

An Overview of Techniques to Introduce External Data Into the SAS® System

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 are stored in or can be accessed by the SAS System. This presentation will provide a brief introduction to many of the different methods that can be used to pass data into the SAS System. Topics will include the menu-driven SAS Import Wizard, the INPUT and INFILE statements, DDE, ODBC, and many others - as many as time and space will allow!

The goal of this presentation is to provide information that will be useful to all users of the SAS System. Some topics are tailored to the novice, while others will be more applicable to the experienced user. Please note that some information, by its very nature, will only be applicable to select operating systems.

SAS IMPORT FACILITY

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 Facility*. Available under Release 6.12 of the SAS System, the SAS Import Facility (also known as 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** from the **File** pulldown menu on the SAS toolbar.

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. 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.

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.

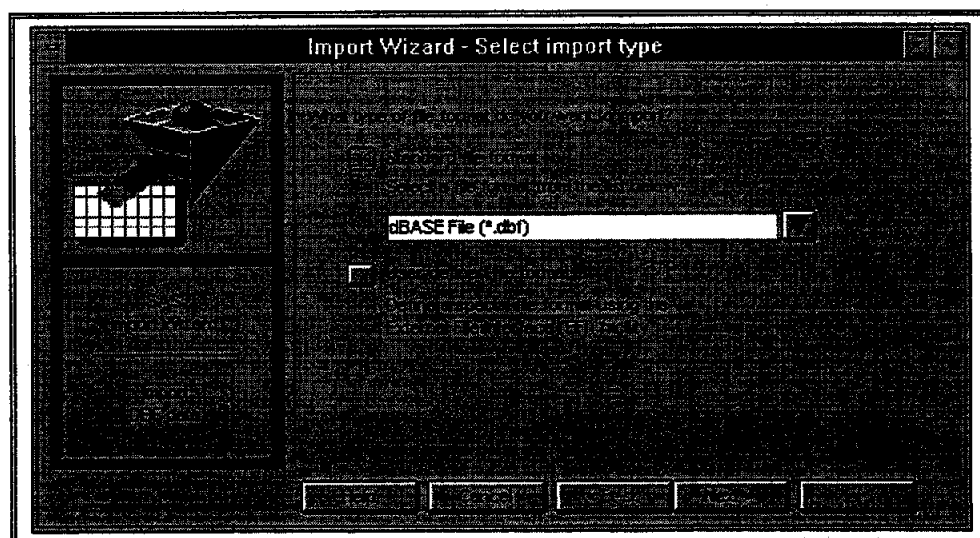


Figure "A" - SAS Import Wizard "Import Type" Screen

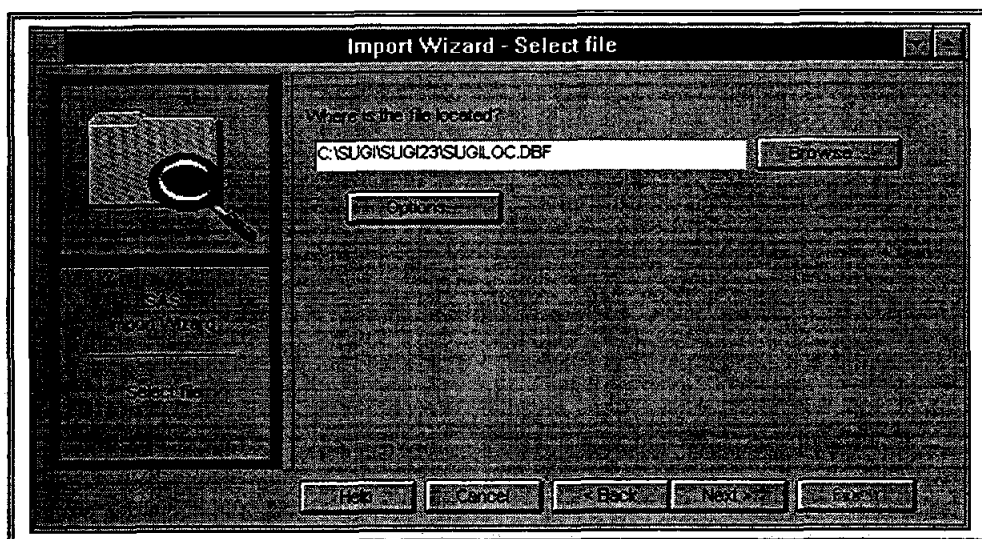


Figure "B" - SAS Import Wizard "Select File" Screen

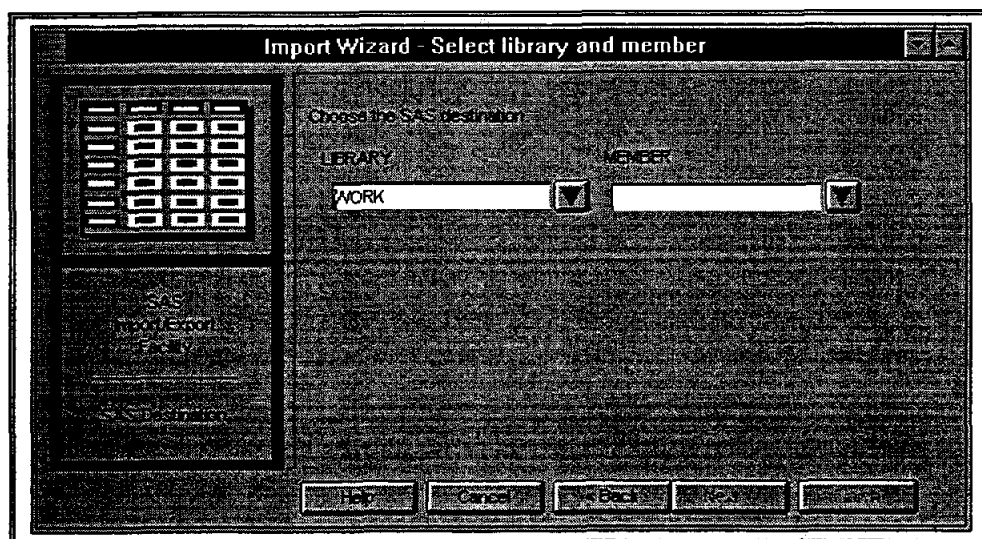


Figure "C" - SAS Import Wizard "Library & Member" Screen (initial)

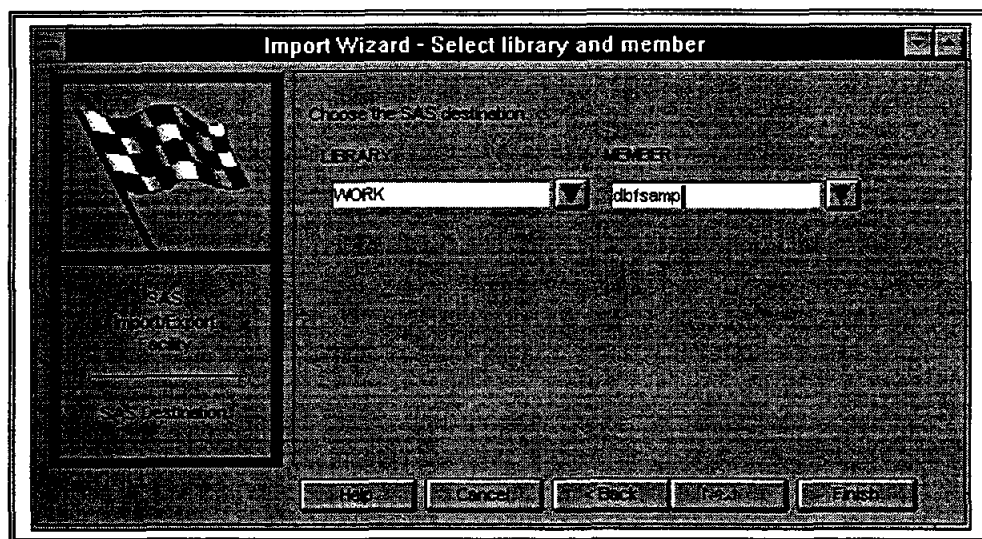


Figure "D" - SAS Import Wizard "Library & Member" Screen (completed)

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

Next, the user is prompted to "Select 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 C and D for samples of this screen. If "standard file format" was originally selected, please note that the picture on the left side of the screen changes to a checkered flag when entry is finished, and the *Next* button becomes unusable, replaced by the *Finish* button. Otherwise, the checkered flag will be displayed on the next screen, which will advise the user that the Import Wizard is complete, and control will be transferred to the External File Interface.

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 an IBM 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.

