

Advanced Methods 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, with special emphasis on those methods for use on the Personal Computer. Topics will include the menu-driven SAS Import Wizard, DDE, ODBC, and 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. This presentation will be tailored to Version 8 of the SAS System, however special note will be made of those topics that have changed between Versions 6 and 8.

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.

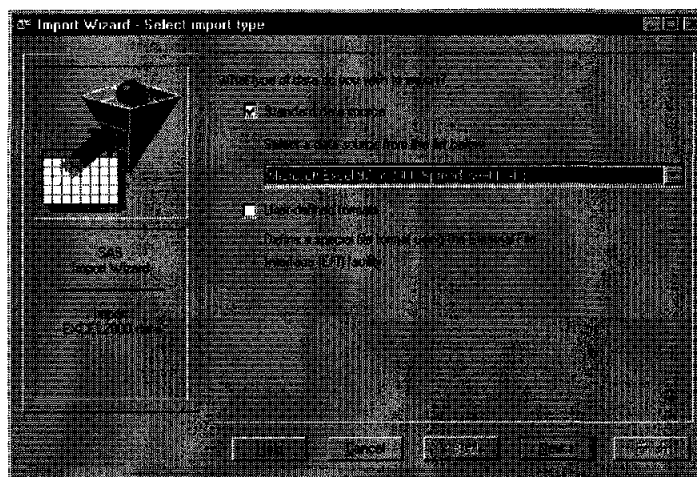


Figure A – SAS Import Wizard "Import Type" Screen

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.

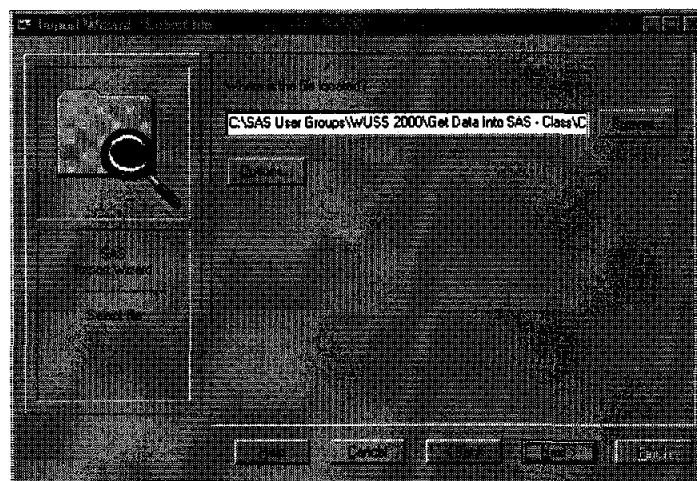


Figure B - SAS Import Wizard "Select File" Screen

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.

