Paper 159-26

Introducing External Data to the SAS® System – the Interactive Session

Andrew T. Kuligowski – Nielsen Media Research

ABSTRACT / INTRODUCTION

One of the first challenges that a new SAS user encounters is to enter their data into the SAS System. Manual entry is obviously not usually the answer; however, it is often unclear just how to best accomplish this task. This Hands-On Workshop will utilize an interactive approach to illustrate various tools that will bring assorted types of files – sequential, CSV, Excel, and others - into SAS for further processing.

SAS IMPORT WIZARD

Often, the best tool for a job is the simplest one. To illustrate that point, the first topic we will discuss is the *SAS Import Wizard.* Available since Release 6.12 of the SAS System, the SAS Import Wizard is a menu-driven system to define and import external data into the SAS System. It is typical in appearance and in approach to Wizards available in other Windows products, such as Microsoft Excel. The Import Wizard is accessed by selecting **Import Data** from the **File** pulldown menu on the SAS toolbar. (Those who have not yet upgraded to Version 8 will find the selection is called **Import** under Version 6.12.)

The first screen requires the user to specify the type of file to be imported. The first choice is a "standard file format"; a pull-down menu allows various options such as dBASE, LOTUS, Excel, or delimited files. (NOTE: Some of these options are only available if the site has licensed SAS/ACCESS to PC File Formats.) The other choice, "user-defined file format", provides an interface to the *External File Interface.* (The External File Interface, which permits the user to specify the details of an external file via menus, will not be discussed in this presentation.) **See Figure A for a sample of this screen – this and subsequent screen prints will illustrate the IMPORT of an Excel 2000 file.**

The SAS Import Wizard now displays a screen prompting the user to "Select File". The file name can be manually typed in the space provided, or the *Browse* button can be selected in order to search for the file. **See Figure B for a sample of this screen.**