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 newer SAS *data 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 CONFPCITY $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, CONFPCITY, 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

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).

DDE

The next method that we shall discuss 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 (!).

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 M for an example of reading an Excel 5.0 spreadsheet via SAS. For further examples covering a number of PC products, the reader is directed 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 CONFSCD;
  INFILE SASCONF;
  INPUT DAY TIME TITLE AUTHOR;
RUN;

```

Figure "M" - INPUT from EXCEL using DDE

SAS/ACCESS ENGINES

SAS/ACCESS software is available for a variety of host systems, covering traditional mainframe, personal computer, and UNIX environments. It provides a method to view and transfer data from several common database management systems (DBMS) and a number of common PC file formats, into the SAS system.

The ACCESS procedure can create *descriptor* files that will provide information about the data stored in the DBMS table or PC file format, and use that information to create a SAS data file. In addition, use of an *interface view engine* will allow SAS to read data from the file formats directly into SAS routines. The interface view engine is used by the SAS SQL procedure to directly access external data bases without leaving the SAS session. However, the SQL statements used in the procedure are beyond the scope of this Tutorial.

There are two types of descriptor files created by the ACCESS procedure: an *access descriptor* and a *view descriptor*. Access descriptors provide information regarding the structure of the file to be accessed. This includes data types, table names, and column names, as well as the related SAS dataset information such as variable names and formats. This access descriptor can then be used to create the view descriptor which will contain criteria to be used to select columns and rows from the selected DBMS table or PC file. The data can be used directly from the view descriptor in the SAS routine, or it can be extracted from the DBMS or PC file into a SAS data file.

The type of DBMS or PC file to be used is specified in the PROC ACCESS statement in the form:

