# The Redmond to Cary Express - A Comparison of Methods to Automate Data Transfer Between SAS® and Microsoft Excel®

Michael T. Mumma, Westat, Rockville, MD

## ABSTRACT

Data transfer between SAS and Microsoft Excel is addressed by many SUG papers, and is continuously a hot topic within SAS-L. There is even a web page dedicated solely to this subject (Cram 1999). Many factors determine the 'best' (or simply viable) approach to performing this transfer. Five methods of automating this exchange are presented, including example SAS code. Methods discussed will include Dynamic Data Exchange (DDE), SAS/ACCESS® (PROC ACCESS, PROC IMPORT, and PROC SQL), and third-party software products under the Windows 9x operating systems. Some useful, yet seldom documented, Excel macro commands (issued via DDE) will be presented, as well as sources of documentation for each of the five methods.

## INTRODUCTION

Microsoft Excel is a user-friendly and popular spreadsheet application used to enter, display, and analyze data. Many techniques can be used to convert Excel data into SAS data sets. One common technique is to export Excel data to a text file, then use a SAS DATA step with an INFILE statement to read the text file into a SAS data set. Various import/export wizards can also be used. Refer to *Window by Window – Capturing Your Data Using the SAS System* for an excellent overview. Using these methods often involves many manual steps, which must be performed each time the Excel file is updated. The methods available to transfer data from Excel to SAS are limited when the objective is to automate the conversion process and eliminate user intervention. The operating system, Excel version, SAS version, SAS software licensed, and data characteristics will determine which methods are viable solutions. The 'best' approach should also take into consideration programming requirements, ease of code maintenance, and frequency of updates to the Excel data.

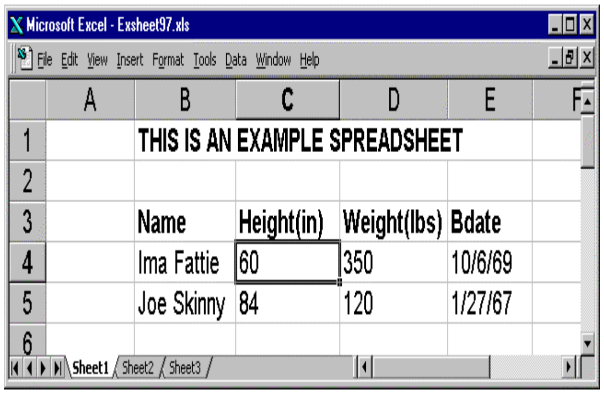
The purpose of this paper is to:

1) compare five different methods of automating the transfer of Excel data to SAS data sets. These include:
- Dynamic Data Exchange (DDE)
- PROC ACCESS
- PROC IMPORT
- PROC SQL
- Third party software (e.g. DBMS/COPY®, DBMS/Engines®, Stat/Transfer®) ;

2) provide example SAS code utilizing each method; and

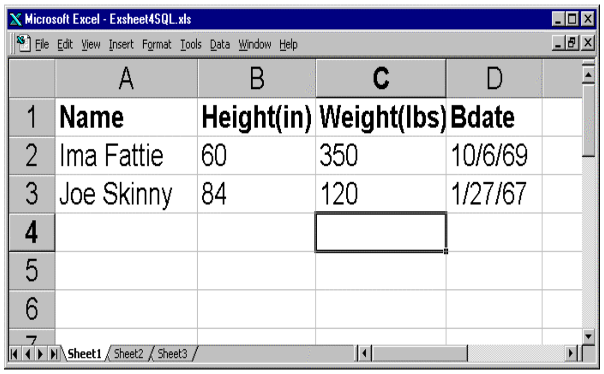
3) provide references for documentation of each method.

Within example SAS code, capital lettering denotes SAS keywords, where lowercase lettering is used for data set names, variable names, filerefs (file references), and librefs (SAS library references). A simple Excel spreadsheet with four columns (a character with embedded spaces, two numerics, and a date) and two rows is used as model data. Shown in Figure 1 is the spreadsheet Exsheet97. An identical file, Exsheet95, is saved in Excel 5.0 format. A third file, Exsheet4SQL is also used as a spreadsheet model for the PROC SQL example. Note that extraneous text, such as titles, have been removed (Figure 2).


Microsoft Excel - Exsheet97.xls
File Edit View Insert Format Tools Data Window Help
Sheet1 Sheet2 Sheet3

|   | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 |  | THIS IS AN EXAMPLE SPREADSHEET |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  | Name | Height(in) | Weight(lbs) | Bdate |
| 4 |  | Ima Fattie | 60 | 350 | 10/6/69 |
| 5 |  | Joe Skinny | 84 | 120 | 1/27/67 |
| 6 |  |  |  |  |  |

Figure 1 - Example Spreadsheets Exsheet97.xls/Exsheet95.xls


Microsoft Excel - Exsheet4SQL.xls
File Edit View Insert Format Tools Data Window Help
Sheet1 Sheet2 Sheet3

|   | A | B | C | D |
|---|---|---|---|---|
| 1 | Name | Height(in) | Weight(lbs) | Bdate |
| 2 | Ima Fattie | 60 | 350 | 10/6/69 |
| 3 | Joe Skinny | 84 | 120 | 1/27/67 |
| 4 |  |  |  |  |
| 5 |  |  |  |  |
| 6 |  |  |  |  |

Figure 2 - Example Spreadsheet Exsheet4SQL.xls

Space does not allow for full discussion of the complete syntax of all statements and procedures within example code. However, an attempt will be made to point out areas that can be especially tricky. There are many intricacies inherent to each of these methods. The reader is advised to refer to other sources, including SAS-L, to resolve these issues.

