SAS® Tutorial Session - Reading and Writing Data to External Files

Dr. Ronald Cody - Robert Wood Johnson Medical School

New SAS users are often confused by the different ways SAS software can read and write data to external files. This is due to the fact that SAS programs can read and write many different types of data files. For example, simple ASCII files are read with INFILE and INPUT statements, while SAS data sets use two-level SAS data set names and do not require INPUT statements. This tutorial session will discuss several ways that SAS software can read and write a variety of data types. The use of temporary and permanent SAS data sets is discussed along with the advantages and disadvantages of each. Included in this discussion are the importing and exporting of data to Lotus® spread sheets and dBASE® files.

EXAMPLE 1. Reading Data that is Part of the SAS Program Itself.

This is the simplest way that a SAS program can read data. The data lines are incorporated in the program itself, following a CARDS statement.

```
DATA EX1;
INPUT GROUP $ X Y Z;
CARDS;
CONTROL 12 17 19
TREAT 23 25 29
CONTROL 19 18 16
TREAT 22 22 29
PROC MEANS N MEAN STD STDERR MAXDEC=2;
   TITLE 'MEANS FOR EACH GROUP';
   CLASS GROUP;
   VAR X Y Z;
RUN;
```

In this first example, we use an INPUT statement to tell the program the names we want to associate with the data values. Notice that we did not indicate any column or format specification in this statement, resulting in what is called a list directed read. This form of the INPUT statement allows us to list our data values separated by one or more blanks. The INPUT method used, column specification, formats, pointers, etc. will not change any of our examples, so for the most part, simple list input is used. The CARDS statement tells the program that the data lines will follow. The word CARDS is obviously a throwback to the old days when many of us used actual punch cards in a deck to be read into a card reader. CARDS meant that the data cards were to follow. (An aside: I gave a talk to a group of 9th graders the other day and used a SAS program to demonstrate a point. Just out of curiosity, I asked if any of them knew what data cards were. None did! I started to feel very old.) You might ask yourself, how does the program know when the data lines end and the remainder of the SAS program begins? How does the program know that the line "PROC MEANS N MEAN STD STDERR MAXDEC=2;" is not a line of data? Good, We knew you would figure it out. It's the semicolon at the end of the line. While we're on the subject, there are a few special (and rare) cases where we might run into trouble with this instream form of data entry. Suppose we wrote the above program example like this:

```
DATA EX1;
INPUT GROUP $ X Y Z;
CARDS;
CONTROL 12 17 19
TREAT 23 25 29
CONTROL 19 18 16
TREAT 22 22 29
PROC MEANS N MEAN STD
   STDERR MAXDEC=2;
   TITLE 'MEANS FOR EACH GROUP';
   CLASS GROUP;
   VAR X Y Z;
RUN;
```

It might be necessary to use more than one line to write the first SAS statement following the data (suppose we had a long list of options for PROC MEANS for example). What will happen when we run this program? Well, we just figured out that the program identified the first programming statement following the data by scanning ahead and seeing the semicolon at the end of the line. In the form above, the line "PROC MEANS N MEAN STD" will be read as a data line. The program still thinks that the value of GROUP is "PROC", and will print error messages telling us that the values for X Y and Z are not numeric. Worse yet, the next line "STDERR MAXDEC=2;" will be treated as the first SAS statement following the data (it ends in a semicolon) and, since this line is not a proper SAS statement, an error message will result. This is a fairly rare problem, but if it happens to you, you'll know the cause and here is the solution. Place a null statement after the last line of data when you do not have a semicolon at the end of the first SAS statement following your data. The corrected program looks like this:

```
DATA EX1;
INPUT GROUP X Y Z;
CARDS;
CONTROL 12 17 19
TREAT 23 25 29
CONTROL 19 18 16
TREAT 22 22 29
;
PROC MEANS N MEAN STD
   STDERR MAXDEC=2;
   TITLE 'MEANS FOR EACH GROUP';
   CLASS GROUP;
   VAR X Y Z;
RUN;
```