```
PROC ACCESS DBMS=filetype
```

Filetype can take on many different values. To cite just a few examples, the user can obtain data from DB2®, SYBASE®, and ORACLE® by selecting filetypes DB2, SYBASE, and ORACLE, respectively. In a Windows environment, XLS, and WK n (where n is a valid version number) will allow the transfer of data from Excel or Lotus® spreadsheets, while DBF and DIF are obviously the filetype for interfacing with .DBF and DIF formatted files, respectively.

The actual access or view descriptor is then created with the following syntax:

```
CREATE libref.member-name.ACCESS or
CREATE libref.member-name.VIEW
```

The PROC ACCESS and CREATE statements are followed by a statement that identifies the name of the DBMS, DBF, XLS or other file that will be accessed. In addition, there are other editing statements; these provide information about the structure of the DBMS or PC file being accessed, and select columns to be viewed. See Figure N for an example of using PROC ACCESS create a view, with a subsequent use of its output.

```

LIBNAME VWLIB 'c:\confdat\';
PROC ACCESS DBMS=xls;
  CREATE vwlib.states.access;
  PATH 'c:\confdat\state.xls';
  <editing statements omitted>
  CREATE vwlib.states2.view;
  <select, format, and
  subset statements omitted>
RUN;

DATA _NULL_;
  SET vwlib.states2;
  <statements omitted>
RUN;

```

Figure "N" - PROC ACCESS : Creation and use of a View

The SAS view descriptor can be used in any PROC or DATA step just like a SAS data set. It is also possible to use PROC ACCESS to create a SAS dataset from the view descriptor. This is accomplished by issuing PROC ACCESS with both the `VIEWDESC=libref.view-descriptor` and `OUT=libref.sas-data-filename` options. See Figure O for an example of using PROC ACCESS create a view, with a subsequent use of its output.

```
PROC ACCESS VIEWDESC=vwlib.states2
      OUT=vwlib.stdata ;
RUN;

PROC PRINT DATA=vwlib.stdata;
RUN;
```

Figure "O" - PROC ACCESS : Creation of a SAS Dataset

It is not possible to fully cover PROC ACCESS in the limited space of this paper -- there are a number of manuals dedicated to the topic! For further information, including details of the DBLOAD procedure that will transfer data from SAS to the assorted DBMS and PC products, the reader is directed to the assorted SAS/ACCESS manuals.

PROC DIF / PROC DBF

SAS/ACCESS to PC File Formats, which requires a separate license from Base SAS software, also provides additional interfaces to two traditional types of PC files. *DBF* and *DIF* formats date back to the "ancient" days of PCs in the early 1980s; however, data continues to exist in these formats and one must be prepared to deal with it.

DBF files contain data originally formatted for the dBASE™ database management product, and are accessed through PROC DBF. PROC DBF accepts two options. The first, `OUT=` should be self-explanatory to anyone with even a modest experience in use of the SAS System. The other, `DBn=`, is required, and contains the file reference to the DBF file. The *n* in "DBn" refers to the version of dBASE for which the file was created; "2", "3", "4", and "5" are valid version numbers. Please note that all dBASE formatted files are assigned the extension .DBF, regardless of the version of dBASE (or other product) under which they were created. SAS will produce an error message if the version number is incompatible with the file format.

PROC DBF will produce a SAS dataset, with variable names mapped from the original dBASE field names. Field names will be truncated to 8 characters, if necessary, to conform to current SAS variable name

restrictions. See Figure P for examples of PROC DBF.

```
10 /* This is a dBASE IV format file. */
11 filename sasconf
12 'c:\sasconf\confloc.dbf';
13 proc dbf db2=sasconf out=conf_db2;
14 RUN;

ERROR: Input file is not a DBASEIII file.
WARNING: Data set WORK.CONF_DB2 not replaced
        because new file is incomplete.
NOTE: The SAS System stopped processing this
      step because of errors.
NOTE: The PROCEDURE DBF used 0.44 seconds.

13 proc dbf db4=sasconf out=conf_db4;
14 RUN;

NOTE: 7 observations written to the output
      SAS data set.
NOTE: The PROCEDURE DBF used 0.38 seconds.
```

Figure "P" - PROC DBF - Invalid and Correct

DIF files, short for Data Interchange Format, date back to VisiCalc™ and other early PC spreadsheet products. PROC DIF can be used to convert DIF files into SAS datasets. The format for PROC DIF is similar to PROC DBF, with the `DIF=` option used in place of `DBn=`. Variable names are assigned `COL1` to `COLn` in the output SAS dataset; it is recommended that the user subsequently reassign them to something more meaningful. See Figure Q for an example of PROC DIF.

