Reading and Processing the Contents of a Directory

Ben Cochran, The Bedford Group, Raleigh, NC

ABSTRACT:

On occasions, a SAS user might find themselves in the position where they need to write a SAS® program that can read and process files in a specific directory. In this case, the contents are all excel spreadsheet files. All these files need to be read and converted into SAS datasets. This paper illustrates how to do this in a step by step process using the DATA step.

INTRODUCTION:

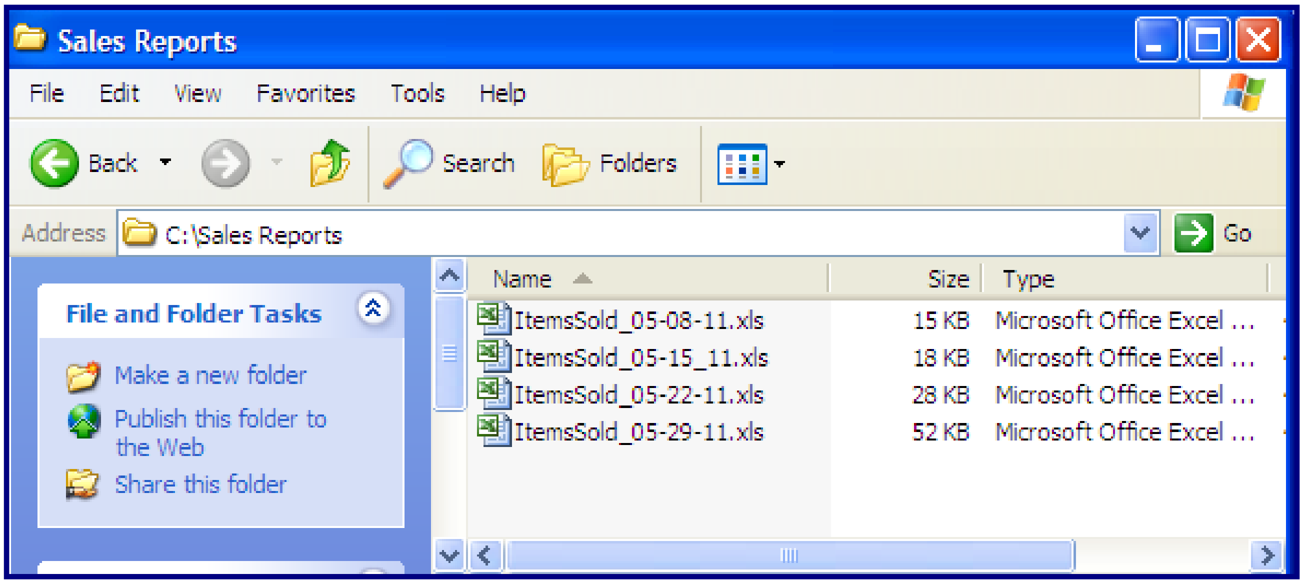
There are four functions that are especially useful when performing this task. They are DOPEN, DNUM, DREAD, and DCLOSE.

- The **DOPEN** function - opens a directory and returns the directory identifier. syntax: **DOPEN**(fileref) example: directory_id = DOPEN(fileref) **;** * you must associate a fileref with the directory before using the DOPEN function.
- The **DNUM** funtion - returns the number of members in a directory
  syntax: **DNUM**(directory_id)
  example: number = DNUM (directory_id) **;**

- The **DREAD** function - returns the name of a directory member
  syntax: name = **DREAD**(directory_id, member-number)
  example: filename=DREAD(directory_id, i) **;**

- The **DCLOSE** function – closes a directory opened by the DOPEN function.
  syntax: DCLOSE (directory_id)
  example: rc=DCLOSE(directory_id);

