

How to Excel with SAS®

Keith Cranford

Office of the Attorney General of Texas, Child Support Division

SAS can both read from and write to MS Excel. This workshop will demonstrate how SAS can use MS Excel as a data source and as a reporting tool. PROC IMPORT, PROC EXPORT, LIBNAME, and ODS will be among the SAS tools used to interact with MS Excel. The workshop is intended as an introduction to accessing MS Excel through source code and is most appropriate for SAS users with limited experience to these techniques, but intermediate users may pick up a trick or two.

Reading from Excel Spreadsheets

Microsoft Excel spreadsheets can be read from SAS in several ways. Two of these will be demonstrated here. First, PROC IMPORT allows direct access to Excel files through SAS/Access to PC File Formats or access to Comma-Separated (CSV) files through Base SAS. The second method uses the Excel LIBNAME engine.

PROC IMPORT

The IMPORT procedure reads from external sources and creates a SAS data set. Two sources are Excel spreadsheets and CSV files. A particular SAS/Access product may be required for certain sources, however. In our example, SAS/Access to PC File Formats is required to read an Excel file, but a CSV file can be accessed with Base SAS.

General Syntax for PROC IMPORT:

PROC IMPORT

```
DATAFILE="filename" | TABLE="tablename"  
OUT=<libref.>SAS-data-set <(SAS-data-set-options)>  
<DBMS=identifier> <REPLACE> ;
```

For Excel you use the DATAFILE="filename" option to specify the Excel file to be read. (The TABLE="tablename" option would be applicable if you were reading from a database such as Microsoft Access.) The OUT= option specifies the SAS data set to be created. The DBMS= option identifies the type of file to be read. In this case, you will use either EXCEL or CSV to read an Excel spreadsheet or CSV file, respectively. Finally, the REPLACE option determines whether to replace the data set that is created, if it already exists.

There are other supporting statements. A few will be mentioned below, but for a complete list please refer to SAS documentation. Fortunately, much can be done without any or very few of these statements.

This first example reads an Excel file, example1.xls, and creates the SAS data set, example1_ages. If the data set already exists, it will be replaced.

```

**-- PROC IMPORT using Excel DBMS --**;
proc import file='example1.xls' out=example1_ages
    dbms=excel replace ;
run ;

```

This is the Excel file being read and the resulting data set.

	A	B	C	D
1	name	age		
2	Tom	35		
3	Dick	29		
4	Harry	64		
5				
6				
7				

	name	age
1	Tom	35
2	Dick	29
3	Harry	64

Notice that PROC IMPORT used the first row to automatically create the variable names. If the first row did not contain column headings, you can use the GETNAMES=NO statement to not look for variable names, and the resulting data set will contain variables such as F1, F2, etc.

By default PROC IMPORT reads the first worksheet in the Excel spreadsheet. The SHEET= statement allows you to specify which worksheet to read. The following example reads the worksheet "SHEET1" from the example1.xls spreadsheet.

```

**-- PROC IMPORT using Excel DBMS and specifying sheet --**;
proc import file='example1.xls' out=example1_sheet1
    dbms=excel replace ;
    sheet="Sheet1" ;
run ;

```

The resulting data set is similar to the first example.