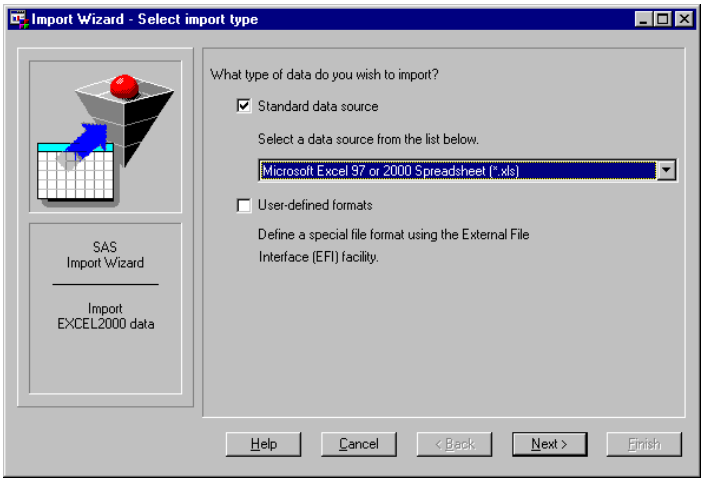

Figure A – SAS Import Wizard "Import Type" Screen

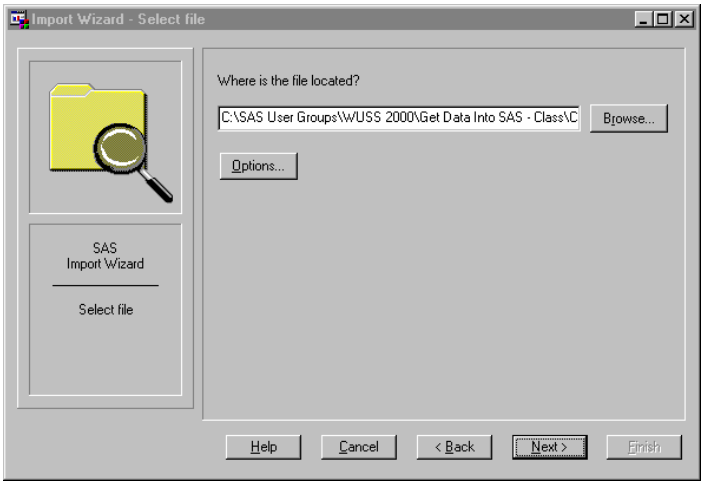

Figure B - SAS Import Wizard "Select File" Screen

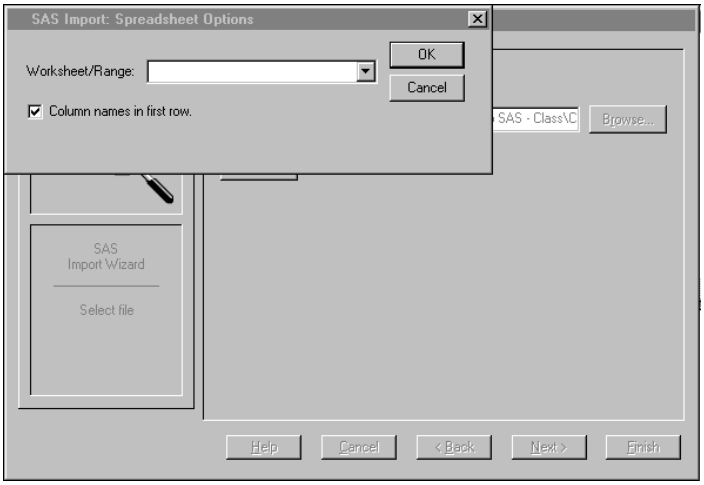

Figure C – SAS Import Wizard "Select File – Options" Screen

This screen also has an Options button. This button causes a separate pop-up window to be displayed, which can be used to select any options that are specific to the type of file selected on the first screen. **See Figure C for a sample of this screen.**

Next, the user is prompted to choose the "SAS Destination", better known as the Library and Member. Again, the user can manually enter the SAS library and member into which the data is to be stored, or they can use pull-down menus to select from those which are already known to the SAS session. **See Figures D and E for samples of this screen.**

If "standard file format" was originally selected and the screen was correctly and completely filled out, please note that the picture on the left side of the screen changes from a tabular image to a checkered flag when entry is finished. Normally, a checkered flag alludes to a finish line or completion – in fact under Version 6.12, this *was* the final screen of the SAS Import Wizard. However, under Version 8, an additional screen allows the user to store the generated code in a file, allowing for reuse and/or modification. **See Figure F for an example of this screen.**

At this point, the SAS System processes the request. Success is indicated by a simple message in the SASLOG, advising that the SAS dataset is now available for use :

```
NOTE: [SASdsn] was successfully
      created
```

At this point, it is possible for those users who are still using Version 6.12 of the SAS System to save their generated code, although the process is slightly more complicated than entering in values on a menu as with Version 8. The process to save generated SAS code under Version 6.12 is as follows:

- Toggle to the PROGRAM window in SAS Display Manager.
- RECALL (F4 by default) your previously submitted code.

The generated code will be echoed in the Program Window, and it can now be saved to an external file.

It should be noted that the code generated under Version 6.12 will look radically different than the code generated under Version 8. This is because Version 8 uses **PROC IMPORT**, which was not available under Version 6.12. **Refer to Figures G and H for examples of code generated under both versions of SAS.**

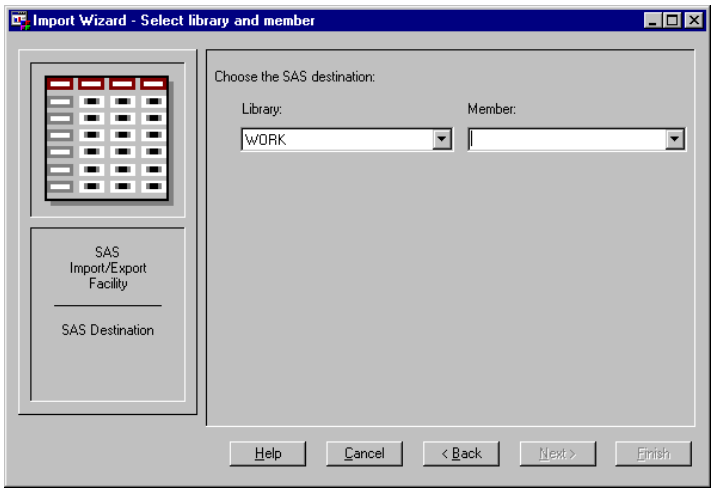

Figure D – SAS Import Wizard "Library & Member" Screen (Initial)