Before we leave this topic, here is one more (and rare) possibility you may encounter. What happens when your data contains semicolons? For example, suppose you had:

```
INPUT AUTHOR $8. TITLE $40.;
CARDS;
SMITH  The Use of the ; in Writing
FIELD  Commentary on Smith's Book
PROC SORT;
   BY AUTHOR;
```

The program, seeing the semicolon in the first line of data, will treat the line as a SAS statement and generate more error messages than you would like to see. The solution to this rare problem is to use the special SAS statement CARDS4 which requires four semicolons in a row ";;;;" to indicate the end of your data. The corrected example would look like this:

```
INPUT AUTHOR $8. TITLE $40.;
CARDS4;
SMITH   The Use of the ; in Writing
FIELD   Commentary on Smith's Book
;;;;
PROC SORT;
   BY AUTHOR;
```

EXAMPLE 2. Reading ASCII Data from External Files

It is a common occurrence to be given data in an external file to be analyzed with SAS software. Whether on a floppy diskette, on a microcomputer or on a tape used with a mainframe computer, we will want a way to have our SAS program read data from an external source. For this example, we will assume that the data file is either an ASCII (American Standard Code for Information Interchange) file or a "card image" file on tape (also called "raw" data). To read this file is surprisingly easy. The only changes to be made to first example are these: 1. Precede the INPUT statement with an INFILE statement. and 2. Omit the CARDS statement, and, of course, the lines of data. An INFILE statement is the way we tell a SAS program where to find external "card image" data. On a mainframe version of SAS software, an INFILE statement will refer to what is called a DDname (DD stands for Data Definition) which gives information on where to find the file. On OS batch systems, the DDname is included in the JCL (Job Control Language). On systems such as CMS, the DDname is defined with a FILEDEF statement. On a microcomputer, the INFILE statement can either name a file directly or it can be a filename defined with a FILENAME statement. We will show examples of all of these variations.

PC-SAS Example - Reading ASCII data from an External Data File

```
DATA EX2A;
INFILE 'B:MYDATA';
*THIS INFILE STATEMENT TELLS THE
 PROGRAM THAT OUR INPUT DATA IS
 LOCATED IN THE FILE MYDATA ON A
 FLOPPY DISKETTE IN THE B DRIVE;
INPUT GROUP $ X Y Z;
PROC MEANS N MEAN STD STDERR MAXDEC=2;
   VAR X Y Z;
RUN;
```

File MYDATA (located on the floppy diskette in drive B) looks like this:

```
CONTROL 12 17 19
TREAT 23 25 29
CONTROL 19 18 16
TREAT 22 22 29
```

An alternative way of writing the INFILE statement in the example above is to use a FILENAME statement to create an alias or fileref for the file (a short "nickname" which we associate with the file). This is shown below:

```
DATA EX2B;
FILENAME GEORGE 'B:MYDATA';
INFILE GEORGE;
*THIS INFILE STATEMENT TELLS THE
 PROGRAM THAT OUR INPUT DATA IS
 LOCATED IN THE FILE MYDATA ON A
 FLOPPY DISKETTE IN THE B DRIVE;
INPUT GROUP $ X Y Z;
PROC MEANS N MEAN STD STDERR MAXDEC=2;
   VAR X Y Z;
RUN;
```

Note the difference between these two INFILE statements. The first INFILE statement refers to the external file directly and the filename is placed in single quotes. The second INFILE example defines an alias first with a FILENAME statement and then uses the alias with the INFILE statement. Notice that when we use a fileref it is not in single quotes. This point is important since it is the only way that the program can distinguish between an actual file name

and a fileref.

