FileVar= statement, as shown in the following code.

```
Data Concatenated_Files ;
  Length Name $ 20
         Sex  $ 1 ;
  Informat Height Weight 8.2 ;
  Set Multiple_Files ;
  Infile Dummy FileVar = File DLM = ',' End = EOF ;

  Do While ( Not EOF ) ;
     Input Name Sex Age Height Weight ;
     Output ;
  End ;
Run ;
```

The FileVar= statement takes a text value which tells SAS to read in the files specified in the text string one after the other. Two interesting notes, one is since it is reading the data one file after the other the files have to have the same structure or you will need to separate and conditionally run the input statements based on which file your reading in at the time. Secondly the INFILE statement uses 'Dummy' in place of the normal filename shortcut. Dummy is a key word that tells SAS to look for the file and path in the value contained in the Filevar= option.

Now, you may be thinking, "What if I have many input data files and don't want to write all those paths out". Well, that's not a problem. The following code will solve this issue.

```
FileName MyDir Pipe "dir /s/b C:\Users\sas\Desktop\Paper\*.csv" ;

Data Files ;
  Length File $ 40 ;
  Infile MyDir ;
  Input File ;
Run ;
```

The preceding code will issue the dir /s/b commands allowing DOS to create the list for you while sending the list to the DATA step. Under a Unix platform, the following code is produced:

FileName MyDir Pipe "ls /Users/sas/Desktop/Paper/*.csv" ;

Reading Data with Data-driven Input Files and the Trailing @

A data file that contains values for controlling the way data is read is referred to as a data-driven input file. Data files of this nature are actually quite common in are found in a number of industries. We'll illustrate the structure and content of a simple data-driven input file using DATA step code to position the input pointer for the execution of a read operation. In the following example, the first INPUT statement obtains the starting column position for the movie title. For each G-rated movie, the second INPUT statement then positions the pointer in the input buffer to execute the read operation. Essentially, the starting column data value serves as the data-driven component that informs SAS how to read another data value. **Note:** Data-driven applications like this are only as reliable as the data itself.

SAS Code

```
DATA MOVIES(DROP=COL);
  INFILE CARDS MISSOVER;
  INPUT COL @20 RATING $ @;
  IF RATING='G' THEN INPUT @COL TITLE : $11.;
  ELSE DELETE;
  DATALINES;
7     Jaws         PG
11        Rocky,II PG
3 Wizard of Oz     G
;
RUN;
```

Reading Hierarchical Data

When data is structured in a hierarchical layout, similar to a parent-child relationship, formatted-style of input with line- and column-pointer controls can be used to read the input data file. In the following example the first INPUT statement reads the type of input record. The input pointer is held on the current record using the trailing @ enabling SUMMARY and DETAIL records to be processed differently within the same DATA step.

SAS Code

```
DATA SUMMARY(KEEP=TYPE QTY)
     DETAIL(KEEP=TYPE TITLE);
  INPUT @1 TYPE $1. @;
  IF TYPE='S' THEN DO;
     INPUT @2 QTY 4.; OUTPUT SUMMARY;
  END;
  ELSE IF TYPE='D' THEN DO;
     INPUT @2 TITLE $11.; OUTPUT DETAIL;
  END;
  DATALINES;
S30002000
DBrave HeartR
S37002001
DTitanic    PG-13
;
RUN;
```

Conclusion

The purpose of this paper has been to cut through much of the mumble-jumble and complexities often associated with reading raw data and describe many of the SAS® System tools users need to read data from a variety of in-stream and external raw data sources. We have illustrated numerous examples to improve the level of understanding associated with the syntax for successfully turning raw data into meaningful and actionable SAS data sets that all can use.

References

Benjamin Jr., William E. (2011). *"The Little Engine That Could: Using EXCEL LIBNAME Engine Options to Enhance Data Transfers between SAS® and Microsoft® Excel Files,"* Owl Computer Consultancy, LLC, Phoenix, Arizona, USA.

BusinessDictionary.com (2012). *A Definition of Information,* http://www.businessdictionary.com/definition/information.html.

Cisternas, Miriam and Ricardo Cisternas (2004). *"Reading and Writing XML files from SAS®,"* Ovation Research Group and MGC Data Services, Calsbad, California, USA.

Cody, Ronald Ed.D. (2004). *"The Input Statement: Where It's @,"* Robert Wood Johnson Medical School, Piscataway, New Jersey, USA.

Cohen, Matthew (2008). *"Reading Difficult Raw Data,"* Wharton Research Data Services.

Dunn, Toby and Kirk Paul Lafler (2012). *"The Good, The Bad, and The Ugly,"* Proceedings of the 2012 South Central SAS Users Group (SCSUG) Conference, AMEDDC&S, San Antonio, Texas, USA and Software Intelligence Corporation, Spring Valley, California, USA.

Dunn, Toby and Kirk Paul Lafler (2012). *"The Good, The Bad, and The Ugly,"* Proceedings of the 2012 SAS Global Forum (SGF) Conference, AMEDDC&S, San Antonio, Texas, USA and Software Intelligence Corporation, Spring Valley, California, USA.