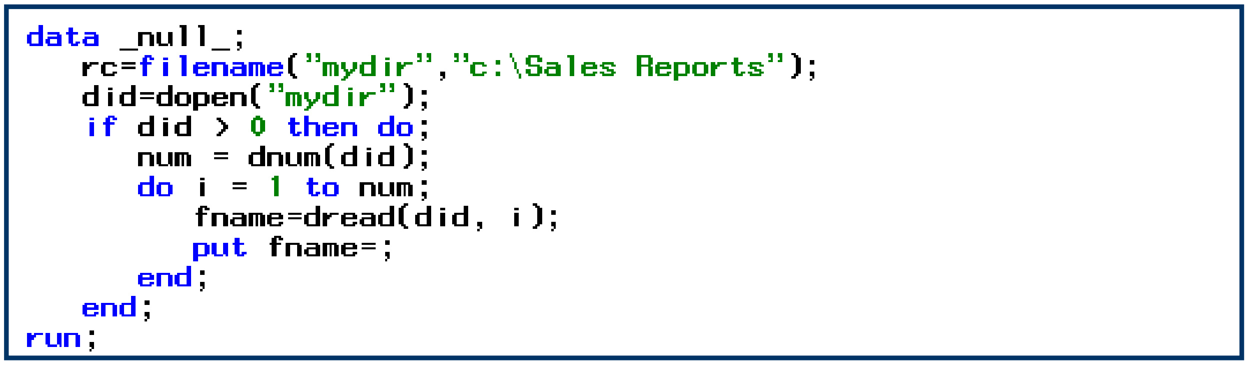
All the spreadsheets are located in the C:\Sales Reports directory. (See Figure 1.)



Name	Size	Type
ItemsSold_05-08-11.xls	15 KB	Microsoft Office Excel ...
ItemsSold_05-15_11.xls	18 KB	Microsoft Office Excel ...
ItemsSold_05-22-11.xls	28 KB	Microsoft Office Excel ...
ItemsSold_05-29-11.xls	52 KB	Microsoft Office Excel ...

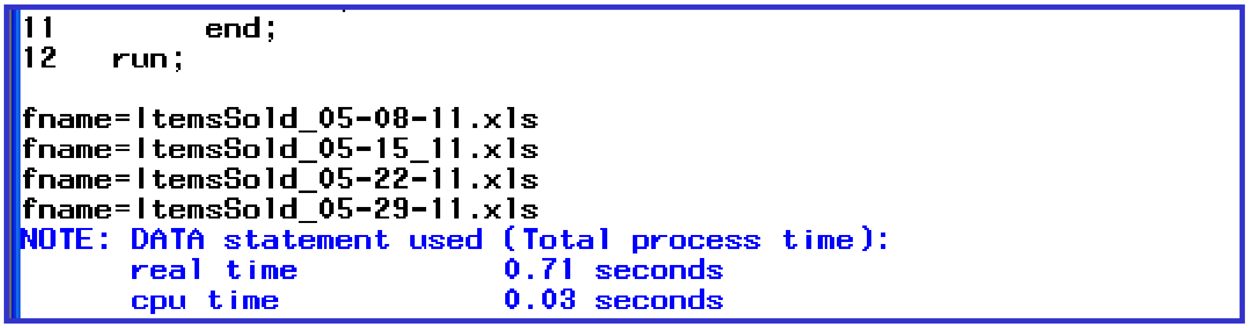
Figure 1. - The directory.

STEP 1: Write a DATA Step that will read this directory and each file within it. Use the **DOPEN, DNUM,** and **DREAD** functions to process a directory. (See figure 2.) This step is just a proof on concept to see if we are actually reading the contents correctly. The results are written to the log.



```
data _null_;
   rc=filename("mydir","c:\Sales Reports");
   did=dopen("mydir");
   if did > 0 then do;
      num = dnum(did);
      do i = 1 to num;
         fname=dread(did, i);
         put fname=;
      end;
   end;
run;
```

Figure 2. The DATA step.