```
filename sasconf 'c:\sasconf\confloc.dif';
proc dif dif=sasconf out=conf_dif;
run;
proc print uniform;
run;
```

Figure "Q" - PROC DIF

ODBC

Open DataBase Connectivity, or ODBC, started off as a standard for the exchange of data between DataBase Management Systems (DBMS) under Microsoft's Windows environment. Since those early days, interfaces to other operating systems and machines, such as the Apple Macintosh, have been developed. It is necessary to use the SAS/ACCESS Interface to ODBC in order to use ODBC to bring data into the SAS System.

It is important to note that the SAS/ODBC interface does not directly obtain data from an external source, unlike other data sources which are available via SAS/ACCESS. Instead, it interfaces with the ODBC manager, which in turn interfaces with the other external data sources. In order to use the

SAS/ODBC Interface, it is therefore first required to install the appropriate drivers, or "Data Sources".. This is done via the ODBC Icon available through the operating system. (In Windows 3.1, for example, this could be found in the "Control Panel".) See Figures R, S, T, and U for examples of the screens used to define a SAS driver in ODBC.

The SAS / ODBC interface is also unlike the other products in the SAS/ACCESS family, in that PROC

ACCESS is not used to bring external data into SAS. Instead, the SAS System utilizes an enhancement to PROC SQL, the *SQL Procedure Pass-Through Facility*. (Please note that it is still necessary to license and install SAS/ACCESS to ODBC in order to use the SQL Pass-Through Facility in conjunction with ODBC. It should also be noted that the SQL Pass-Through Facility can also be used with other databases via separate SAS/ACCESS licenses; these will not be discussed in this presentation.)