Dunn, Toby and Kirk Paul Lafler (2011). *"How to Read Data into SAS® with the DATA Step,"* AMEDDC&S, San Antonio, Texas, USA and Software Intelligence Corporation, Spring Valley, California, USA.

Eberhardt, Peter (2005). *"The SAS® Data Step: Where Your Input Matters,"* Fernwood Consulting Group Inc., Toronto, Ontario, Canada.

Franklin, David (2011). *"Beyond the Comma Delimited File for Bringing Data into a SAS Dataset,"* TheProgrammersCabin.com, Litchfield, New Hampshire, USA.

Kuligowski, Andrew T. (2008). *"Interactions between the DATA Step and External Files,"* The Nielsen Company, USA.

Kuligowski, Andrew T. (2006). *"DATALINES, Sequential Files, CSV, HTML and More – Using INFILE and INPUT Statements to Introduce External Data into the SAS® System,"* The Nielsen Company, USA.

Lafler, Kirk Paul and Toby Dunn (2012). *"The Good, The Bad, and The Ugly,"* Proceedings of the 2012 MidWest SAS Users Group (MWSUG) Conference, Software Intelligence Corporation, Spring Valley, California, USA and AMEDDC&S, San Antonio, Texas, USA.

Llano, Jaime A. (2006). *"Reading Compressed Text Files Using SAS Software,"* Independent Consultant, Bogota, Colombia.

Loftis, Lisa M. (2008). *"Data to Intelligence: The Business Intelligence Advantage for Financial Services,"* Intelligent Solutions, Inc.

Matlapudi, Anjan and Daniel J. Knapp (2010). *"Challenge! Reading Mainframe Hex Delimited Flat File Where Each Line Has Different Layout,"* Pharmacy Informatics, PerformRx, The Next Generation PBM, Philadelphia, Pennsylvania, California, USA.

McLeod, Kathy and Nancy Mae Bonney (1992). *"Advanced Techniques With the INFILE and INPUT Statements,"* ARC Professional Services Group and Federal Reserve Board, USA.

Medrano, Archie (2000). *"Creating Multiple SAS® Data Sets from an ASCII File,"* Ehnc Inc., San Diego, California, USA.

Osborne, Anastasiya (2010). *"From Unfriendly Text File to SAS® Dataset – Reading Character Strings,"* Farm Service Agency (USDA), Washington, DC, USA.

Mullin, Charley (2011). *"Finding Your Way Through the Wilderness: Moving Data from Text Files to SAS® Data Files,"* SAS Institute Inc., Cary, North Carolina, USA.

Rickards, Clinton S. (1999). *"Reading External Files Using SAS® Software,"* Purdue Pharma L.P., USA.

Ritzow, Kim L. Kolbe (1997). *"Advanced Techniques for Reading Difficult and Unusual Flat Files,"* Systems Seminar Consultants, Kalamazoo, Michigan, USA.

Wooding, Nat (2005). *"Extracting Data from PDF Files,"* Dominion Virginia Power, Richmond, Virginia, USA.

Acknowledgments

The authors extend a special thanks to Kenny Bissett and Lizette Alonzo, SCSUG 2013 Conference Chairs, for accepting our abstract and paper, and the SCSUG Executive Committee, SAS Institute, and Conference Leadership for organizing a great conference!

Trademark Citations

About the Authors

Toby Dunn is a civilian statistician for the U.S. Army, providing data-driven, solution-based decision support for Army Medical Command. While he wears many hats in his position, recently he has taken on the role of "data governor" and is in charge of all the data coming and going out of his group, ensuring data integrity and validity while also taking under consideration security and privacy concerns. He earned a B.S. in Agriculture Business from Sul Ross State University and an M.S. in Agricultural and Applied Economics from Texas Tech University. He is the author or coauthor of several papers that have been presented at SAS user group conferences and SAS Global Forum.

Kirk Paul Lafler is consultant and founder of Software Intelligence Corporation and has been using SAS since 1979. He is a SAS Certified Professional, provider of IT consulting services, trainer to SAS users around the world, and sasCommunity.org emeritus Advisory Board member. As the author of five books including PROC SQL: Beyond the Basics Using SAS, Second Edition (SAS Press 2013), Kirk has written more than five hundred papers and articles, been an Invited speaker and trainer at four hundred-plus SAS International, regional, special-interest, local, and in-house user group conferences and meetings, and is the recipient of 22 "Best" contributed paper, hands-on workshop (HOW), and poster awards.

Comments and suggestions can be sent to:

Toby Dunn
AMEDC&S
E-mail: Toby.Dunn@amedd.army.mil
~~~

Kirk Paul Lafler
Senior Consultant, Application Developer, Data Analyst, Trainer and Author
Software Intelligence Corporation
E-mail: KirkLafler@cs.com
LinkedIn: http://www.linkedin.com/in/KirkPaulLafler
Twitter: @sasNerd
~~~