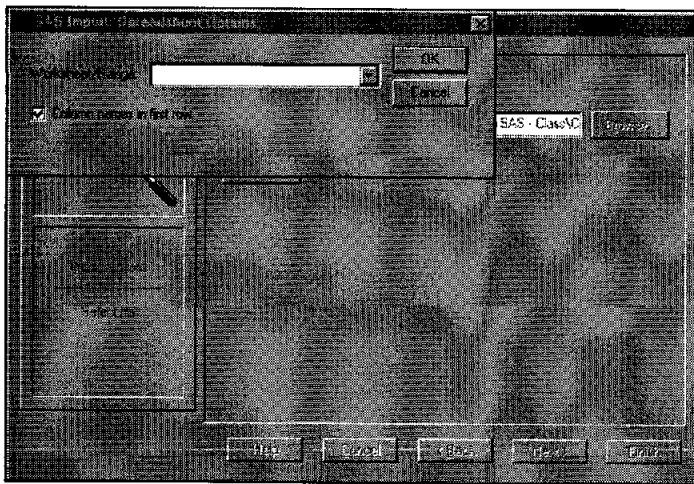


Figure C – SAS Import Wizard “Select File – Options” Screen

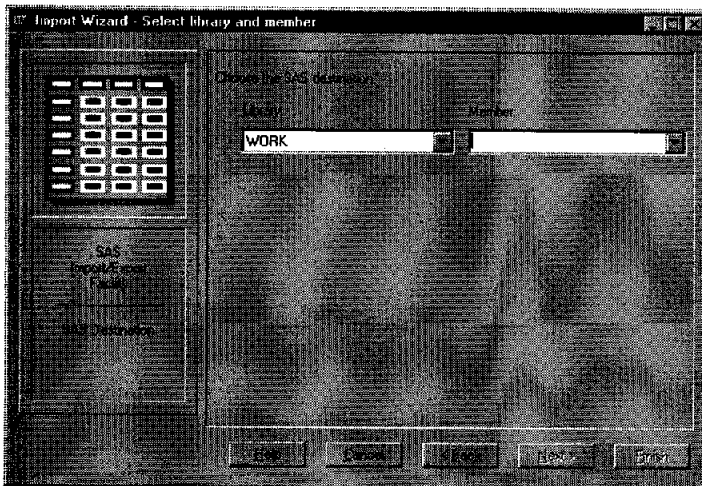


Figure D – SAS Import Wizard “Library & Member” Screen (Initial)

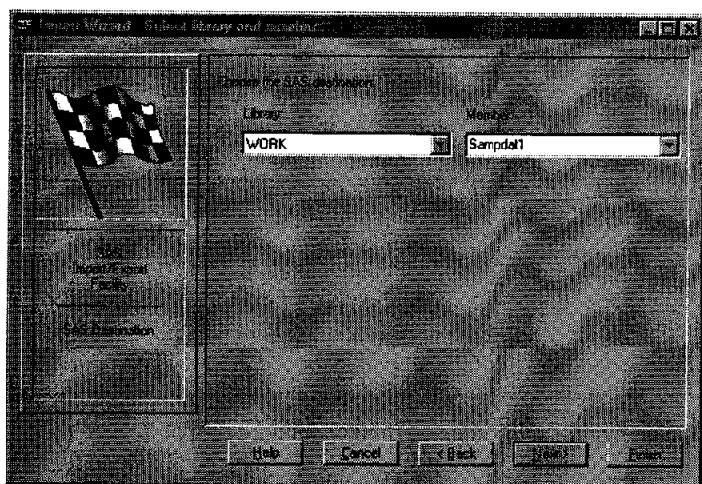


Figure E – SAS Import Wizard “Library & Member” Screen (Completed)

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.

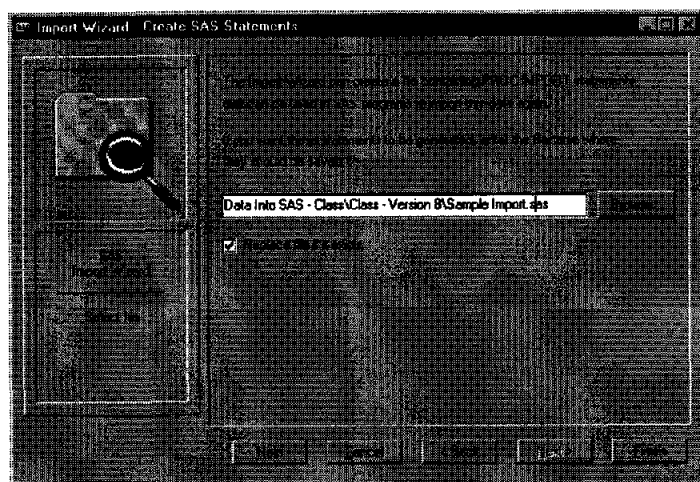


Figure F – SAS Import Wizard “Create SAS Statements” Screen

```

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

TUTORIALS

(or other delimited file) if the users' site has not licensed SAS/ACCESS to PC File Formats.

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 1 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 1 - CSV File

DDE

The next 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 (!).

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 J** 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 NOKSYNC NOKWAIT;
X 'C:\EXCEL SASCONF.XLS' ;
FILENAME SASCONF DDE
'Excel|[Book1]Sheet1!R1C1:R250C4';
DATA CONFSCHD;
  INFILE SASCONF;
  INPUT DAY TIME TITLE AUTHOR;
RUN;

```

Figure J - INPUT from EXCEL using DDE

It should be noted that the use and support of DDE is declining, as more recent technological advances become the tools of choice in the new millennium. However, the reader is encouraged to develop an understanding of DDE for several reasons:

- New DDE applications are still being coded at many sites around the world.
- There are many existing applications that were coded to use DDE; these applications will need maintenance.
- DDE is available with base SAS, and does not require additional product licensing or installation, making it a viable alternative for many sites.