The mainframe example which is shown next is basically the same as the microcomputer example shown above. The only difference is in the way we create the fileref. On an OS batch system, we would create the fileref with a DD statement in the JCL like this:

```
//JOBNAME JOB (ACCT,BIN),'RON CODY'
//    EXEC SAS
//SAS.GEORGE DD DSN=ABC.MYDATA,DISP=SHR
//SAS.SYSIN DD *
DATA EX2C;
INFILE GEORGE;
*THIS INFILE STATEMENT TELLS THE PROGRAM
 THAT THE FILE ABC.MYDATA CONTAINS OUR
 EXTERNAL DATA FILE
 (ASSUME IT IS CATALOGUED);
INPUT GROUP $ X Y Z;
PROC MEANS N MEAN STD STDERR MAXDEC=2;
   VAR X Y Z;
RUN;
```

This example on a CMS system would be the same except that a FILEDEF statement would be used to associate the DDname with the file instead of the DD statement in the JCL. Here it is:

```
CMS FILEDEF GEORGE DISK MYDATA DATA B;
*THE FILE MYDATA DATA IS ON THE B
 MINIDISK OF MY CMS SYSTEM;
DATA EX2D;
INFILE GEORGE;
*THIS INFILE STATEMENT TELLS THE PROGRAM
 THAT THE DATA IS LOCATED IN THE FILE
 WITH FILENAME MYDATA, FILETYPE DATA,
 AND FILEMODE B.;
INPUT GROUP $ X Y Z;
PROC MEANS N MEAN STD STDERR MAXDEC=2;
   VAR X Y Z;
RUN;
```

So, once we know how to create a DDname or a fileref in our computing environment, the SAS statements to read the file are the same. You will need to refer to your manual on how to create a fileref with TSO or VSE. Again, the SAS statements will not change.

There are a variety of options that can be used with an INFILE statement to control how data are read and to allow the SAS program more control over the input operation. These options are placed after the word INFILE and before the semicolon. We will demonstrate several useful options.

Useful options with INFILE;

END=variable name

This option will automatically set the value of "variable name" to 0 unless the current observation is the last record in the file. This can be used when you want to read several different files and combine their data. (An alternative is to use the EOF=label option which branches to "label" when the end of file is reached.)

```
DATA EX2E;
*WE WILL FIRST READ DATA FROM OSCAR AND
 THEN FROM BIGBIRD.TXT.  OSCAR IS AN
 ASCII FILE ON THE FLOPPY DISKETTE IN
 THE B DRIVE AND  BIGBIRD.TXT IS IN A
 SUBDIRECTORY NAMED DATA ON OUR HARD
 DISK (C DRIVE).;
FILENAME X 'B:OSCAR';
FILENAME Y 'C:\DATA\BIGBIRD.TXT';
IF TESTEND NE 1 THEN
   INFILE X END=TESTEND;
ELSE INFILE Y;
INPUT GROUP $ X Y Z;
PROC MEANS N MEAN STD STDERR MAXDEC=2;
   VAR X Y Z;
RUN;
```

Notice here that we can conditionally execute an INFILE statement, thus giving us complete control over the file reading operation. This same example would be valid on OS or CMS systems with the only change being the way that the DDnames or filerefs were assigned.

Option MISSOVER

The MISSOVER option is very useful when you have records of different length and have missing values at the end of a record. This is frequently the case when a text file was created with a word processor and the records were not padded on the right with blanks. Suppose our file called MYDATA has a short record and looks like the one below:

```
CONTROL 1 2 3
TREAT 4 5
CONTROL 6 7 8
TREAT 8 9 10
```

The program EX2A or EX2B would have a problem reading the second record of this file. Instead of assigning a missing value to the variable Z, it would go to the next record and read "CONTROL" as the value for Z and print an error message (since CONTROL is not a numeric value). The SAS LOG would also contain a NOTE telling us that SAS went to a new line when the INPUT statement reached past the end of a line. The remainder of the third record would not be read and the next observation in our data set would be GROUP=TREAT, X=8, Y=9, and Z=10. To avoid this problem, use the MISSOVER option on the INFILE statement. This will set all variables to missing if any record is short. The entire program would look like this:

```
DATA EX2F;
FILENAME GEORGE 'B:MYDATA';
INFILE GEORGE MISSOVER;
INPUT GROUP $ X Y Z;
PROC MEANS N MEAN STD STDERR MAXDEC=2;
   VAR X Y Z;
RUN;
```

Option - LRECL=reclength

You may need to specify your logical record length if it exceeds the default value for your system. When in doubt, add the LRECL (this stands for logical record length and is pronounced El-Recle) option to the INFILE statement. It will not cause a problem if you specify an LRECL larger than your actual record length. For example, suppose you have an ASCII file on a floppy diskette with 210 characters per line and your system default LRECL is 132. To read this file, you would write the INFILE statement like this:

```
INFILE filename LRECL=210;
```

There are many other INFILE options that allow you more control over how data is read from external files. They can be found in the SAS Procedures Guide, or the SAS User's Guide: Basics.