## DYNAMIC DATA EXCHANGE (DDE)

Dynamic data exchange allows SAS to establish a connection with a running Windows application similar to a client-server relationship. SAS (the client) can request data from and issue commands to the application (the server) using INPUT and PUT statements. In order to retrieve data from Excel using DDE, the workbook containing the Excel data must be open.

For the initial DDE connection to be established, Excel must be running. Either a X or %SYSEXEC (used in SAS macro code) statement is used to invoke Excel from within the SAS program. Note the NOXWAIT and NOXSYNC options must be set. Getting the correct X statement to open Excel can be tricky. Be careful that embedded spaces do not exist within the X statement, for example “Program Files”. If necessary, use the truncated DOS name to point to the executable file that invokes Excel. A SLEEP function is then used to allow Excel time to completely open before SAS begins to send DDE commands.

A specialized FILENAME statement is used to create the DDE link to the Excel application. Any statements written to the fileref (in the example code, the fileref is *commands*) will be sent to Excel via DDE. Make sure that within Excel, the ‘Ignore other Applications’ option is not checked (under **Tools**..**Options**..**General**). A PUT statement issues the ‘Open’ Excel macro command from SAS to Excel in order to open the Excel workbook. *[Note: When opening spreadsheets that contain VBA macros, a message appears, warning the user and prompting whether they want to continue. Macro protection within Excel may be (cautiously) turned off to prevent this warning message and eliminate the need for user intervention.]*

Once the workbook is open, another specialized FILENAME statement is used to reference a range of cells in the worksheet within the open workbook. The text within quotation marks is referred to as the DDE triplet. For Excel, the triplet has the syntax ‘EXCEL|[myfile.xls]mysheet!R*x*C*y*:R*w*C*z*’ where *x*,*y*,*w* and *z* are integers used to denote the range of cells within Excel. (When using DDE to interface with other applications, the DDE triplet may have a slightly different structure.) Data within these cells is then available for input using a DATA step with an INFILE statement, similar to reading a text file. Note the NOTAB option on the FILENAME statement following the DDE triplet. The DSD and MISSOVER options, with a tab (‘09’x) as a delimiter, is then used on the INFILE statement to read data from the spreadsheet. This is done to correctly input strings containing embedded spaces (in this case the *name* variable). See the *SAS Companion for the Microsoft Windows Environment, Version 6, Second Edition* for an overview, or Schreier (1998) for an excellent discussion on how the NOTAB option operates. Also note the use of INFORMAT and FORMAT statements to read date values (which are passed as strings) and prevent truncation of character variables. A QUIT command, within a _NULL_ DATA step, is then sent to quit Excel.

```
OPTIONS NOXWAIT NOXSYNC;
 X "C:\Progra~1\Micros~4\Office\EXCEL";
DATA _NULL_;
 X = SLEEP(5);
RUN;
FILENAME commands DDE 'EXCEL|SYSTEM';
DATA _NULL_;
  FILE commands;
PUT '[OPEN("C:\nesug99\exsheet97.xls")]';
RUN;

FILENAME myssheet DDE
'Excel|[exsheet97.xls]Sheet1!r4c2:r50c5' NOTAB;

DATA mysasset;
  INFORMAT name $25. bdate MMDDYY8. ;
  FORMAT name $25. bdate MMDDYY8. ;
  INFILE myssheet DLM='09'X DSD MISSOVER;
  INPUT name height weight bdate;
  IF name NE ' ';
RUN;

DATA _NULL_;
  FILE commands;
  PUT '[QUIT]';
RUN;
```

There are some disadvantages to using DDE as a means of transferring data between Excel and SAS. First, as previously mentioned, the application (in this case Excel) must be running before a DDE connection can be established. This can significantly slow down the conversion process, especially if long sleep times are required. Also, because numeric values are passed as formatted strings, loss of precision can occur.