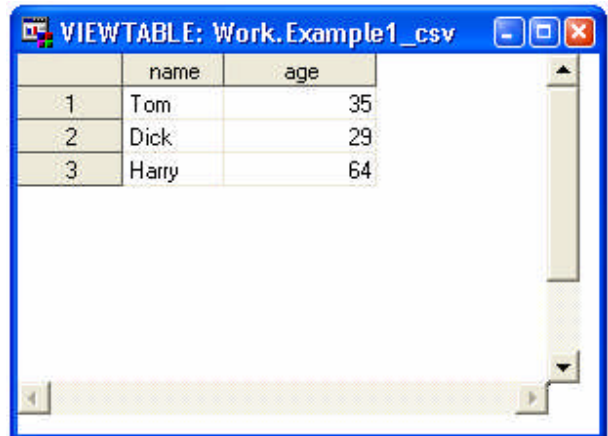
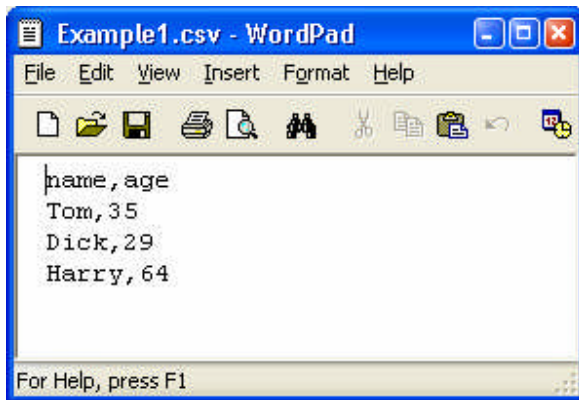
	A	B	C	D
1	col1	col2		
2		1 A		
3		2 B		
4		3 C		
5				
6				
7				
8				

	col1	col2
1		1 A
2		2 B
3		3 C

PROC IMPORT can also be used to read CSV files. Use the DBMS=CSV option to indicate that a CSV file is to be read. As mentioned above, this does not require SAS/Access to PC File Formats. This example reads the example1.csv, which is similar to the first example.

```
**-- PROC IMPORT using CSV DBMS --**;  
proc import file='example1.csv' out=example1_csv  
    dbms=csv replace ;  
run ;
```

The CSV file and resulting data set are shown below.



	name	age
1	Tom	35
2	Dick	29
3	Harry	64

Excel LIBNAME Engine

SAS can also use the LIBNAME statement to access Excel files. This allows you to access the worksheets in an Excel file in much the same way that you do SAS data sets in a SAS library. Each worksheet will appear as a data set in the library and can be accessed through SAS procedures, such as PROC PRINT, using a two-level data set name.

First, you must create the library reference with a LIBNAME statement. The syntax for the LIBNAME statement using the EXCEL engine is as follows.

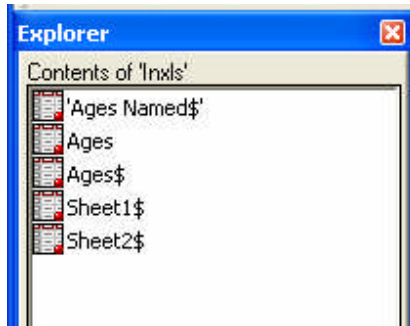
```
LIBNAME libref EXCEL <physical-file-name>;
```

The *libref* is your library reference just as you would use with a SAS data library. The EXCEL engine name specifies the engine to use. The <physical-file-name> must refer to an Excel file.

For example, you can create a library reference to the workbook used above.

```
**-- Using Excel LIBNAME Engine --**;  
libname inxls excel 'example1.xls' ;
```

Now you can reference worksheets within this Excel files with a two-level name. Opening the INXLS library from SAS Explorer you can see the list of available worksheets and named ranges.



The worksheet names include a \$ sign on the end, whereas the named ranges are merely listed. This is also a clue to how they are used in procedures. You must use a name literal to access the worksheets, which means enclosing the name in quotes followed immediately by the letter n, much like a date literal. The named ranges can be referenced in the same way as a SAS data set.

The following example prints the Ages worksheet.

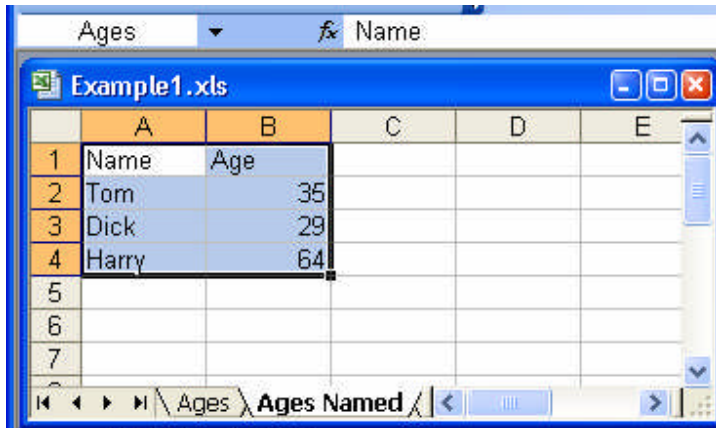
```
**-- Name literal --**;  
proc print data=inxls."Ages$"n ;  
run ;
```