There will be times where you have data within the SAS program itself (following a CARDS; statement) and not in an external file yet you want to use one or more of the INFILE options to control the input data. You can still use these options by specifying a special fileref or DDname called CARDS, followed by any options you wish. Suppose you want to use MISSOVER and you have included the data within the program. You would proceed as follows:

```
DATA EX2G;
INFILE CARDS MISSOVER;
INPUT X Y Z;
CARDS;
1 2 3
4 5
6 7 8
PROC MEANS;
```

EXAMPLE 3. Writing ASCII or "Raw Data" to an External File

We may have reason to have our SAS program write data to an external file in "card image" or ASCII format. Writing raw data to a file would have the advantage of being somewhat "universal" in that most software packages would be able to read it. On most microcomputer systems, as ASCII file could be read by a word processing program, a spread sheet program, or a data base management package. Writing raw data to a file is very much like reading data from an external file. We use one statement, FILE, to tell the program where to send the data, and PUT to indicate which variables and in what format to write them. Thus, to read raw data files we use the statements: INFILE and INPUT; to write raw data files we use the statements: FILE and PUT. Here is a simple example of a program that reads a raw file (MYDATA), creates new variables, and writes out the new file (NEWDATA) to a floppy disk.

```
DATA EX3A;
*THIS PROGRAM READS A RAW DATA FILE,
 CREATES A NEW VARIABLE AND WRITES THE
 NEW DATA SET TO ANOTHER FILE;
FILENAME IN 'C:MYDATA';
FILENAME OUT 'C:NEWDATA';
INFILE IN;
FILE OUT;
INPUT GROUP $ X Y Z;
TOTAL = SUM (OF X Y Z);
PUT GROUP $ 1-10 @12 (X Y Z TOTAL)(5.);
RUN;
```

An alternative form using PC-SAS would be to omit the FILENAME statements and indicate the file names in quotes directly in the INFILE and FILE statements. Running this program will produce a new file called NEWDATA which looks like this:

```
CONTROL     12   17   19   48
TREAT       23   25   29   77
CONTROL     19   18   16   53
TREAT       22   22   29   73
```

Notice that we can employ any of the methods of specifying columns or formats that are permissible with an INPUT statement, with the PUT statement. In the example above, we specified columns for the first variable (GROUP) and a format (in a format list) for the remaining four variables. This gives us complete control over the structure of the file to be created. It goes without saying that this example will work just the same on a mainframe under OS or CMS, providing that the correct JCL or FILEDEF statements are issued. Note that on an OS system, if we are creating a new file, we will have to provide all the parameters (such as RECFM, DISP, UNIT, DSN, DCB, etc.) necessary for your system.

EXAMPLE 4. Creating a Permanent SAS Data Set

So far, we have seen how to read raw "card image" data and to write the same type of data to an external file. We will now demonstrate how to create a permanent SAS data set. A SAS data set, unlike a raw data file, is not usable by

software other than SAS. It contains not only the data, but your variable names, labels, and format assignments (if any). Before we show you an example, let's first discuss the pros and cons of using permanent SAS data sets. First, some cons: As we mentioned, non-SAS programs cannot read SAS data sets. If we are using PC-SAS software, we cannot use a word processor or editor to look at or change anything in a SAS data set. Likewise, on a mainframe, we cannot use any of the editors or utilities to list the contents of the SAS data set. To read, update, or modify a SAS data set requires using SAS software and either writing a program (for example to change a data value) or using SAS/FSP (Full Screen Product) to display and/or update an observation. When you write a SAS program or use SAS/FSP to modify a SAS data set, you must keep in mind that the original raw data are not modified and you can no longer recreate the SAS data set from the raw data without making the modification again. Finally, in the minus column, SAS data sets typically take up more storage than the original data set and are usually kept in addition to the original raw data, thus more than doubling the system storage requirements.