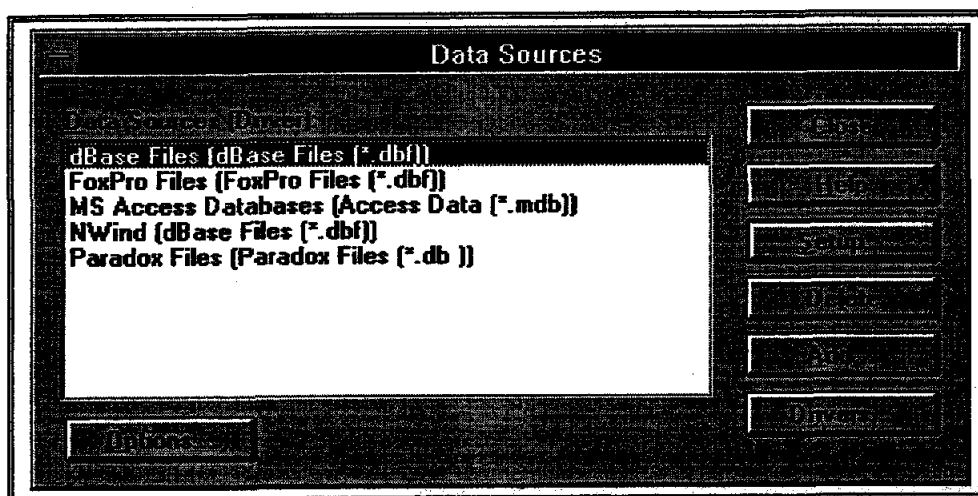


Figure "R" - Windows 3.1 ODBC Driver Configuration - Active Data Sources

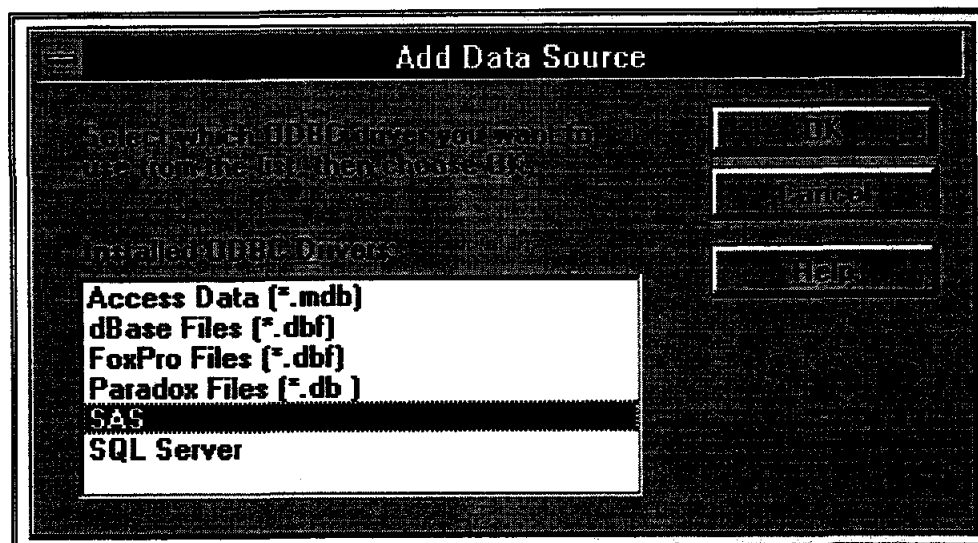


Figure "S" - Windows 3.1 ODBC Driver Configuration - Add Data Source

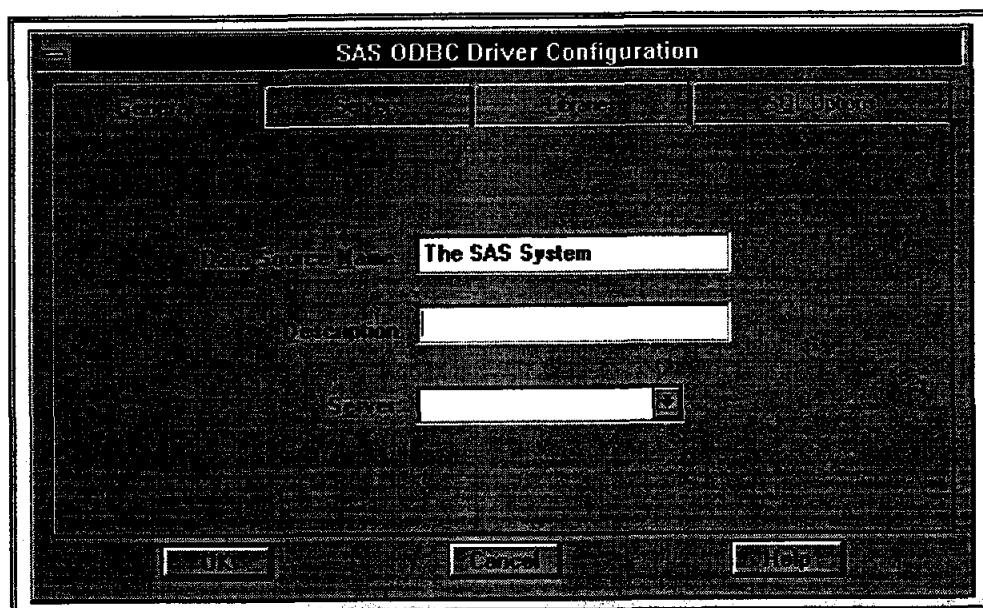


Figure "T" - Windows 3.1 ODBC Driver Configuration - General Information

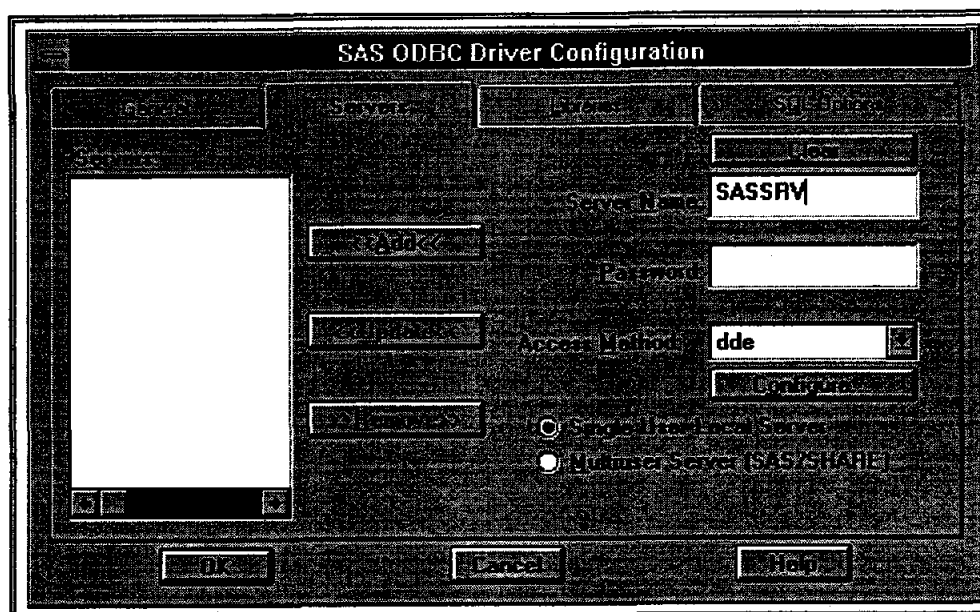


Figure "U" - Windows 3.1 ODBC Driver Configuration - Servers

The SQL Procedure Pass-Through Facility has four statements associated with it; or more correctly, three statements and a "component", which is included within the SQL itself.

The first statement, **CONNECT**, establishes a connection with ODBC. The syntax is:
CONNECT TO ODBC <AS alias> <(options)>
 The alias is optional; however, if used, the word "AS" *must* immediately precede it. The options are used to specify the data source which is to be processed, and must be enclosed within parentheses. **DSN=data-source** can be used to specify a

previously defined data source, as described above. (Note that "DSN" stands for "Data Source Name", and not "Data Set Name".) Alternatively, the word **PROMPT** will walk the user through the process of setting up an ODBC interface at execution time. Note that **PROMPT** and **DSN=** are mutually exclusive of each other. See Figures V and W for an example of the screens displayed to define a dBASE data source. See Figures X and Y for the SAS source code to use these screens. One other option that should be noted is **LOG**, which will cause all warnings and errors generated by the ODBC API (Application Programming Interface) to