SAS/ACCESS ENGINES

SAS/ACCESS software is available, under separate licenses, 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 databases 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 K** for an example of using PROC ACCESS create a view, with a subsequent use of its output.

```
LIBNAME VWLIB 'c:\confdat\';
PROC ACCESS DEMS=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 "K" - 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 the `VIEWDESC=libref.view-descriptor` and `OUT=libref.sas-data-filename` options. See Figure L 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 "L" - PROC ACCESS : Creation of a SAS Dataset

The process to create and use a SAS/ACCESS view, as described above, is reasonably easy to understand and to use. However, under Version 8, SAS/ACCESS became even easier to employ, by allowing view definitions via the LIBNAME statement! The syntax is straightforward:

```
LIBNAME libref enginename
        <engine-specific options>
        <general LIBNAME options>;
```

This process will be described under the ODBC section of this presentation.

It is not possible to fully cover PROC ACCESS in the limited space of this paper -- there are a number of separate 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 and 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 M for examples of PROC DBF.

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

ERROR: Input file is not a DBASEII 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;
RUN;

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

Figure "M" - 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 N 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 "N" - 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. (Under Windows 98, for example, this could be found in the "ODBC Administrator" from the Program Menu, or the "ODBC Data Sources" icon in the Control Panel.) **See Figures O, P, Q, R, and S for examples of the screens used to define a SAS driver in ODBC with the ODBC Data Sources method. The ODBC Administrator uses the same basic menu s, although the first 2 menus are different.**

It should be noted that under Version 6, the SAS ODBC driver used DDE for product to product communication. As of Version 7, the SAS ODBC driver uses TCP/IP protocol for communication. Version 6 users will need to update their driver settings when upgrading to Version 8.

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.)

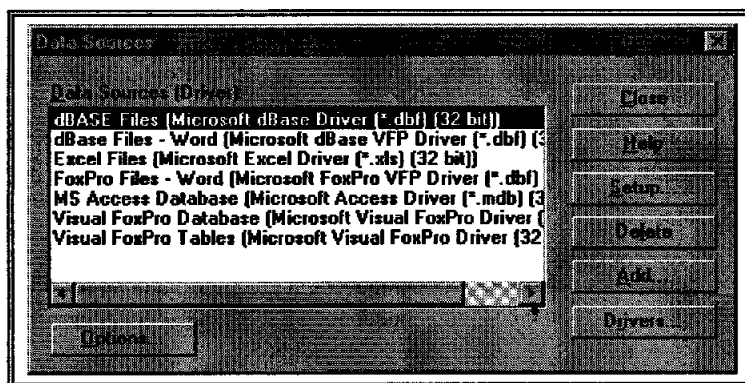


Figure "O" - ODBC Driver Configuration Active Data Sources

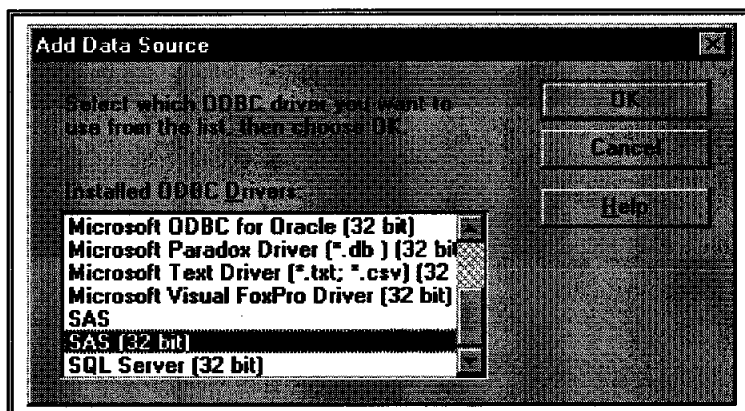


Figure "P" ODBC Driver Configuration - Add Data Source

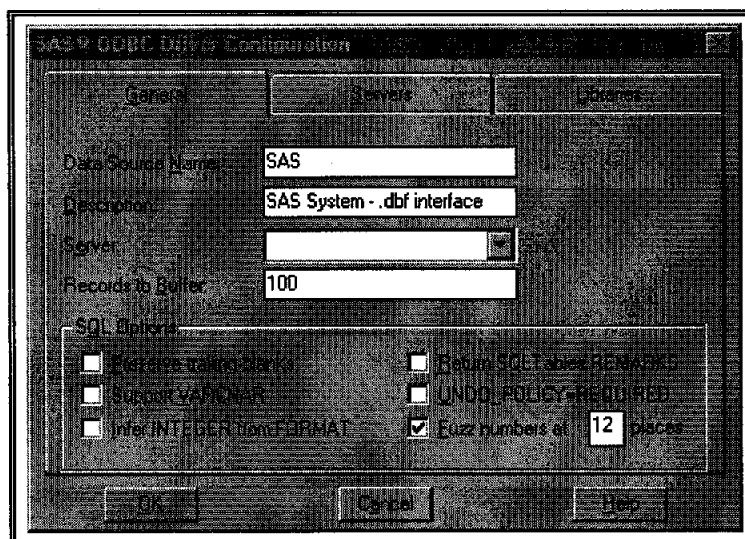


Figure "Q" - ODBC Driver Configuration - General Information

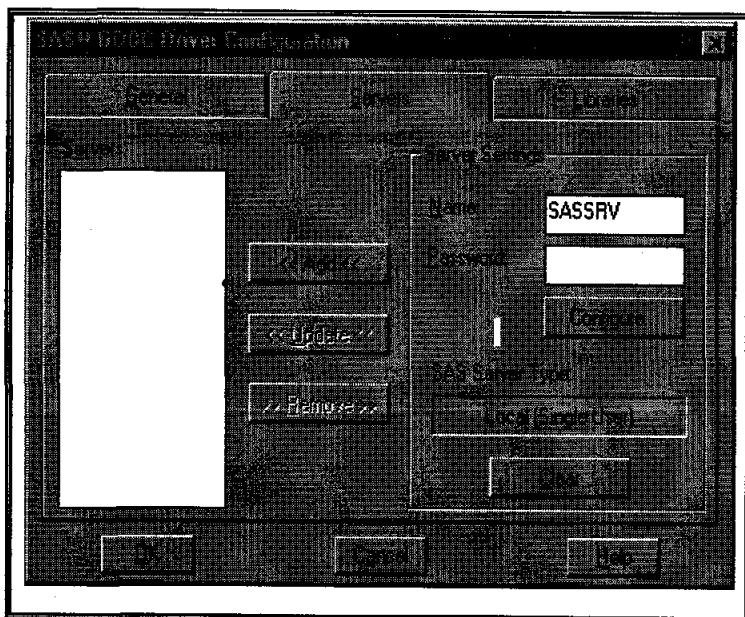


Figure "R" - Windows 3.1 ODBC Driver Configuration Servers (1 of 2)

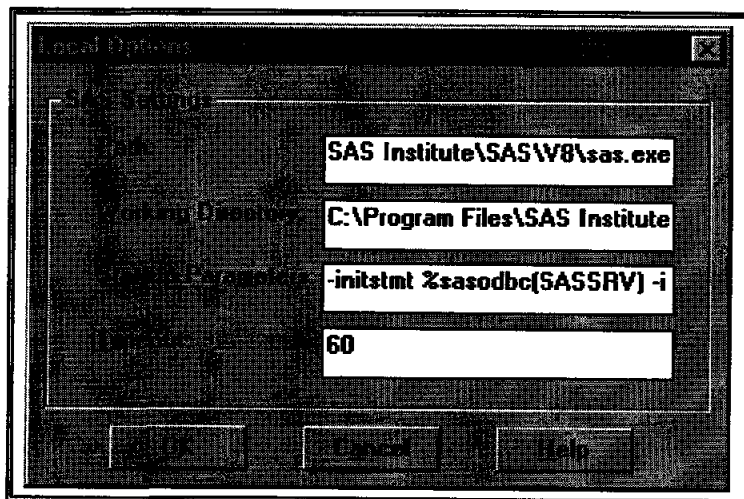


Figure "S" - Windows 3.1 ODBC Driver Configuration Servers (2 of 2)

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 T and U for an example of the screens displayed to define a dBASE data source. See Figures V and W 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 V for an example of **PROC SQL** using **PROMPT**. See Figure W for an example of **PROC SQL** using **DSN=**.