With all these negatives, why create permanent SAS data sets? Probably the most compelling reason to create and use permanent SAS data sets is speed. We think it is safe to say that typical SAS programs use most of the machine resources in the data step. If you plan to be running many different analyses on a data set that will not be changing often, it is a good idea to make the data set permanent for the duration of the analyses. SAS data sets are also a good way to transfer data to other users providing they have SAS software available. Knowing the data structure is no longer necessary since all the variables, labels, and formats have already been defined. We will see shortly how to use PROC CONTENTS to see what is contained in a SAS data set.

Our first example in this section will be to write a SAS program which has the data in the program itself, and creates a permanent SAS data set.

```
LIBNAME FELIX 'C:\SASDATA';
DATA FELIX.EX4A;
*THIS PROGRAM READS DATA FOLLOWING THE
 CARDS STATEMENT AND CREATES A PERMANENT
 SAS DATA SET IN A SUBDIRECTORY CALLED
 \SASDATA ON THE C DRIVE;
INPUT GROUP $ X Y Z;
CARDS;
CONTROL 12 17 19
TREAT 23 25 29
CONTROL 19 18 16
TREAT 22 22 29
RUN;
```

The way we distinguish between temporary and permanent SAS data sets is by the SAS data set name. If we have a two-level name (two names separated by a period), we are defining a permanent SAS data set name. With a single level SAS data set name, we are defining a temporary data set which will disappear when we exit the SAS environment.

In PC-SAS the first part of the two-level name (the part before the period) names a subdirectory, defined with a LIBNAME statement, where the SAS data set is to be stored (or read). We can have many SAS data sets contained within a single subdirectory. On a mainframe implementation, the first level name refers to a fileref (defined with the appropriate control statement) or a DDname (defined in the JCL). On mainframe systems, a single OS data set can contain several SAS data sets.

When this program executes, the data set EX4A will be a permanent SAS data set located in the \SASDATA subdirectory of our C disk. On a microcomputer, if we look at a list of files in the \SASDATA subdirectory, there will be a file called EX4A.SSD. The extension SSD is added to all PC-SAS data sets. Note that on any SAS system, the first level name does not remain with the data set; it is only used to point to a SAS library. The only requirement is that the first level name match either the LIBNAME, the fileref, or the DDname within a program.

Now that we have created a permanent SAS data set, let's see how to read it and determine its contents.

EXAMPLE 5. Reading Permanent SAS Data Sets

Once we have created a permanent SAS data set, we can use it directly in a procedure once we have defined a LIBNAME, fileref, or DDname. Following our PC-SAS example where we created a SAS data set called EX4A and placed it in the C:\SASDATA subdirectory, we will now show you a SAS program which uses this permanent data set.

```
LIBNAME ABC 'C:\SASDATA';
PROC MEANS DATA=ABC.EX4A N MEAN STD
   STDERR MAXDEC=3;
RUN;
```

You can see right away, how useful it is to save SAS data sets. Notice that there is no data step at all in the above program. All that is needed is to define a SAS library (where the SAS data

set is located) and to use a DATA= option with PROC MEANS to indicate on which data set to operate. First, observe that the fileref ABC is not the same name we used when we created the data set. The fileref ABC is defined with the LIBNAME statement and indicates that we are using the subdirectory C:\SASDATA. Therefore, the first part of the two-level SAS data set name is ABC. The second part of the two-level name tells the system which of the SAS data sets located in C:\SASDATA is to be used. It is important to remember that we **must** use the DATA= option with any procedure where we are accessing previously stored SAS data sets because the program will not know which data set to use. When we create a SAS data set in a DATA step, the system keeps track of the "most recently created data set" and uses that data set with any PROCEDURE where you do not explicitly indicate which data set to use with a DATA= option. Just so that we don't short change the mainframe users, the same program, written on an OS system, would look something like this:

```
//GROUCH JOB (1234567,BIN),'OSCAR THE'
//    EXEC SAS
//SAS.ABC DD DSN=OLS.A123.S456.CODY,
//  DISP=SHR
//SAS.SYSIN DD *
PROC MEANS DATA=ABC.EX4A N MEAN STD
   STDERR MAXDEC=3;
/*
//
```