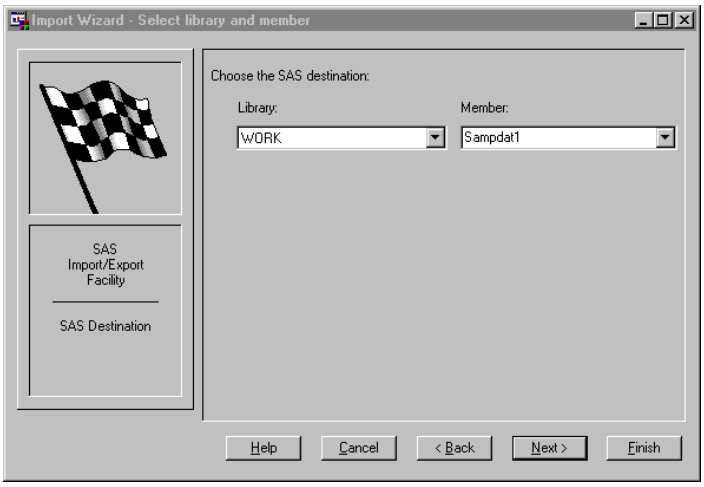

Figure E – SAS Import Wizard "Library & Member" Screen (Completed)

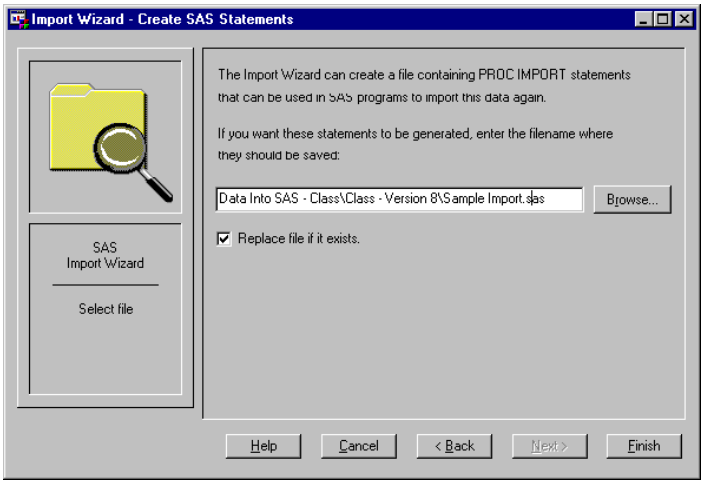

Figure F – SAS Import Wizard "Create SAS Statements" Screen

```
PROC ACCESS DBMS=EXCEL;
  CREATE WORK._IMEX_.ACCESS;
  PATH='C:\SAS Class\Sample
        Data\NHL 1999-2000 Scoring
        Leaders.xls';
  GETNAMES YES;
  SCANTYPE=YES;
  CREATE WORK._IMEX_.VIEW;
  SELECT ALL;
RUN;
DATA WORK.tempdata;
  SET WORK._IMEX_;
RUN;
```

Figure G – SAS Import Wizard Generated Code – Version 6.12

```
PROC IMPORT OUT=WORK.Sampdat1
     DATAFILE="C:\SAS Class\Sample
        Data\NHL 1999-2000 Scoring
        Leaders.xls"
     DBMS=EXCEL2000 REPLACE;
  GETNAMES=YES;
RUN;
```

Figure H – SAS Import Wizard Generated Code – Version 8

The SAS Import Wizard is not always the optimal solution - if it was, this would be an awfully short presentation! There are two major drawbacks to the Import Wizard. First of all, it is not available when running SAS in a mainframe environment. Secondly, it is an interactive tool, which renders it useless for a batch, schedule-oriented environment. However, the SAS Import Wizard can be the easiest, quickest, and therefore best solution for the one-time-only processing of an external file during an interactive SAS session. It also has great utility as a code generator, saving time for the user to invest in other areas.