Notice the use of the two-level name with the worksheet referenced as a name literal. This produces the following output.

```
The SAS System          11:02 Monday, July 16, 2007  
  
Obs    name    age  
1      Tom     35  
2      Dick    29  
3      Harry   64
```

This gives you access to the same data as you had importing the worksheet, but without having to first import the data into a SAS data set. PROC PRINT processes the data directly from the worksheet. Of course, this requires that your worksheet has column names and it is structured properly. However, this method is very convenient.

A similar method allows you to access named ranges within a worksheet. The "Ages Named" worksheet contains a named range called Ages.



PROC PRINT can be used to print this data similar to printing any SAS data set.

```

**-- Named Ranges ---**;
proc print data=inxls.Ages ;
run ;

```

The result is similar to the example above.

```

The SAS System      11:02 Monday, July 16, 2007

Obs   Name   Age
-----
1     Tom    35
2     Dick    29
3     Harry   64

```

The advantage of this method is that your code does not contain name literals so looks like any other SAS code. The disadvantage is that you have to take the time to create these named ranges.

Column Headers

By default, the Excel engine assumes that a worksheet contains column headers, which has been the case in the examples so far. What happens if the worksheet does not contain column headers. This is shown in the following example.

```

**-- Headers ---**;
libname inxls2 excel 'example2.xls' ;

```

This Excel workbook contains a worksheet called NoHeader.

	A	B	C
1	1 A		
2	2 B		
3	3 C		
4			
5			
6			

Opening this data in SAS reveals how SAS uses the first row to create variable names, even when this is not intended.

	F1	A
1	2 B	
2	3 C	

Since the first cell value (the number 1) did not contain a valid variable name, SAS automatically assigned the variable F1. The second cell value, A, is a valid variable name so was used. Of course, this is not what was intended, since the first row contains data values. You need to specify with the HEADER option in the LIBNAME statement that there is no header row. This is done as follows.

```
libname inxls2 excel 'example2.xls' header=no;
```

This results in the following data set. As intended, now the first row is part of the data, and SAS assigned the variable names F1 and F2.

	F1	F2
1	1 A	
2	2 B	
3	3 C	

One thing to note is that the HEADER= option applies to all worksheets in the specified workbook. You cannot have some worksheets with headers and some without in the same workbook, and have SAS handle these properly.

Since the spreadsheets are locked while you have the library reference in effect, it is a good practice to clear the reference when you are finished. Use the LIBNAME statement to do this.

```
libname inxls clear ;  
libname inxls2 clear ;
```

As seen in these examples, SAS can read Excel files with either PROC IMPORT or the LIBNAME statement, using very few statements.

Writing to Excel Spreadsheets

SAS can also write to MS Excel Spreadsheets. Three ways will be discussed here. First, PROC EXPORT can be used to write directly to a spreadsheet through SAS/Access to PC File Formats. Secondly, the EXCEL engine on the LIBNAME statement allows you to write to Excel much as you would to any SAS data set. These methods work very similar to what you have seen above in reading Excel spreadsheets. The last method shown takes advantage of Output Delivery System (ODS) features to create Excel files.

PROC EXPORT

PROC EXPORT using the DBMS option can be used to write a SAS data set to an Excel spreadsheet. This is basically the reverse of what you saw using PROC IMPORT to read an Excel spreadsheet. The general syntax for PROC EXPORT is as follows.

```
PROC EXPORT DATA=<libref.>SAS-data-set <(SAS-data-set-options)>  
  OUTFILE="filename" | OUTTABLE="tablename"  
  <DBMS=identifier> <REPLACE>;  
  <data-source-statement(s);>
```

The DATA= option specifies the SAS data set used as the source. For Excel you use the OUTFILE="filename" option to specify the Excel file to be written to. (The OUTTABLE="tablename" option would be applicable if you were writing to a database such as Microsoft Access.) The DBMS= option identifies the type of file to be read. In this case, you will use EXCEL2000 to write to an Excel spreadsheet. Finally, the REPLACE option determines whether to replace the contents of the worksheet that is created, if it already exists.