be written to the SASLOG. This may be especially useful during the development phase of an application.

As one would expect, **DISCONNECT** is the opposite of **CONNECT**. This statement ends the connection with ODBC. The syntax is :

DISCONNECT FROM ODBC | alias;
using either the word "ODBC" or the alias that was specified in the **CONNECT** statement. It is optional, as an implicit **DISCONNECT** is issued when the **QUIT** statement ends **PROC SQL**. However, it may be desired to explicitly break the connection if more SQL statements are to be issued, or if a different **CONNECT** is desired.

The **EXECUTE** statement sends non-query SQL statements to ODBC. These are primarily used to update the external database, and as such are outside the scope of this presentation.

CONNECTION TO is a clause that can be inserted in an SQL **SELECT** statement to send an SQL query to the external data source accessed via ODBC. The syntax is:

FROM CONNECTION TO ODBC | alias
(SQL to external source)

again using either the word "ODBC" or the alias that was specified in the **CONNECT** statement. There are a number of special queries that can be used in conjunction with ODBC; these will be omitted from this presentation due to space limitations.

All of these statements and components can be combined with the statements normally associated with **PROC SQL** to form a successful ODBC query. See Figure X for an example of **PROC SQL** using **PROMPT**. See Figure Y for an example of **PROC SQL** using **DSN=**.

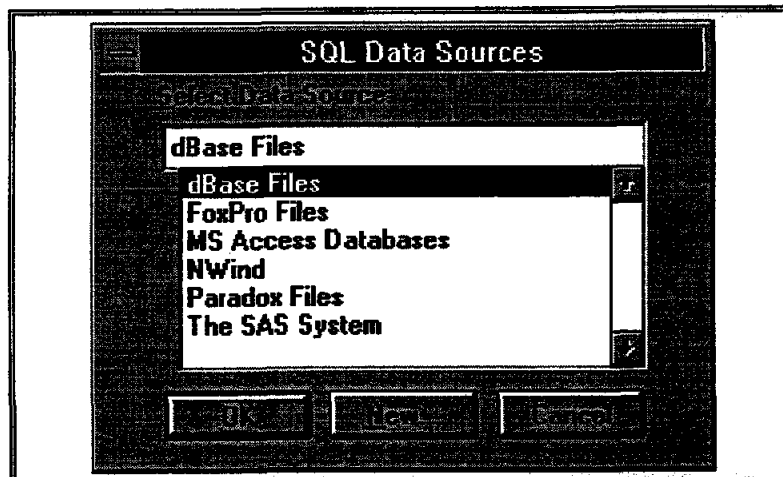


Figure "V" - SQL Pass-Through Facility **CONNECT TO ... PROMPT / Data Source**

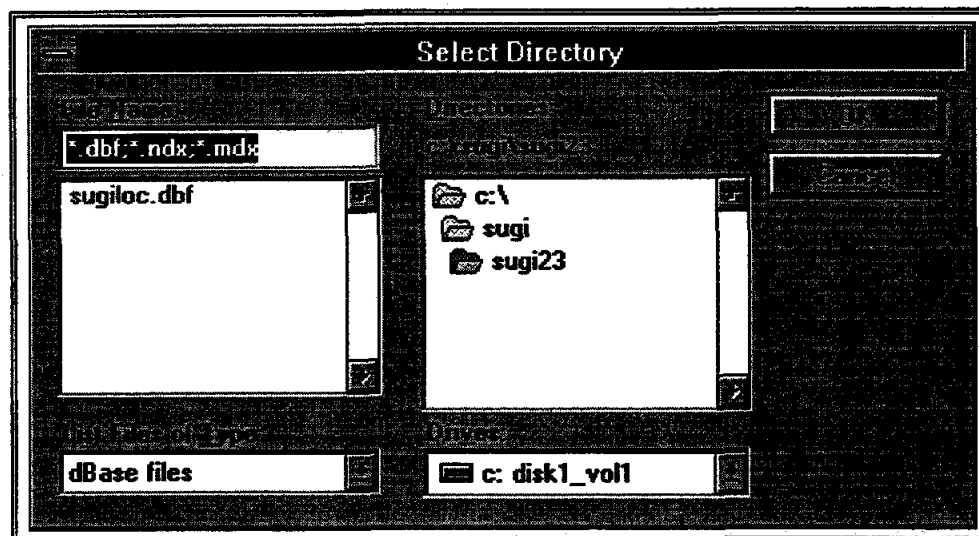


Figure "W" - SQL Pass-Through Facility **CONNECT TO ... PROMPT / Select Directory**