PROC IMPORT

As mentioned earlier, PROC IMPORT (along with the corresponding PROC EXPORT) is a new addition to the SAS System. It provides a simple-to-code method that facilitates the transfer of data from an external source to the SAS System.

It is not necessary to understand the syntax of PROC IMPORT if the user generates the code from the Import Wizard – in general, wizards are designed so that the end user does not need any advanced knowledge of the tool in question. However, since PROC IMPORT can be invoked manually, either from scratch or by including code that was generated by an earlier execution of the Import Wizard – with or without further modification – it is worth briefly reviewing the procedure at this time.

There are two required arguments on the PROC IMPORT statement. The first, **OUT=**, should be familiar to all but the most inexperienced SAS users. The other required "argument" can actually be one of two different arguments, depending on the source of the input data:

- **DATAFILE="filename"** is used to specify most input data sources, such as sequential files or Excel spreadsheets.
- **TABLE="tablename"** is used to specify DBMS tables, such as from Microsoft Access.

There are also two optional arguments. **DBMS=** specifies the type of data to be imported. It is not required in conjunction with DATAFILE=, as long as the filename contains a valid extension associated with the data source such as .XLS. However, it *is* required if the file name does not have a valid extension or if TABLE= was specified instead of DATAFILE=. The other optional argument, **REPLACE**, controls whether or not a preexisting SAS dataset is overwritten by the procedure.

There are a number of statements available for use in conjunction with PROC IMPORT, depending on the data source that is being processed and on whether DATAFILE= or TABLE= is being used on the PROC statement. To conserve space, these options will not be discussed in this paper – the reader is directed to the Version 8 *SAS Procedures Guide* for details.

As with the Import Wizard, the options available to PROC IMPORT are limited to .TXT and .CSV (or other delimited file) if the users' site has not licensed SAS/ACCESS to PC File Formats.

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 I 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 I - 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**, or the shortened alias **LINES**, 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 J 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 J - 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 K 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 K - 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 printed version of the *SAS Language: Reference* manual under Version 6 devoted 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 L 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 L - 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. It is also possible to use a text string with 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. **See Figure M for examples of @"string".** 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.

```
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 M - 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 that 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 N for an examples of this approach.**

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

Figure N - 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 O 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 O - CSV File

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 not entirely true. There are *two* file structures that can be created by the DATA step : the traditional SAS data set, and the newer *SAS data view*.

The SAS data view does not contain actual data; rather, it contains a description of data that 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 P for an example of the creation and use of a SAS DATA step view (along with an SQL view).**

DDE

The final method of obtaining external data that shall be discussed in this presentation is *Dynamic Data Exchange*. Dynamic Data Exchange, or DDE, allows a client application to request information from a server application in a Windows or OS/2 environment. Effective with Release 6.08, the SAS System acts as a client application in this relationship. It can request data from a server application, with the requirement that the server application must be running. (It can also send commands and data to a server application, but that is a topic for another presentation.)

In order to use DDE, a connection must be established between the client application and the server application. This is accomplished by issuing a FILENAME statement with the keyword "DDE". The syntax for this statement, in this context, is:

```
FILENAME fileref DDE 'DDE-triplet' ;
```

The DDE-triplet is a specialized argument, and is made up of three components:

```
application|topic!item
```

Application is the name of the server application, such as Excel. *Topic* is defined as the "topic of conversation"; basically, this is the file to be processed. *Item* is the "item of conversation"; in a spreadsheet, this is the range of cells that is to be included. For example, the DDE-triplet an Excel worksheet would be:

```
Excel|[Book1]Sheet1!R1C1:R250C4
```

Note that the application and topic are separated by a vertical bar (|), while the topic and item are separated by an explanation point (!).