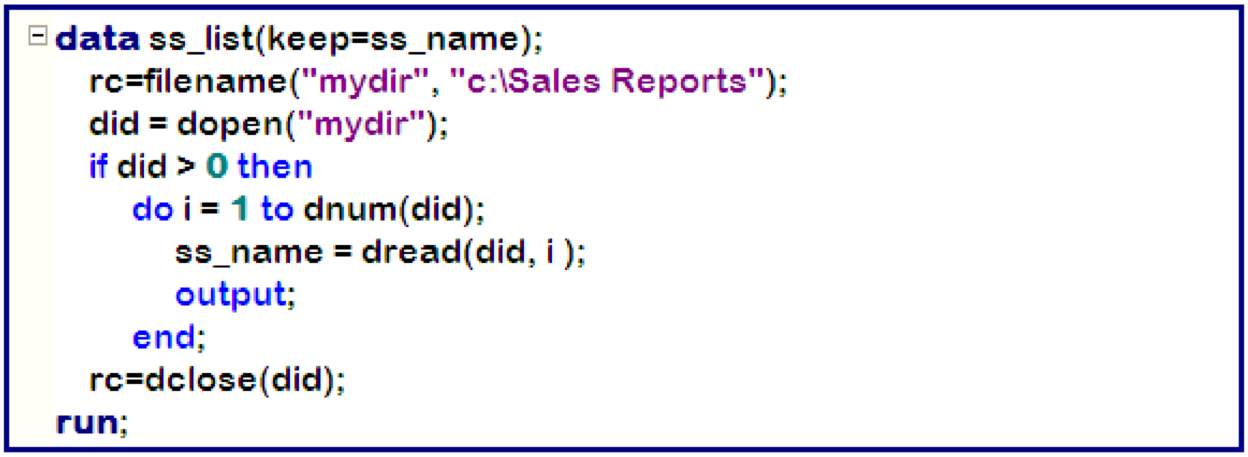


```
11          end;
12     run;

fname=ItemsSold_05-08-11.xls
fname=ItemsSold_05-15_11.xls
fname=ItemsSold_05-22-11.xls
fname=ItemsSold_05-29-11.xls
NOTE: DATA statement used (Total process time):
      real time           0.71 seconds
      cpu time            0.03 seconds
```

Figure 3. The SAS Log.

Everything appears to be working fine, so let's modify the DATA step to create a SAS dataset that contains an observation for each spreadsheet name.

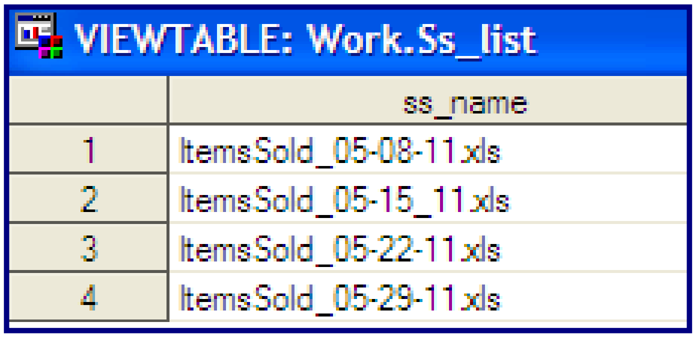
STEP 2: Modify the program to create a **dataset** that contains the spreadsheet names.



```
data ss_list(keep=ss_name);
   rc=filename("mydir", "c:\Sales Reports");
   did = dopen("mydir");
   if did > 0 then
      do i = 1 to dnum(did);
         ss_name = dread(did, i );
         output;
      end;
   rc=dclose(did);
run;
```

Figure 4. The modified DATA step.

The contents of the **SS_List** dataset is shown in figure 5.