```

84  /* "feedback" option expands SQL */
85  /* statements in the SASLOG.      */
86  proc sql feedback ;
87      connect to odbc(prompt);
88      create table confloc as
89          select * from connection to odbc
90              (select * from confloc );
NOTE: Statement transforms to:
      select YEAR, CONFEREN, CITY, STATE
      from connection to ODBC
          /* dbms=ODBC,
      connect options=(prompt) */
          ( select * from confloc );
NOTE: Table WORK.CONFLOC created, with
      7 rows and 4 columns.
91      disconnect from odbc ;
92      quit ;
NOTE: The PROCEDURE SQL used 7.91 seconds.

```

Figure "X" - SQL Pass-Through Facility using PROMPT

```

218 proc sql ;
219     connect to odbc as sasconf
220         (dsn="confdemo");
221     create table confloc as
222         select * from connection to sasconf
223             (select * from confloc );
NOTE: Table WORK.CONFLOC created, with
      7 rows and 4 columns.
224     disconnect from sasconf ;
225     quit ;
NOTE: The PROCEDURE SQL used 3.35 seconds.

```

Figure "Y" - SQL Pass-Through Facility using DSN=

OLE

Object Linking and Embedding, or OLE, is another client/server methodology which will allow the transfer of data between SAS and other products. In the SAS System, OLE functionality is typically attained through SAS/AF[®] using FRAME catalog entries or through SAS/EIS[®], by invoking other products from within SAS. However, for our purposes, the SAS OLE Automation Server will allow other products to interface with Base SAS.

OLE is more graphically oriented than DDE; OLE will permit the SAS System to share a number of different objects, such as graphs and charts, with other products. As the name implies, OLE consists of an *object linking* process and an *object embedding* process. The object link permits an object to be updated on either the client or server side of the link, with the change reflected on the other side of the link. An embedded object, on the other hand, can only be changed from the client application.

The SAS OLE Automation Server is only supported for the Windows 95 and Windows NT operating systems.

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.

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.)

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

Under MVS:
//TRANFILE DD DSN=mvse.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.

PROC FSEDIT / PROC FSVIEW

All of the mechanisms discussed to this point have been based on the assumption that the data exists in an electronic format which can be accessed from the SAS System. This is not always true; sometimes, the data might only be available on a printed report. Under these circumstances, it might be desirable to manually enter the data directly into a SAS dataset.

The SAS System offers two tools, available under SAS/FSP, to facilitate data entry directly into a SAS dataset. The first, PROC FSEDIT, creates a data entry screen that will accept one observation. The second, PROC FSVIEW, creates a data entry table that will accept multiple observations per screen.

The technique used to create and populate a new SAS dataset using either of these tools is quite simple. The first step is to issue the appropriate PROC FSEDIT (or FSVIEW) statement, using the NEW= option to specify the name of the new SAS dataset, in place of the traditional DATA= option. The syntax is :

```
PROC FSEEDIT NEW=newdsn; RUN;
```

(In order to use FSVIEW, simply substitute "FSVIEW" for "FSEEDIT" in the syntax listed above.)

The second step is to fill out a screen, containing entries for Variable Name, Type, Length, Label, and Format. The third and final step is to enter the data.

See Figures CC, DD, and EE for an example of creating a new SAS dataset using PROC FSEDIT.

There are a number of options available to customize the appearance and usage of PROC FSEDIT and PROC FSVIEW. The reader is strongly encouraged to refer to the 100+ pages on the topic in the *SAS/FSP Software Usage and Reference* manual for further details.

```
proc fsedit new=confdemo;
run;
```

Figure "CC" - Invoking PROC FSEDIT

[illegible]

Figure "DD" - The FSEDIT New Window