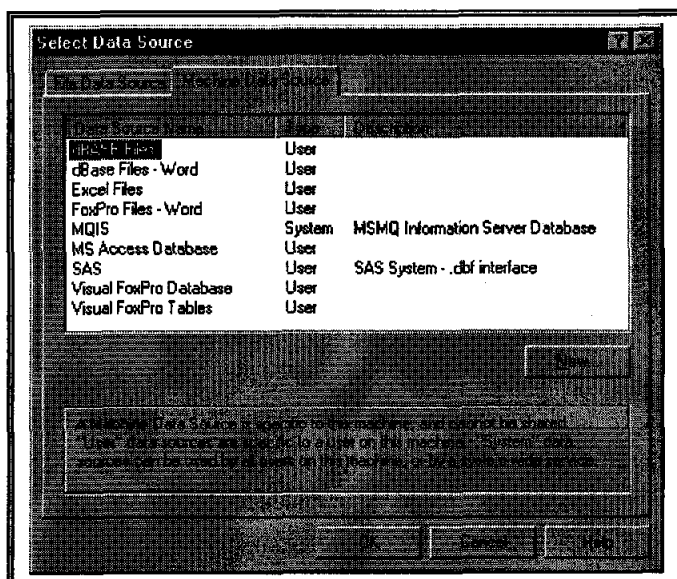


Figure "T" - SQL Pass-Through Facility CONNECT TO ... PROMPT / Data Source

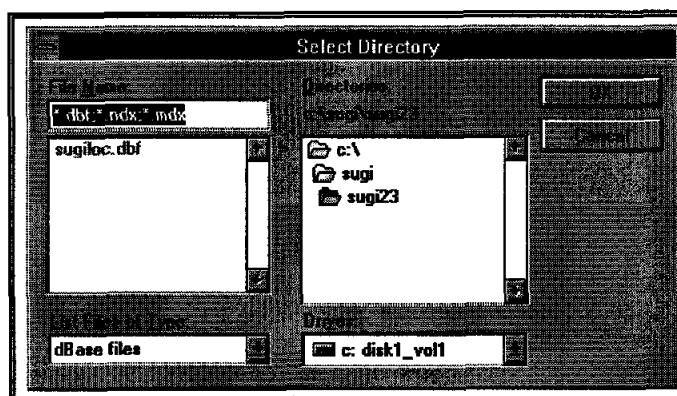


Figure "U" - SQL Pass-Through Facility CONNECT TO ... PROMPT / Select Directory

```

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 "V" - 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
                (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 "W" - SQL Pass-Through Facility using DSN

The ODBC examples covered so far have been based on Version 6 technology, although they have been run using Version 8. However, under Version 8, it has become even easier for a SAS routine to interface with ODBC by using the LIBNAME statement.

By using the ODBC keyword, the LIBNAME statement can be used to provide a straightforward interface between the external data source and the SAS System:

```

LIBNAME libref ODBC
        <ODBC-specific options>
        <general LIBNAME options>;

```

See Figure X for an example of using the LIBNAME statement with ODBC.

```

245 libname odbclib odbc
246     datasrc='SASClass Example';
NOTE: Libref ODBCLIB was successfully
      assigned as follows:
      Engine:      ODBC
      Physical Name: SASClass Example

247 proc print data=odbclib.confloc; run;
NOTE: There were 10 observations read
      from the dataset ODBCLIB.CONFLOC.
NOTE: PROCEDURE PRINT used:
      real time      0.10 seconds

```

Figure "X" - LIBNAME statement with ODBC

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.