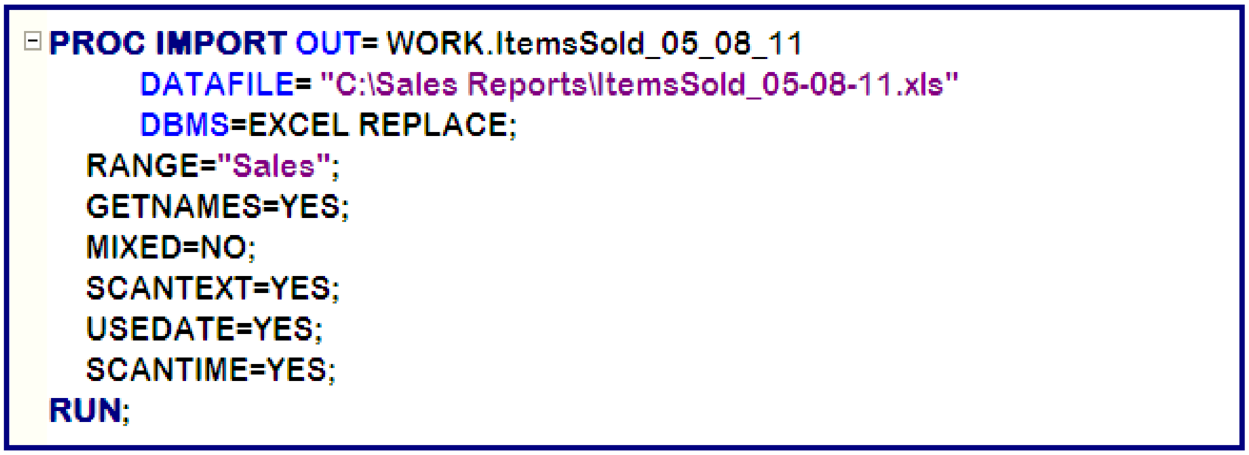
VIEWTABLE: Work.Ss_list

	ss_name
1	ItemsSold_05-08-11.xls
2	ItemsSold_05-15_11.xls
3	ItemsSold_05-22-11.xls
4	ItemsSold_05-29-11.xls

Figure 5. The SS_List data set.

Notice the names of the spreadsheets. They are not spelled consistently. Sometimes this happens when different people create each one.

STEP 3: Write a PROC IMPORT program to read the first spreadsheet and create the first SAS dataset. .

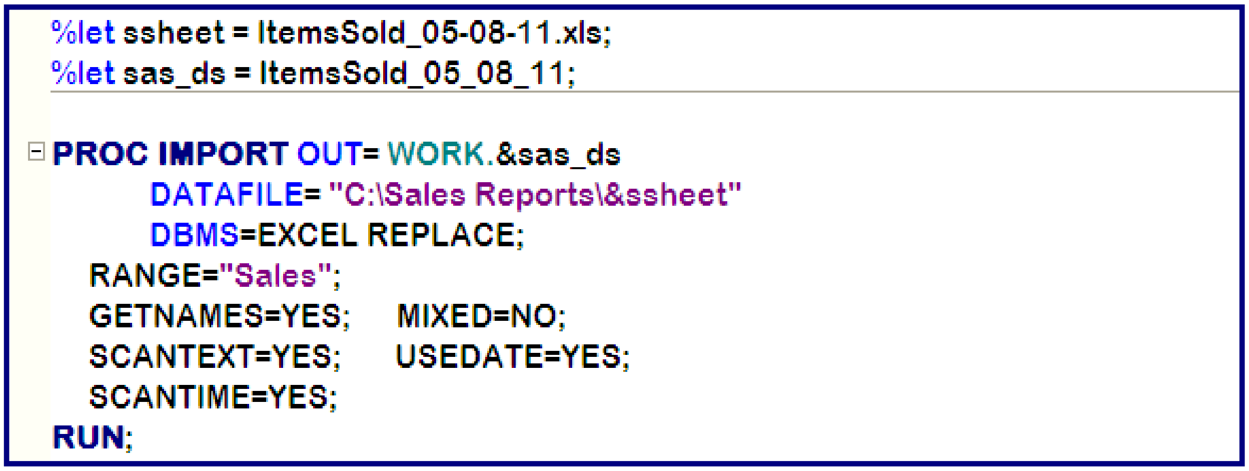


```
PROC IMPORT OUT= WORK.ItemsSold_05_08_11
         DATAFILE= "C:\Sales Reports\ItemsSold_05-08-11.xls"
         DBMS=EXCEL REPLACE;
   RANGE="Sales";
   GETNAMES=YES;
   MIXED=NO;
   SCANTEXT=YES;
   USEDATE=YES;
   SCANTIME=YES;
RUN;
```

Figure 6. The PROC IMPORT step.

Notice the name of the dataset that is created. Notice how it differs slightly from name of the spreadsheet itself. Valid spreadsheet names are not always valid SAS dataset names.