Another disadvantage to using DDE is that the documentation for the Excel 4 macro commands sent to Excel (you cannot send VBA commands to Excel from SAS) is very poor and difficult to find. I was lucky enough to find a copy of the ‘Excel 4.0 Function Reference’ which contains most of the available macro commands. Documentation for using DDE within SAS seems to be scattered among SAS technical notes (SAS TSD #325, Note F885) and *SAS Companion for the Microsoft Windows Environment*. Often, examples of the most useful DDE commands are not included. The best resources found for answering more in-depth questions are often SUG papers (see Schreier (1998), Kuligowski (1999), Asam and Usavage (1997), Lee (1997)) and SAS-L. Below are examples of useful Excel macro commands as they would appear within the SAS code.

```
DATA _NULL_;
 FILE commands;
* Opens a file as read-only;
 PUT '[OPEN("c:\mydir\myfile.xls",0,TRUE)]';
* Saves the current file;
PUT '[SAVE]';
* Saves current file with new name;
PUT '[SAVE.AS("c:\mydir\myfile.xls")]';
* Deletes a file;
 PUT '[FILE.DELETE("c:\mydir\myfile.xls")]';
* Minimizes the Excel application;
PUT '[APP.MINIMIZE()]';
* Maximizes the Excel application;
PUT '[APP.MAXIMIZE()]';
```

```
   * Closes the current workbook;
   PUT '[CLOSE]';
   * Quits Excel;
   PUT '[QUIT]';
   RUN;
```

SAS will not resolve macro variables enclosed within single quotes. Therefore, if SAS macro variables are to be used within PUT statements, a slightly different syntax is required. A single set of double quotes should enclose all text after the PUT statement. Then, *two* sets of double quotes should enclose the appropriate command parameters. For example, if the file reference for the Excel spreadsheet is a macro variable, the following command could be used to open this file before importing the data:

```
   %LET newfile = c:\nesug99\exsheet4sql.xls;
   DATA _NULL_;
     FILE commands;
   * OPEN command with macro variable ;
     PUT "[OPEN(""&newfile"")]";
   RUN;
```

There are some ways around knowing the antiquated Excel macro commands. One way is to create a VBA macro within Excel. This can easily be done using the macro recorder and/or Visual Basic editor. For example, a VBA macro can be created which saves an Excel file as some form of text file (i.e. tab delimited, column delimited, csv, etc.). A single DDE command can then be issued from SAS to run the stored VBA macro. Once the text file is created, SAS can read in the text file using a standard DATA step with INFILE statement. The command to run a stored VBA macro is as follows:

```
  PUT '[RUN("myfile.xls!MyMacro",False)]';
```

where MyMacro is the name of the created VBA macro within myfile.xls. Note that the macro names are case sensitive. (See SAS note F885 for more details.)

The most significant advantage of DDE is that it is the only method to automate the import of Excel files without using another software product, such as SAS/ACCESS or DBMS/COPY. Because the range of data is defined explicitly within the FILENAME statement, the Excel spreadsheet can contain extraneous information, such as titles, column headers, comments, etc. However, explicitly defining the range within the SAS code requires that the code be modified if more rows or columns are added to the spreadsheet. In the above example, this problem was somewhat circumvented by defining a range much larger than the current data. A subsetting IF statement is then used to eliminate blank records. In this way, if more records are added to the spreadsheet, no modification to the SAS code is required.

Another advantage to using DDE is that once familiar with DDE concepts and techniques (e.g. X statements to invoke applications, DDE triplets), DDE can later be used to interface with other Windows applications such as MS Word (see Bross (1997)) and MS Access (see Asam and Usavage (1997)). In addition, the ability to issue a DDE command to run a stored VBA macro can be especially powerful.

## SAS/ACCESS

SAS/ACCESS software provides a number of methods to transfer data between various database management systems (DBMS), PC file formats (including Excel), and the SAS system. Within SAS/ACCESS, three procedures can be used to import Excel data: 1) PROC ACCESS, 2) PROC IMPORT (SAS version 7), and 3) PROC SQL. Various components of SAS/ACCESS must be installed and licensed at your site in order to use these methods. [*Note: In order to determine the SAS components currently licensed at your site, submit the following SAS code:* `PROC SETINIT; RUN;` *Installed components will be displayed in the log.*]

### PROC ACCESS

To use PROC ACCESS to import Excel data, the SAS/ACCESS to PC Files Formats® software must be installed and licensed at your site. The structure of the ACCESS procedure is slightly unconventional. To access external data, two types of *descriptor* files must be created: an access descriptor and a view descriptor. The access descriptor contains information such as file type (i.e. Excel), file name, file location, worksheet name, column names, and data range. The view descriptor is created from an access descriptor and determines which variables (columns) are to be selected. Both descriptor files can be created within one PROC ACCESS procedure. However, a view descriptor can also be created from an existing access descriptor in a separate procedure. Access and view descriptors can be stored as permanent files by using a permanent LIBNAME statement just like a SAS data set. The view descriptor can then be used in any subsequent procedure or DATA step. See Engle (1997) for an excellent discussion of creating access and view descriptors. Below is an example of creating an access and view descriptor in one procedure:

```
   LIBNAME my_lib "c:\nesug99";

   PROC ACCESS DBMS=EXCEL;
   /* Creates the access descriptor */
    CREATE my_lib.my_acc.ACCESS;
   /* Required-Could also use a fileref here */
    PATH='c:\nesug99\exsheet95.xls';
    SCANTYPE = YES;
    WORKSHEET = 'Sheet1';
   /* Optional-Could also used a named range */
    RANGE 'b3..e5';
    GETNAMES = YES;
    SKIPROWS = 0;
    RENAME name = fullname;
    TYPE NAME = C;
    MIXED = NO;
```

```
/* Creates the view descriptor */
 CREATE my_lib.my_view.VIEW;
 SELECT ALL ;
 LIST ALL;
 RENAME bdate = birthday;
RUN;
PROC PRINT DATA = my_lib.my_view; RUN;
```