Imagine, a one-statement SAS program! The DDname was defined in the JCL, indicating the SAS data set was stored in the OS data set called OLS.A123.S456.CODY which was catalogued. On a CMS system, the DDname or first part of a two-level name is what CMS calls the filetype in the general filename filetype filemode method of defining a file. The filename corresponds to the SAS second level name. Thus, without even issuing a FILEDEF command, we could write:

```
PROC MEANS DATA=ABC.EX4A N MEAN STD
   STDERR MAXDEC=3;
```

as long as we had a filetype of ABC and a filename of EX4A.

How to Determine the Contents of a SAS Data Set (PROC CONTENTS)

As we mentioned earlier, we cannot use our system editor to list the contents of a SAS data set. How can we "see" what is contained in a SAS data set? We use PROC CONTENTS. This very useful procedure will tell us important information about our data set: The number of observations, the number of variables, the record length, and an alphabetical listing of variables (which includes labels, length, and formats). As an option, you can obtain a list of variables in order of their position in the data set. As an example, here are the statements to display the contents of the permanent SAS data set EX4A created above:

```
LIBNAME SUGI 'C:\SASDATA';
PROC CONTENTS DATA=SUGI.EX4A POSITION;
```

Output from this procedure is shown in figure 1:

One final point of information, the DATA= option of PROC CONTENTS can be used to list all the SAS data sets contained in a SAS library instead of a single data set. Instead of using the form libname.datasetname use the form libname._ALL_. This will display all the SAS data sets stored in the library referred to by the libname. Of course, if you to have SAS/FSP (full screen product), you can browse and edit a SAS data set with FSBROWSE and FSEDIT.

A Special Note on Permanent SAS data Sets with Formats

One special note is needed to caution you about saving permanent SAS data sets in which you have assigned formats to one or more of the variables in the DATA step. If you try to use this data set (in a procedure for example), you will get an error that formats are missing. The important thing to remember is this: if you create a permanent SAS data set which assigns formats to variables, you must make the format library permanent as well. Also, if you give someone else the data set, make sure you give him or her the format library. To make your format library permanent, add the LIBRARY= option to PROC FORMAT. Then, issue a LIBNAME statement with the special library name LIBRARY pointing to the format library when accessing the data set. Since this sounds rather complicated, we will show the code to create a permanent format library and the code to access a permanent data set where formats were used.

Code to Create a Permanent Format Library and Assign the Format to a Variable

```
LIBNAME LIBRARY 'C:\SASDATA';
*THE SAS FORMAT LIBRARY IS LOCATED IN
 C:\SASDATA;
PROC FORMAT LIBRARY=LIBRARY;
```

```
   VALUE XGROUP 'TREAT'='TREATMENT GRP'
    'CONTROL'='CONTROL GRP';
LIBNAME FELIX 'C:\SASDATA';
DATA FELIX.EX4A;
INPUT GROUP $ X Y Z;
FORMAT GROUP XGROUP.;
CARDS;
CONTROL 12 17 19
TREAT 23 25 29
CONTROL 19 18 16
TREAT 22 22 29
RUN;
```

Program to Read a Permanent SAS Data Set with Formats

```
LIBNAME C 'C:\SASDATA';
LIBNAME LIBRARY 'C:\SASDATA';
PROC PRINT DATA=C.EX4A;
```

Rename Warning

Before we leave the topic of permanent SAS data sets, one final note concerning SAS data sets created with PC-SAS. Do not use the DOS REN (rename) command to change the name of a SAS data set (.SSD file). You will not be able to read it if you do. SAS data sets contain the data set name internally and the DOS file name and the internal name must match. If you wish to change the name of a SAS data set, use PROC DATASETS with a change statement to do it. For example, to rename the permanent data set EX4A to OSCAR, the appropriate statements would be:

```
LIBNAME XXX 'C:\SASDATA';
*THE LIBRARY WHERE THE DATA
SET IS LOCATED;
PROC DATASETS LIBRARY=XXX;
   CHANGE EX4A=OSCAR;
*THE SYNTAX IS CHANGE old name
 = new name;
```