```

FSEDIT WORK.CONFDEMO  Obs 0

CONFNAME: _____
CONFCITY: _____
CONFST:  _____
YEAR:    _____
  
```

Figure "EE" - FSEDIT Screen

THIRD PARTY PRODUCTS

In addition to the methods built into the Base SAS product and additional modules from SAS Institute, there are products produced by third-party providers that facilitate the transfer of data into SAS. In some cases, the user base for a given product does not warrant a separate SAS/ACCESS engine. The vendor for that product may provide their own interface to the SAS System. In other cases, an entrepreneur may believe that they have found a niche market which they believe SAS Institute has overlooked, or that they can provide an interface which is faster / quicker / cheaper / etc.; the marketplace will determine whether or not they have succeeded.

One example of a proprietary third-party product is PROC SAGE™. The Computer Corporation of America (CCA) supplies and supports a mainframe database called Model 204®. They also supply an interface called PROC SAGE, which passes data from a Model 204 database into a SAS dataset.

Conceptual Software, Inc. and Circle Systems both market products which they assert can be used to transfer external data into the SAS System under both Windows (3.1, '95, and NT) and various UNIX platforms. Conceptual Software, Inc. has two products that fit this category: DBMS/COPY™, to provide connectivity between over 80 different external data sources, including the SAS System; and DBMS/Engines™, to provide engines which allow the SAS System to interact with all of the products that can interface with DBMS/COPY. Circle Systems also has a product, Stat/Transfer™, which can be used to transfer data between SAS datasets and various other products. NOTE: It is not the purpose of this presentation or its author to either endorse or condemn any of these 3rd party products or their respective vendors. All claims are summarized based solely on information found on the companies' web sites.

There are also some utilities available in the public domain - the proper tool for your needs may already be available at no additional cost! The Proceedings from SUGI and from the various Regional SAS User Group conferences are good places to look, as are the SAS-L archives and the World Wide Web. For example, there are at least two published methods to bring the contents of a .dbf file into SAS; ironically, each is called DBFTOSAS. The first is a PERL script, written by Frank Stetzer, which will create two sequential files from a .dbf file - one containing the file's data in a sequential format, and the other being

a SAS routine to process the first file. The other is a SAS macro, written by Richard Hockey. (The URLs for the Web sites containing these utilities can be found at the end of this presentation.)

ADDENDUM : VERSION 7 OF THE SAS SYSTEM

In March of 1998, SAS Institute announced the scheduled release of the long-awaited Version 7 of the SAS System. The upgrade is scheduled to be in Beta Test through much of 1998, with production shipping scheduled to begin in the 4th quarter of the year.

Most SAS routines written in Version 6 should be able to run under Version 7 without alterations. Therefore, the information contained in this presentation should not be rendered obsolete by the new release. However, the release will have a number of improvements, including some enhancements that should facilitate the transfer of external data into the SAS System. These include:

- **Variable names** will be increased to a maximum of 32 characters in length. These names will be stored in mixed case, although subsequent use of the variable name will not be case-sensitive. In addition, the SAS System can waive the normal variable name rules for SAS/ACCESS, including the allowance of blanks and special characters, through the use of the new SAS system option **VALIDVARNAME=ANY**. These features should combine to facilitate the use of field names from external DBMS - without having to convert them to an 8 character name for use within SAS.
- The **maximum length of character fields** has been increased from 200 to 32,767 characters.
- The **_INFILE_** pseudo-variable can now be used under the same circumstances as a regular variable. Formerly, it could only be used in conjunction with the PUT statement.
- **SCANOVER**, a new option on the INFILE statement, will permit the INPUT statement to automatically scan the next line when using @"textstring" construct, should the desired string not be found on the current input data line.
- **PROC IMPORT**, a new feature, will permit the use of the Import Facility within a SAS routine. Currently, this is only available via a series of interactive menus. In addition, the Import Facility

will now support Excel 97 (*.xls) and Microsoft Access (*.mdb) database files.

- **SAS/ACCESS** definitions can now be made via a **LIBNAME** statement - **ACCESS** and **VIEW** tables are no longer required. This will allow the SAS System to refer to and use an external database as though it was a SAS data library. As a result, full-screen support for the **ACCESS** and **VIEW** descriptors will no longer be provided.
- **SAS/ACCESS** engines for **AS/400** and **MS SQL** have been dropped from Version 7. Users are encouraged to move to the **SAS/ODBC** interface - a new **PROC CV2ODBC** has been provided to facilitate the conversion.

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.

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

Clegg, Jennifer, and Rigsbee, Carol (1997). "OLE and the SAS System for Windows Release 6.12". *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.

Gona, Vino, and Van Wyk, Jana (1998). "Version 7 Enhancements to SAS/ACCESS Software". *Proceedings of the Twenty-Third 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). "Basic Methods to Introduce External Data into the SAS System". This is currently an unpublished work.

Kuligowski, Andrew T. (1998). "You Can RUN; But Your Data Cannot Hide : Advanced Methods to Introduce External Data into the SAS System". This is currently an unpublished work.

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

Ma, Meimei, and Schlotzhauer, Sandra (1998). "Tips and Techniques for Moving Between Operating Environments". *Proceedings of the Twenty-Third Annual SAS Users Group International Conference*. Cary, NC: SAS Institute, Inc.

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

Praxis International, Inc. (1993), *SAGE/204 User's Guide Release 2.1*. USA: Praxis International, Inc.

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

Riba, S. David, and Riba, Elisabeth A. (1996), "ODBC: Windows to the Outside World". *Proceedings of the Fourth Annual Conference of the SouthEast SAS Users Group*. USA.

Riba, S. David (1996), "Open DataBase Connectivity and SAS". *The NESUG Express*, October 1996 Edition.

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), *DBF and DIF Procedures, Release 6.11, Preliminary Documentation*. Cary, NC: SAS Institute, Inc.

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

SAS Institute, Inc. (1994), *Getting Started with SAS/ACCESS Software, Version 6, First Edition*. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1994), *SAS/ACCESS Software Changes and Enhancements : SQL Pass-Through Facility, Version 6*. 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. (1994), *SAS ODBC Driver Technical Report: User's Guide and Programmer's Reference, Version 6.10*. 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.

Web Sites

Circle Systems, "Stat/Transfer. Home Page".
<http://www.stattransfer.com>.

Conceptual Software, Inc., "Conceptual Software, Inc. Home Page".
<http://www.conceptual.com>.

Cram, Donald P., "Excel 2 SAS and Back Webpage".
<http://www-leland.stanford.edu/class/gsb/excel2sas.html>.

Hockey, Richard, "DBFTOSAS Macro".
<http://ftp.urz.uni-heidelberg.de/ftp/put/sas/makros/dbtosas.sas..>

Riba, S. David, "JADE Tech Publications".
<http://www.jadetech.com/publicat.htm>.

SAS Institute, Inc., "Technical Support Document #325 - The SAS System and DDE".

<http://www.sas.com/techsup/download/technote/TS325.doc>. (*)

SAS Institute, Inc., "Technical Support Document #501 - Data Acquisition and Exportation in PC SAS".

<http://www.sas.com/techsup/download/technote/TS501.pdf>. (*)

Stetzer, John, "DBFTOSAS PERL Script".

http://www.iiug.org/memb_software/archive/dbftosas.perl

(*) Most SAS Institute Technical Support Documents are available in multiple formats, including .DOC, .PDF, and .TXT.

SAS, SAS/ACCESS, SAS/AF, SAS/EIS, and SAS/FSP are registered trademarks or trademarks 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:

Andrew T. Kuligowski

0005949476@mcimail.com

kuligoat@tvratings.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. I would also like to thank Nancy, along with Linda Kuligowski, S. David Riba, and Nathaniel Wooding, for their assistance and support during the production of this document.