The first CREATE statement creates the permanent access descriptor *my_acc*. The WORKSHEET statement identifies which worksheet contains the data to import (default is Sheet1). The RANGE statement selects the cells containing Excel data to import. Note in the example that the range includes column headers. A named range (created within the Excel worksheet) could also be used. Also note the syntax of specifying a range of cells is different from that used in DDE. The GETNAMES= statement determines whether SAS generates variable names from column names in the Excel file's first row of data. If GETNAMES=YES, SAS also sets the SKIPROWS value to 1. If GETNAMES=NO, or if the column names are not valid SAS names, PROC ACCESS uses Var0, Var1, etc. The SCANTYPE= option (which must precede editing statements) and the TYPE and MIXED= options determine how SAS assigns data types (i.e. character or numeric) and formats to each variable. (By default, formats are assigned based on the Excel formats found in the first row of data.) *See SAS/ACCESS Software for PC File Formats, Version 6, First Edition* for a complete description of how these options work.

If this code leaves you slightly (or completely) confused, take heart. SAS release 6.12 contains an Import Wizard, which generates PROC ACCESS code through a series of interactive windows (selecting Import from the **File** menu on the SAS toolbar starts the Import Wizard). After the import has been completed, simply recall the program to see the PROC ACCESS code. See Kuligowski (1999) for a thorough discussion on using the Import Wizard.

One of the biggest drawbacks to PROC ACCESS is Excel files must be saved in either Microsoft Excel 4 or 5/7 format. Neither the ACCESS nor DBLOAD procedures interface with files in Excel 97 (Version 8) format. An add-on product is available from the SAS Institute, which allows the user to use the Import Wizard to read Excel 97 files. This add-on product does not, however, generate or allow use of PROC ACCESS (or PROC DBLOAD) statements, and therefore the conversion cannot be automated (see http://www.sas.com/service/techsup/unotes/V6/F642.html). One workaround to this problem is to save the Excel 97 file in the Excel 97/5 dual file format by using the **Save As** command on the **File** menu. PROC ACCESS can then interface with the dual format Excel file. Another solution is to upgrade to SAS version 7 and use PROC IMPORT (discussed below).

Another problem with PROC ACCESS is that a specified range of cells cannot contain a block of empty cells. This is important if records are regularly added or removed from the spreadsheet. One solution is to not specify any data range (the default range is the entire worksheet). The SKIPROWS= option can then be used to specify the first row of data if extraneous data, such as titles, are used. However, the GETNAMES= option must be set to NO since GETNAMES=YES automatically sets SKIPROWS= value to 1. Variable will then need to be renamed from the default Var0, Var1, etc. convention using a RENAME statement. This allows modifications to the spreadsheet without subsequent modifications to the SAS code.

**PROC IMPORT**

SAS version 7 contains new IMPORT and EXPORT procedures, which like PROC ACCESS, perform conversion of many file formats, including Excel files, to SAS data sets. A license for SAS/ACCESS for PC File Formats is still required. The code is fairly simple, and somewhat similar in structure to PROC ACCESS. However, unlike PROC ACCESS, the creation of access and view descriptors is not required. Also, PROC IMPORT creates a SAS data set instead of a view descriptor. PROC IMPORT code can also be generated using the Import Data wizard provided with SAS v 7. Below is the PROC IMPORT example code.

```
FILENAME mysheet "c:\nesug99\exsheet97.xls";
PROC IMPORT DATAFILE = mysheet
 OUT = mysasset
 DBMS=EXCEL97
 REPLACE ;
 GETNAMES = YES ;
 SHEET = sheet1;
/* Cannot use absolute range w/Excel97 files */
 RANGE = myrange;
RUN;
```

The REPLACE statement is required in order to overwrite an existing SAS data set. The DBMS= statement determines the type of file to import. For Excel files, the choices are EXCEL, EXCEL4, EXCEL5, and EXCEL97. *[Note: PROC IMPORT can recognize the difference between Excel Version 4 and 5 spreadsheets when you use the extension .XLS, regardless of whether you specify DBMS=EXCEL, DBMS=EXCEL4, or DBMS=EXCEL5. However, you must specify DBMS=EXCEL97 to import Excel 97 files.]* GETNAMES= statement determines whether SAS generates variable names from column names in the first row of data. The SHEET= statement, like the WORKSHEET statement in PROC ACCESS, identifies the Excel worksheet that contains the data to import. If you do not specify SHEET=, PROC IMPORT defaults to the first spreadsheet in the file.