```
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 CONFCITY $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, CONFCITY, 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 P - SAS Data View, Creation and Use (along with example of SQL View)

The DDE triplet for an application should be defined in the documentation for that application. However, most people find it easier to let SAS determine the proper DDE triplet. The following is a step-by-step method to obtain the proper DDE triplet for an application, assuming both SAS and that application are active:

- Toggle to your application, and use the standard PC "cut" techniques to store the portion of the client application to be processed in the Windows Clipboard. (For example, use the mouse to highlight the area to be "cut", then select CUT or COPY on the EDIT pop-up menu of most Windows applications.)
- Toggle to your SAS session, and click on the "Options" menu on the Menu Bar in SAS.
- The Options menu will contain an option called "DDE Triplet". Click on it.

- This will display an Information Box, which will contain the DDE-triplet.
- Enter this DDE-triplet into the FILENAME statement of your SAS routine.

If the user is willing to perform a little manual intervention, it is even possible to use DDE without ever knowing the name of the DDE triplet!

- As above, toggle to your application, and use the standard PC "cut" techniques to store the portion of the client application to be processed in the Windows Clipboard.
- Toggle to your SAS session, and replace the FILENAME statement with the following:

```
FILENAME fileref DDE CLIPBOARD ;
```

The SAS routine is now ready to be executed. The weakness in this approach is that the data to be processed must be stored in the Clipboard prior to each invocation of your SAS routine. The benefit is that there is no need to ever know the DDE-triplet for your application to use DDE. (Please note that this approach will only work if an application is DDE compliant.)

In order to use DDE with the SAS System, the server application must be running while SAS is running. If the server application is not active, then it can be invoked from within the SAS session with the "X" command. However, the SAS options XSYNC and XWAIT must be turned off before issuing this command, or control will not be returned to the SAS session until that external application is closed -- this defeats the purpose of a DDE link! (Of course, the user could also simply toggle over to the Windows Program Manager and manually invoke the application.)

The actual transfer of data from the external application to the SAS System is done via the combination of an **INFILE** and **INPUT** statement. The actual code to accomplish this task looks exactly like the code to read a sequential file into SAS. **See Figure Q for an example of reading an Excel 5.0 spreadsheet via SAS.** For further examples covering a number of PC products, please refer to "Technical Support Document #325 - The SAS System and DDE", which is available on the SAS Institute web site.

```
OPTIONS  NOXSYNC NOXWAIT;
X 'C:\EXCEL SASCONF.XLS' ;
FILENAME SASCONF DDE
   'Excel|[Book1]Sheet1!R1C1:R250C4';
DATA CONFSCHD;
   INFILE SASCONF;
   INPUT DAY TIME TITLE AUTHOR;
RUN;
```

Figure Q - INPUT from EXCEL using DDE

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. 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.

Bodt, Mark (1996). "Talking to PC Applications Using Dynamic Data Exchange". *Observations*, Volume 5, No. 3 (Second Quarter 1996). 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.

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.

Gilmore, Jodie (1997). "Using Dynamic Data Exchange with Microsoft Word". *Proceedings of the Twenty-Second 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.

Kuligowski, Andrew T. (1998). "An Overview of Techniques to Introduce External Data into the SAS System". *Proceedings of the Sixth Annual Conference of the SouthEast SAS Users Group*. USA.

Kuligowski, Andrew T. (1999). *Course Notes: Turning External Data Into SAS Data.* Dunedin, FL: self-published.

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

Sanders, Roger E. (1998). "Accessing Data from Your PC Using Version 7 of the SAS System". *Proceedings of the Twenty-Third Annual SAS Users Group International Conference*. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1995), *SAS/ACCESS Software for PC File Formats: Reference, Version 6, First Edition.* Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1994), *SAS/ACCESS Software for Relational Databases: Reference, Version 6, First Edition.* Cary, NC: SAS Institute, Inc.

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

SAS Institute, Inc. (2000). *SAS OnlineDoc, Version 8.* Cary, NC: SAS Institute, Inc.

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