EXAMPLE 6. Reading Data from a Lotus Spread Sheet or a dBASE File

You may be given data in the form of a Lotus spread sheet or some other compatible format. Of course you could print a report to a disk file, making sure to omit page formatting (no margins top, bottom, or left and no page breaks). You could then use an INFILE and INPUT statement to read the file and do analyses. A more direct way would be to convert the spread sheet file to either a DIF (data interchange format) file or a dBASE III compatible file. A translate program is a standard part of the Lotus package. This program can convert WKS and WK1 files to a variety of formats including DIF and DBF. We will demonstrate the transfer of data from a spread sheet to a SAS data set using both the DIF and DBF formats.

We will first demonstrate translating a spread sheet to a DBF file and then to a SAS data set. This method will also convert dBASE files to SAS data sets. The first thing to remember is that the first row of the spread sheet should contain your variable names. It is best to keep these column headings to valid SAS variable names. However, if they are not (i.e. too long) the translate facility will truncate or modify the names. Later, when we translate to a SAS data set, invalid characters will be replaced as well. We must also be sure that the first row of data either contains a value (so the system can determine a format by context) or is formatted. Furthermore, missing numeric values in the spread sheet will be turned into zeros since dBASE III uses zero as a missing value. If this will cause trouble, you may want to choose a missing value such as 999 to place in the empty cells. Assuming you have taken care of these details, here are the steps to transfer data from a spread sheet to a SAS data set:

A. Make sure that the first row of the spread sheet contains variable names (preferably valid sas variable names, left justified, not centered).

B. Either format the first row of cells or make sure that there is a value in the first row for each variable.

C. Enter the Lotus (or equivalent) translate program. You will want to translate from WKS or WK1 (depending on which version you are running) to dBASE III. If your original spread sheet is called MYDATA.WK1, the translate routine will create a file called MYDATA.DBF. (Note, instructions from here on will work with original dBASE III files.)

D. Run the following SAS program. For this example, the data set MYDATA.DBF will be assumed to be on a floppy in the A drive. (Modify the program accordingly if the data set is elsewhere.) The SAS data set will be placed in the \SASDATA subdirectory of the C disk.

```
LIBNAME C 'C:\SASDATA';
FILENAME DBIN 'A:MYDATA.DBF';
PROC DBF DB3=DBIN OUT=C.MYDATA;
RUN;
```

The result of running this program will be a SAS data set located in C:\SASDATA called MYDATA.SSD. The

observations in this data set will correspond to the rows of the spread sheet and the variables will be the column headings in the first row of the spread sheet. Note that we did not have to name our SAS data set MYDATA but it is a good idea to keep the same name from the .WK1 to the .DBF to the .SSD file. If you want to recode the missing values to SAS missing values, you can use the following methodology: (Assume the variables are GROUP, X, Y, and Z and that we left the worksheet blank for missing values.)

```
LIBNAME C 'C:\SASDATA';
DATA C.NEWDATA;
SET C.MYDATA;
ARRAY DUMMY(*) X Y Z;
DO I=1 TO DIM(DUMMY);
   IF DUMMY(I)=0 THEN DUMMY(I)=.;
   END;
RUN;
```

This program will change all values of zero for the variables X, Y, and Z to the SAS missing value. The same method would apply to change 999 to missing.

EXAMPLE 7. Converting a SAS data set to a dBASE File or Spread Sheet

The same procedure (PROC DBF) can convert SAS data sets to DBF files. The options to use are:

DB3=name of the DBF file you want to create (or use DB2 if you want a dBASE II compatible file)

DATA=SAS data set name

As an example, suppose we have a SAS data set (HENRY.SSD) in the \SASDATA subdirectory and want to create a Lotus compatible file called HENRY.WK1. Here are the steps:

A. Run the following SAS program:

```
LIBNAME C 'C:\SASDATA';
FILENAME DBOUT 'A:HENRY.DBF';
PROC DBF DB3=DBOUT DATA=C.HENRY;
RUN;
```

This will create a dBASE III file on a floppy diskette in the A drive. If you want to continue on to a spread sheet format continue with step B. below:

B. Enter the Lotus translate program and follow the instructions to translate from DBF format to WK1 or WKS.

EXAMPLE 8. Reading Data from a DIF File.