The RANGE statement is identical to RANGE in PROC ACCESS with one (unfortunate) exception. When importing Excel 97 data, the data range definition can no longer be defined within the SAS code using an *absolute* range (i.e. R3..C5). Instead, a *range-name* must be created within the Excel worksheet. This can be done easily enough. (Highlight the range of cells. Then select **Insert**..**Name**..**Define** and enter a name alias for the range of cells.) In the example, range-name is *myrange*. The range-name is then used in the RANGE statement. However, not having the ability to define an absolute range within the SAS code can be cumbersome when updates or modifications to the SAS code or spreadsheet are required.

The range-name requirement aside, PROC IMPORT does have some distinct advantages over PROC ACCESS. Most importantly, PROC IMPORT can interface with Excel 97 files, where PROC ACCESS cannot. In addition, unlike PROC ACCESS, the defined range-name can contain empty cells. PROC IMPORT can also access data whether the Excel file is open or closed (PROC ACCESS requires the spreadsheet file to be closed). Also, the creation of access and view descriptors is not necessary.

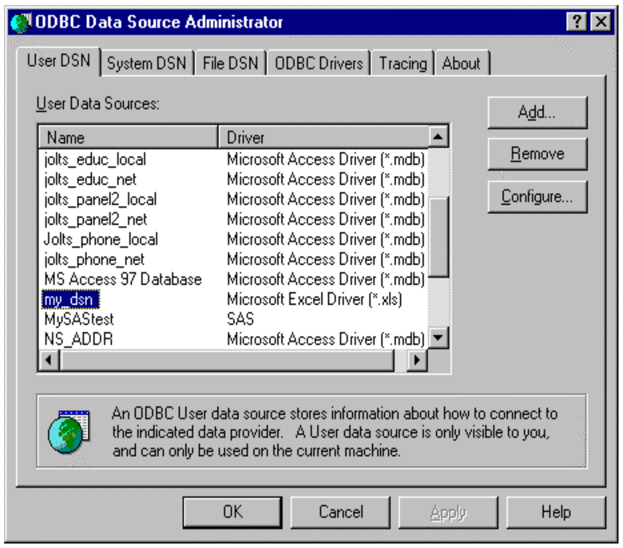
The improvements implemented in the SAS/ACCESS IMPORT procedure greatly improve functionality and ease of programming. PROC ACCESS has a slight convenience in that RENAME, DROP, and FORMAT statements can be included within one procedure (see *SAS/ACCESS Software for PC File Formats* for other optional PROC ACCESS statements not included in this discussion.) However, the RENAME statement may not be as necessary with PROC IMPORT because truncation of variable names is less of a concern due to version 7 support of 32 character variable names.

Using PROC ACCESS/PROC IMPORT has a number of advantages over other methods of importing Excel data. Excel date and time values are automatically converted to SAS dates and times if appropriately formatted within the Excel spreadsheet. Therefore, no LENGTH or INFORMAT statements are required in order to prevent truncation of character variables or perform data/time conversions. See 'Datetime Conversions in the ACCESS Procedure' in Chapter 8 of *SAS/ACCESS Software for PC File Formats* for more details.

**PROC SQL (SQL PASS-THROUGH FACILITY)**

To use the SQL procedure pass-through facility to import Excel data, the SAS/ACCESS Interface to ODBC® software must be installed and licensed at your site. Also, the appropriate ODBC drivers for Excel must be installed on the PC running the SAS program.

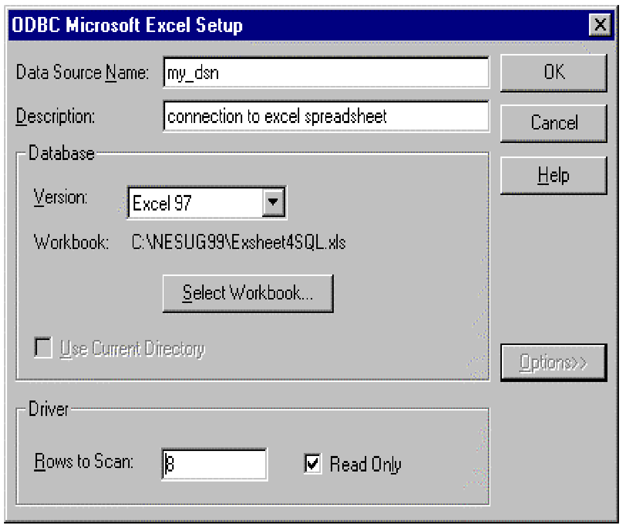
When using PROC SQL, the SAS/ODBC interface does not obtain data directly from a data source (in this case the Excel spreadsheet). Instead, SAS interfaces with an ODBC manager, which then interfaces directly with the Excel file. Therefore, the ODBC manager must have the name and path of the Excel file from which to retrieve data. An alias, or *data source name* (DSN), for the Excel file can be created in Windows 95/98 using the ODBC manager. A data source name can then be assigned to a particular Excel spreadsheet. Shown in Figures 3 and 4 are the screens used to create the DSN (*my_dsn*) used in the SQL example. See Kuligowski (1999) or Li (1999) for more details on how to create a DSN alias using the Windows ODBC manager.