An important <data-source-statement(s);> is the SHEET statement. This allows you to specify the sheet name within the workbook to which to write. The sheet may already exist, or SAS will create it, if necessary. Again, there are other statements, but please refer to your SAS documentation for these.

To illustrate writing to Excel spreadsheets, consider the simple data set TEST1 with two variables and two observations.

	x	y
1	1	2
2	3	4

You can write this data set to Excel using the following PROC EXPORT.

```

**-- Using PROC EXPORT to write to Excel --**;
proc export data=test1
            outfile="test1.xls"
            dbms=excel2000 replace ;
            sheet="Mytable" ;
run ;

```

This code creates the following worksheet Mytable2 in the Excel spreadsheet test1.xls.

	A	B	C	D
1	x	y		
2	1	2		
3	3	4		
4				
5				
6				
7				

If the spreadsheet does not exist, SAS will create it. If it already exists, SAS will write directly to it. The same is true for the worksheet. If the spreadsheet or worksheet already exist, it is important to include the replace option. Otherwise, the spreadsheet will not be updated. The SHEET statement allows you to also write to multiple sheets within the same spreadsheet.

LIBNAME Statement

Similar to what we saw earlier, a LIBNAME statement can be used to write to as well as read from Excel spreadsheets. One additional option is advisable, though. The VERSION option specifies which version of EXCEL file you want to create. In the following example, VERSION=2000 is used.


```
**-- Using Excel LIBNAME Engine --**;  
libname outxls excel 'example_out2.xls' version=2000 ;
```

The assigns outxls as a library reference to the Excel file example_out2.xls, and it will create an Excel 2000 file.

You can then use this library reference just as you would any SAS data library. You can write the TEST1 data set to this spreadsheet using a DATA step.

```
data outxls.simple ;  
    set test1 ;  
run ;
```

This will create the worksheet simple in the example_out2.xls spreadsheet. It will look similar to the previous example using PROC EXPORT.

	A	B	C	D	E
1	x	y			
2	1	2			
3	3	4			
4					
5					
6					

An advantage of the LIBNAME statement over PROC EXPORT is that you can create new columns as you are writing to the spreadsheet. This works in the same manner as any other data step. The following example creates a new column, labeled z, which is the sum of x and y.

```
data outxls.compute ;  
    set test1 ;  
    z = x + y ;  
run ;
```

This adds the compute worksheet to the same spreadsheet.

	A	B	C	D	E
1	x	y	z		
2	1	2	3		
3	3	4	7		
4					
5					
6					

A drawback with the using the EXCEL engine in the LIBNAME statement is that it does not allow replacement of a worksheet. You must delete the worksheet before attempting to write to a particular worksheet.

Finally, as mentioned previously it is a good practice to clear the library reference when you have finished writing to the spreadsheet.

```
libname outxls clear ;
```

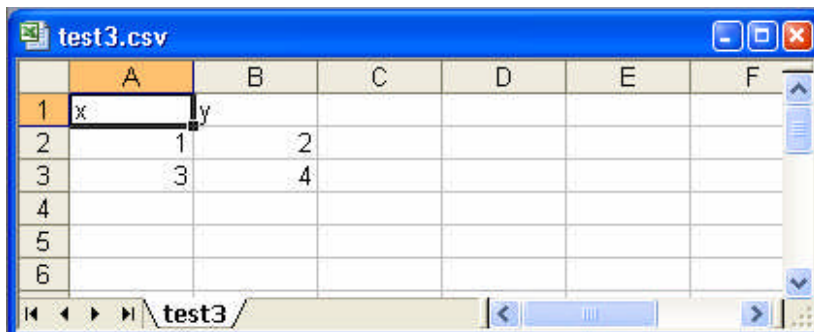
ODS Statement

The ODS statement can also be used to create Excel spreadsheets. This can be done a few different ways. These include the CSV tagsets and the HTML and MSOFFICE2K destination. The general idea using ODS is to write a report out to the ODS destination, which in this case will be an Excel spreadsheet. The only difference in the methods will be the form of the spreadsheet.