Many programs (spread sheets, database programs, etc.) have the ability to create DIF (data interchange format) files. The SAS procedure DIF can convert between DIF format and SAS data sets. If you are moving data from a spread sheet to a SAS data set, the preferable method is via the DBF route since the variable names come along with the data." However, if you have a DIF file from some other application or have a spread sheet without column headings (and don't want to add them), here is a sample program to convert from DIF format to a SAS data set:

```
*PROGRAM TO CONVERT FROM DIF FORMAT TO
 A SAS DATA SET;
FILENAME XYZ 'A:MYDATA.DIF'
*DIFF FILE ON A FLOPPY IN THE A DRIVE;
LIBNAME QWERTY 'C:\SASDATA';
*THE PERMANENT SAS DATA SET WILL BE IN
 C:\SASDATA;
PROC DIF DIF=XYZ OUT=QWERTY.MYSASDTA;
RUN;
```

If the spread sheet has one or more rows before the data begins (e.g. column headings), they can either be removed before converting to DIF format or you can use the SKIP option of PROC DIF to skip n rows. The form is:

```
PROC DIF DIF=XYZ OUT=MYSASDTA SKIP=n;
```

where n is the number of rows to skip. The result of running this program will be a SAS data set with variable names COL1, COL2, etc. If you don't care for these variable names you have two choices. One, use the PREFIX= option of PROC DIF to choose a prefix other than COL (e.g. VAR). Still better, is to rename the variables to meaningful variable names with a RENAME statement with PROC DATASETS. For example:

```
LIBNAME C 'C:\SASDATA';
PROC DATASETS LIBRARY=C;
 MODIFY MYSASDTA;
 RENAME COL1=GROUP COL2=X COL3=Y COL3=Z;
RUN;
```

EXAMPLE 9. Writing a DIF file from a SAS data set

It is very simple to go the other way and change a SAS data set into a DIF file. Again, you use PROC DIF but instead of the OUT= option, use the DATA= option to name your SAS data set. The fileref referenced by the DIF= option, will point to the destination of the DIF file. If the final file is to be a spread sheet, you can use the spread sheet translation program to convert from the DIF file. However, as mentioned before, importing and exporting from a spread sheet is best done via the DBF

format.

I recently reviewed a package called DBMS/COPY from Conceptual Software Inc. (P.O Box 56627, Houston, TX 77256-6627). With this package, you can translate between several dozen formats, including SAS system files (.SSD), SPSS, dBase III, LOTUS (WK1), and many other popular packages. With one COPY command the translation is done. Suppose you had a LOTUS spread sheet called SHEET.WK1 and wanted to run SAS procedures on the data. The command "DBMSCOPY SHEET.WK1 A.SSD" would accomplish the translation and the missing values (blanks in the spread sheet) would be correctly converted to SAS missing values. The only glitch is, as of this writing, the SAS system file can only be a single letter (e.g. A.SSD) because of the encryption algorithm that SAS software uses to combine the SAS data set name and the date. However, this package makes translating from dBase, LOTUS, or other packages very easy. (You can use PROC DATASETS to rename the SAS system file.)

FIGURE 1.

```
SAS                                    9:30 Monday, August 8, 1988    1

CONTENTS PROCEDURE

  Data Set Name:  SUGI.EX4A                Type:
  Observations:   4                        Record Len: 36
  Variables:      4
  Label:

-----Alphabetic List of Variables and Attributes-----

  #  Variable  Type  Len  Pos  Label
  1  GROUP     Char    8    4
  2  X         Num     8   12
  3  Y         Num     8   20
  4  Z         Num     8   28

CONTENTS                          PROCEDURE
----Variables Ordered by Position----

  #  Variable  Type  Len  Pos  Label
  1  GROUP     Char    8    4
  2  X         Num     8   12
  3  Y         Num     8   20
  4  Z         Num     8   28
```