| Name | Driver |
|---|---|
| jolts_educ_local | Microsoft Access Driver (*.mdb) |
| jolts_educ_net | Microsoft Access Driver (*.mdb) |
| jolts_panel2_local | Microsoft Access Driver (*.mdb) |
| jolts_panel2_net | Microsoft Access Driver (*.mdb) |
| Jolts_phone_local | Microsoft Access Driver (*.mdb) |
| jolts_phone_net | Microsoft Access Driver (*.mdb) |
| MS Access 97 Database | Microsoft Access Driver (*.mdb) |
| my_dsn | Microsoft Excel Driver (*.xls) |
| MySAStest | SAS |
| NS_ADDR | Microsoft Access Driver (*.mdb) |

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

Figure 3 - ODBC Administrator Screen



Data Source Name: my_dsn
Description: connection to excel spreadsheet
Version: Excel 97
Workbook: C:\NESUG99\Exsheet4SQL.xls
☐ Use Current Directory
Rows to Scan: 8
☑ Read Only

Figure 4 - ODBC Microsoft Excel Setup Screen

Once the DSN is created, Excel data can be read using PROC SQL. The basic PROC SQL syntax is as follows:

```
 PROC SQL;
   CONNECT TO ODBC (DSN=my_dsn);
   CREATE TABLE mysasset AS
   SELECT * FROM CONNECTION TO ODBC
         (SELECT * FROM "Sheet1$");
DISCONNECT FROM ODBC;
QUIT;
```

An alternative to creating a DSN using the ODBC manager is to pass the DSN to the ODBC manager using the COMPLETE= option. The text inside the double quotes following this option can be a DBMS-specific connect-string, which specifies the file, ODBC drivers, and other options used to establish a connection. The COMPLETE= option can also be used to override options within an existing DSN, such as the Excel file from which to retrieve data, while keeping other DSN parameters unchanged (see example code below). One advantage to using the COMPLETE= option is that ODBC parameters, such as the Excel file, can be changed dynamically within the SAS code using macro variables. Also, because DSNs do not have to be created, the SAS code can be easily transported among different machines.

PROC SQL assigns variable names based on the first row of data within the spreadsheet. By default, character variables are given a length and format of character 200. One way to override the default is to use a combination of an ALTER and FORMAT statement within the SQL code. Another way around the 200-length of character variables is to SET the PROC SQL created data set into another data set using a DATA step with LENGTH statements.

Provided below is a more robust example of retrieving Excel data using PROC SQL using the COMPLETE option, and with ALTER, FORMAT, SELECT and WHERE expressions included.

```
PROC SQL;
  CONNECT TO ODBC
(COMPLETE="DSN=my_dsn;DBQ=c:\nesug99\newS
QLfile.xls");
 CREATE TABLE mysasset AS
 SELECT bdate, name format=$30., weight_l
  FROM CONNECTION TO ODBC
      (SELECT * FROM "Sheet1$")
  WHERE weight_l > 200;
  ALTER TABLE mysasset
  MODIFY name char(30) ;
DISCONNECT FROM ODBC;
QUIT;
```

Because Excel does not support writing data ranges to the ODBC drivers, there is no way to define a data range with the Excel file when using PROC SQL. Consequently, no extraneous text, such as titles, can be in the spreadsheet (shown in Figure 2). However, because no range is specified, modifications are not necessary when columns or rows are added to the Excel file. A definitive disadvantage, however, is that if the data set contains many character variables, each variable must be reformatted within the SAS code if variable length is to be less than 200.

When retrieving Excel data, the spreadsheet can be open or closed. Also, PROC SQL allows use of most of the features and structure of the SQL language. For example, SELECT expressions will allow only certain variables to be retrieved and the WHERE expression allows observations to be easily filtered and merged. See the *SAS Guide to the SQL Procedure, Usage and Reference, Version 6* for the complete syntax.

## THIRD PARTY SOFTWARE

A number of software products perform data conversion between different data formats, including SAS data sets. Some of the best examples are DBMS/COPY, DBMS/Engines (both available from Conceptual Software, Inc., http://www.conceptual.com) and Stat/Transfer (available from Circle Systems, http://www.stattransfer.com). Smith and Carpenter (1999) provide a detailed discussion on the use of these products. The reader is encouraged to try these products (both companies provide trial versions on their websites) and/or see Hilbe (1996) for a review.

DBMS/COPY can be run interactively to develop a program to perform a data conversion. This program can then be executed in batch mode through a DOS command. Because SAS can issue DOS commands to the OS, a stored DBMS/COPY program can be executed to automatically convert an Excel spreadsheet into a SAS data set. A SAS statement to execute a stored DBMS/COPY program is shown below:

```
X "c:\progra~1\dbmscopy\dbmswi32 PLUS
c:\nesug99\dbmscode.prg";
```