The ODS CSV tagset creates a CSV file. The following illustrates this method.

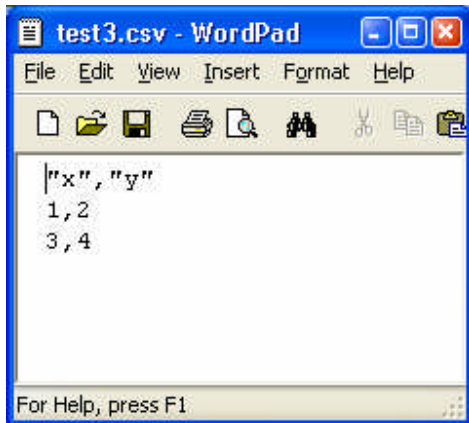
```
**-- Using ODS CSV Tagsets --**;  
ods tagsets.csv file='test2.csv' ;  
    proc print data=test1 noobs ;  
        run ;  
ods tagsets.csv close ;
```

This creates a spreadsheet containing a worksheet with the same name.



	A	B	C	D	E	F
1	x	y				
2	1	2				
3	3	4				
4						
5						
6						

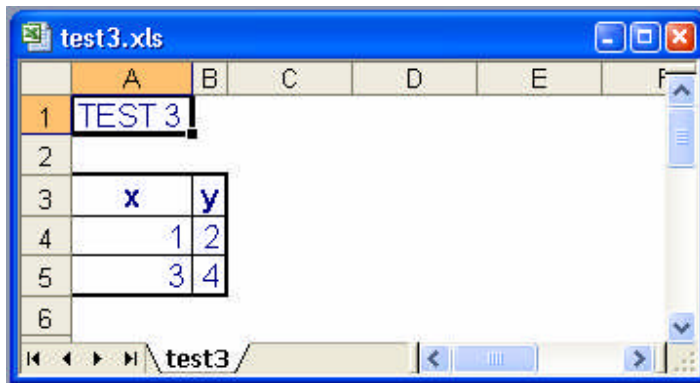
The following view of the file shows that it is a comma-separated file. One advantage of this file is that it will minimize storage space, since no formatting is included in the file.



If you want a file that is closer to Excel, you can use the HTML or MSOFFICE2K destinations. These work in much the same way.

```
**-- Using ODS HTML --**;  
ods html file='test3.xls' ;  
    proc print data=test1 noobs ;  
        title 'TEST 3' ;  
    run ;  
ods html close ;
```

This creates the following file.

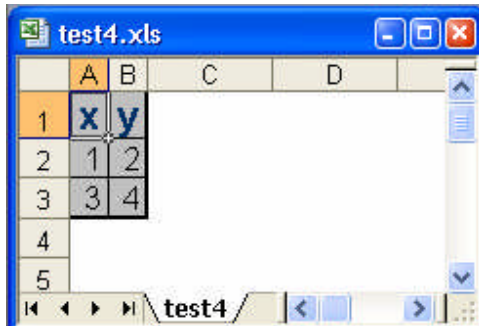


This is really an HTML file with an .xls extension. Also, you can apply an ODS style to format the colors and fonts used in the report.

Alternatively, you can use the MSOFFICE2K destination.

```
**-- Using ODS MSOFFICE2K --**;  
ods msoffice2k file='test4.xls' ;  
    proc print data=test1 noobs ;  
        title ;  
    run ;  
ods msoffice2k close ;
```

This creates the following file.



This also is an HTML file, but with a different set of stylesheets and characteristics.

Conclusion

As illustrated by the examples above, SAS works very nicely with Microsoft Excel in both reading from or writing to Excel spreadsheets. Although this is meant to be an introduction to these techniques, it is hoped that it has provided a basis for handling more complicated tasks.

Contact Information

Keith Cranford
Office of the Attorney General of Texas
Child Support Division
keith.cranford@cs.oag.state.tx.us