Stat/Transfer also has the capability to be run in batch mode by issuing a DOS command. However, the user interface cannot be used to create a program to perform the conversion. Instead, DOS commands must be issued from SAS using X statements. SAS code to convert the example spreadsheet Exsheet4SQL to a SAS data set using Stat/Transfer is shown below. The Exsheet4SQL file is used because Stat/Transfer does not allow for extraneous data within the spreadsheet. Prior to issuing the DOS command to perform the conversion, the current directory must be changed to where Stat/Transfer is installed. Also note that the transfer creates a permanent SAS data set, and therefore must be referenced within SAS by using the two part naming convention.

```
X 'CD C:\PROGRAM FILES\STATTRANSFER5';
X 'ST C:\nesug99\exsheet4sql.xls
c:\nesug99\mysasset.sd2 /Y';

LIBNAME here 'c:\nesug99';
PROC PRINT DATA = here.mysasset;
RUN;
```

With DBMS/COPY, the interactive menu makes it easy to format, rename and drop variables, and sort and filter records. Also, DBMS/COPY can convert Excel data to a number of different SAS formats, including version 6.08-6.12 (*.sd2), SAS for PC/DOS (*.ssd), and version 5 and 6 transport files (*.v5x and *.v6x). However, when using DBMS/COPY, a range containing empty cells cannot be selected. Therefore, the DBMS/COPY program will have to be revised when rows or columns are added or removed from the Excel file. Also, the Excel file must be closed during data conversion. When using Stat/Transfer, the Excel workbook can be either open or closed. However, no data range can be specified, and therefore no extraneous text is allowed. In addition, variables cannot be renamed or dropped, and records cannot be conditionally selected. However, because no program is created, updates are not required.

Another software product, DBMS/Engines, can convert Excel spreadsheets (and many other DBMS formats) to SAS data sets. However, unlike the previous two products, X statements are not required. Access to the Excel file is accomplished by specifying a database-specific engine name within a LIBNAME statement. For example, the LIBNAME statement below enables SAS to access the data in the example Excel 95 spreadsheet (engine name is *DBEXCEL5*). The SPREAD= options specifies the worksheet, data range, and row(s) in which the variable names are located. In this example, the data range is from row 4 to 100, column B to E, on worksheet 1 (i.e. the first worksheet in the workbook), with variable names located on row 3. Note that the data range contains empty cells (see Figure 1).

```
LIBNAME my_lib DBEXCEL5 'c:\nesug99' my_file
= "exsheet95" SPREAD ='1,4,100,B,E,3,3';

PROC PRINT DATA = my_lib.my_file;
RUN;
```

One advantage to DBMS/Engines is that a SAS data set does not actually need to be created. The Excel data can be accessed directly through the two-level data set naming convention whenever used in a PROC or DATA step (PROC PRINT in the example.) Because the data range can contain empty cells, no updates to the LIBNAME statement would be needed when rows are added. Record and variable selection can be accomplished using traditional DATA step IF and DROP/KEEP statements, respectively. The syntax on the LIBNAME options can be slightly cumbersome. However, once the appropriate LIBNAME statement is created, the conversion from Excel to SAS (and back) is transparent. *(Note: A PATH statement must be added to the CONFIG.SAS file in order for SAS to know where the conversion engines are located.)*

Because these three software products support almost all Excel formats (2, 3, 4, 5, and 97), problems with reading different versions of Excel is not a concern as with SAS/ACCESS. SAS programming requirements are minimal (only the appropriate X or LIBNAME statement is needed) and SAS code updates would not be necessary after modifying the Excel file. Another significant advantage to using third party products may be cost, especially if SAS/ACCESS software is not already licensed. At the time of writing, a non-academic individual copy (Windows version) of DBMS/COPY or DBMS/Engines was $295, and Stat/Transfer was $249. For a general comparison, the cost to lease a single non-academic user license for one year of either SAS/ACCESS for PC File Formats or SAS/ACCESS Interface to ODBC is $1,060 or $1,135 (the lower price if base SAS was licensed prior to 1997). In addition to cost savings, these products can be extremely powerful due to the many different data formats, in addition to SAS data formats, that are supported.

## CONCLUSION

Choosing the 'best' method to import Excel data into SAS depends on many factors. Various aspects of the five methods discussed are summarized in Table 1 on the following page. With the exception of PROC SQL, each method also has a way to write data from SAS to Excel, should this be desired. (Perhaps a subsequent paper shall address this topic.) Other factors, such as performance issues and the amount of data to be converted, should also be taken into consideration. However, there is no substitute for trial and error. Often the best way to decide is to simply experiment with as many methods as possible until a suitable solution is found.

## REFERENCES

Asam, Ellen L. and Donna Usavage (1997) “Using Dynamic Data Exchange Within SAS Software to Directly Access Data From Microsoft Windows Applications” in *Proceedings of the Tenth Annual Northeast SAS Users Group Conference*. pp. 296-297.

Bross, Dean (1997) “Preparing Final Reports Using DDE to Link SAS Software and Microsoft Word” in *Proceedings of the Tenth Annual Northeast SAS Users Group Conference*. pp. 580-589.

Cram, Donald P. (1999) “Excel 2 SAS and Back Webpage”: http://www-leland.stanford.edu/class/gsb/excel2sas.html.

Engle, Eric W. (1997) “SAS/ACCESS Software: Proc Access a Quick Start Guide” in *Proceedings of the Tenth Annual Northeast SAS Users Group Conference*. pp. 6-9.

Hilbe, Joseph (1996) “Windows File Conversion Software” in American Statistician, August 1996. pp. 268-270.

Kuligowski, Andrew T. (1999) “Advanced Methods to Introduce External Data into the SAS System" in *Proceedings of the Twenty-Fourth Annual SAS Users Group International Conference*. Paper 53 pp.345-354.

Lee, Han-li (1997) “Small Chat Between Jim and Bill Under Windows using DDE” in *Proceedings of the Tenth Annual Northeast SAS Users Group Conference*. pp. 53-59.

Li, Leiming (1999) “A Process for Automatically Retrieving Database Using ODBC and SAS/ACCESS SQL Procedure Pass-Through Facility” in *Proceedings of the Twenty-Fourth Annual SAS Users Group International Conference*. Paper 89 pp. 567-570.

SAS Institute Inc. (1989) *SAS Guide to the SQL Procedure: Usage and Reference, Version 6, First Edition* Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1994) *Getting Started with SAS/ACCESS Software*. 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. (1996) *SAS Companion for the Microsoft Windows Environment, Version 6, Second Edition* Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1996) “Technical Support Document #325 – The SAS System and DDE” http://ftp.sas.com/techsup/download/technote/TS325.ps.

SAS Institute Inc. (1996) “Technical Support Document TS-589B – Importing Excel Files to SAS Data Sets” http://ftp.sas.com/techsup/download/technote/ts589b.txt.

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

SAS Institute Inc. (1999) “SAS Note F885: How to use DDE to execute Visual Basic Macros from SAS” http://www.sas.com/service/techsup/unotes/V6/D432.html.

Schreier, Howard (1998) “Getting Started with Dynamic Data Exchange” in *Proceedings of the Sixth Annual Southeastern SAS Users Group Conference*. pp. 207-215.

Smith, Richard and Art Carpenter (1999) “The Use of External Software to Import Data into the SAS System” in *Proceedings of the Twenty-Fourth Annual SAS Users Group International Conference*. Paper 222. pp.1310-1313.

## ACKNOWLEDGEMENTS

Thanks to T. Hagerty, B. Hart, K. Hubbell, J. Kuhn, M. Rhoads, and I. Whitlock for reviews of this manuscript.

## AUTHOR CONTACT

Comments, questions, (and corrections!) are valued and encouraged. Contact the author at:

Michael T. Mumma
Westat
1650 Research Blvd.
Rockville, MD 20850
Phone: (301) 517-8089
Email: mummam1@westat.com

## TRADEMARKS

Table 1 - Summary of Five Methods to Import Excel Data into SAS

| | **Dynamic Data Exchange (DDE)** | **SAS/ACCESS® PROC ACCESS** | **SAS/ACCESS® V7 PROC IMPORT** | **SAS/ACCESS® PROC SQL** | **Third Party Software** |
|---|---|---|---|---|---|
| **Additional Software Required** | None | SAS/ACCESS for PC File Formats® | SAS/ACCESS for PC File Formats® | SAS/ACCESS Interface to ODBC®<br>Excel ODBC drivers | Separate product |
| **Programming Requirements** | -DATA step with INPUT<br>-PUT statements to issue DDE commands<br>-X statement to invoke Excel | -PROC ACCESS (Creation of access and view descriptors) | -PROC IMPORT | -PROC SQL statement<br>-DSN configuration | -X statement(s)<br>-LIBNAME statement |
| **Excel status during transfer** | Open | Closed | Open/Closed | Open/Closed | DBMS/COPY®–Closed<br>DBMS/Engines®-Closed<br>Stat/Transfer®-Open /Closed |
| **Excel formats supported** | Excel 4, 5, 7, 97 | Excel 4, 5, 7 | Excel 4 , 5, 7, 97 | Installed ODBC drivers | Product dependent (Most supported) |
| **Documentation/SUG papers** | -*SAS Companion for the Microsoft Windows Environment*<br>-SAS TSD #325, Note F885<br>-Kuligowski (1999)<br>-Lee (1997)<br>-Schreier (1998)<br>-Web site (Cram 1999) | -*Getting Started with SAS/ACCESS Software*<br>-Engle (1997) | - v7 Online Documentation | -*Getting Started with SAS/ACCESS Software*<br>- *SAS Guide to the SQL Procedure*<br>-Kuligowski (1999)<br>-Li (1999) | -Software documentation<br>-Smith and Carpenter (1999) |
| **SAS Version** | 6.08 and later | 6.11 and later | 7.0 | 6.10 | Product dependent |
| **Analogous Method to Convert SAS to Excel** | DDE | PROC DBLOAD | PROC EXPORT | None | DBMS/COPY®–new program<br>DBMS/Engines®-DATA step<br>Stat/Transfer®-new X statement |
| **Data Range** | Defined in DDE triplet | Range statement optional<br>Absolute or named range | Range statement optional<br>Named range only | No option<br>No extraneous text | Product dependent |