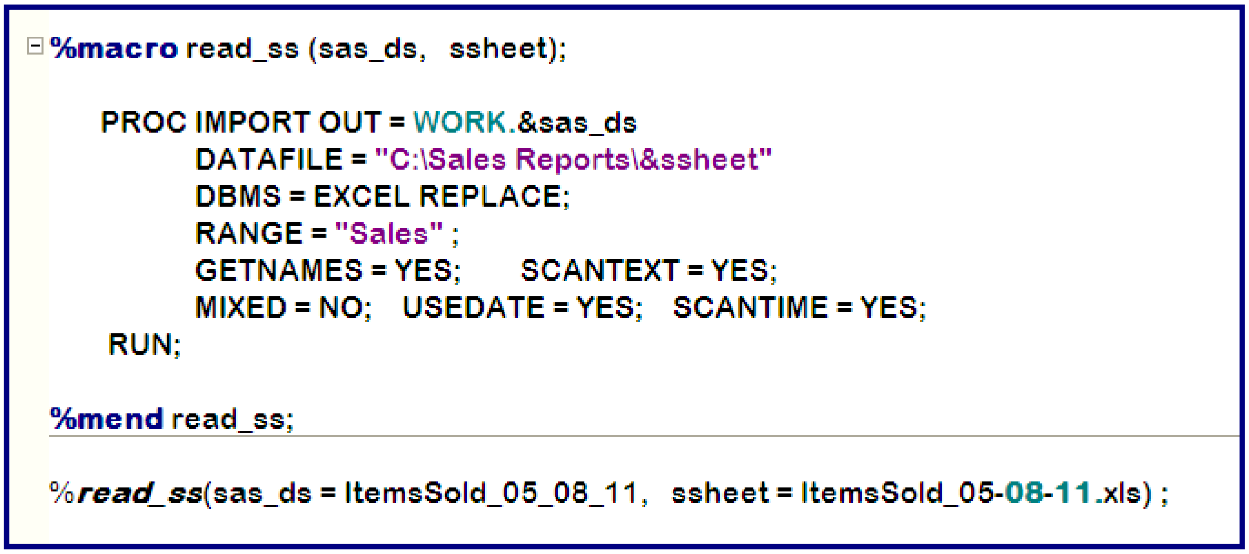
STEP 4: Modify the PROC IMPORT step so that parameters can be passed to it.



```
%let ssheet = ItemsSold_05-08-11.xls;
%let sas_ds = ItemsSold_05_08_11;

PROC IMPORT OUT= WORK.&sas_ds
         DATAFILE= "C:\Sales Reports\&ssheet"
         DBMS=EXCEL REPLACE;
   RANGE="Sales";
   GETNAMES=YES;    MIXED=NO;
   SCANTEXT=YES;    USEDATE=YES;
   SCANTIME=YES;
RUN;
```

Figure 7. Macro variables added to the PROC IMPORT step.

STEP 5: Convert the PROC IMPORT step into a macro program that can accept **KEYWORD** parameters.

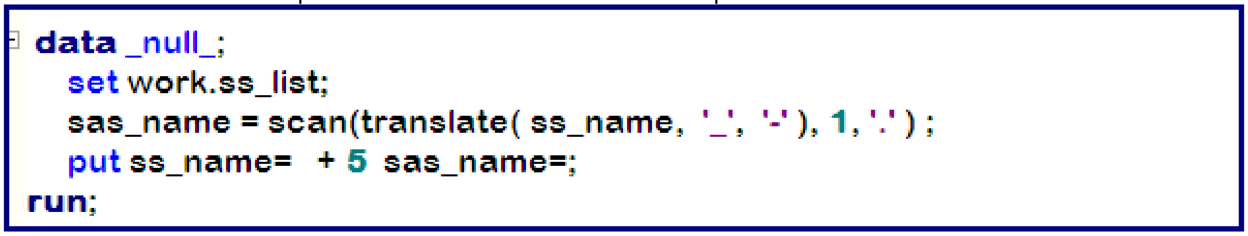


```
%macro read_ss (sas_ds,  ssheet);

    PROC IMPORT OUT = WORK.&sas_ds
            DATAFILE = "C:\Sales Reports\&ssheet"
            DBMS = EXCEL REPLACE;
            RANGE = "Sales" ;
            GETNAMES = YES;     SCANTEXT = YES;
            MIXED = NO;   USEDATE = YES;   SCANTIME = YES;
     RUN;

%mend read_ss;

%read_ss(sas_ds = ItemsSold_05_08_11,  ssheet = ItemsSold_05-08-11.xls) ;
```

.
Figure 8. The macro program,

Notice the name of macro program and the way it is called in the **%read_ss** statement. Again, notice the difference in the name of the spreadsheet and the name of the SAS dataset. The spreadsheet name is an invalid SAS name because it has dashes in it. What we want to do is create valid SAS names from the spreadsheet names.

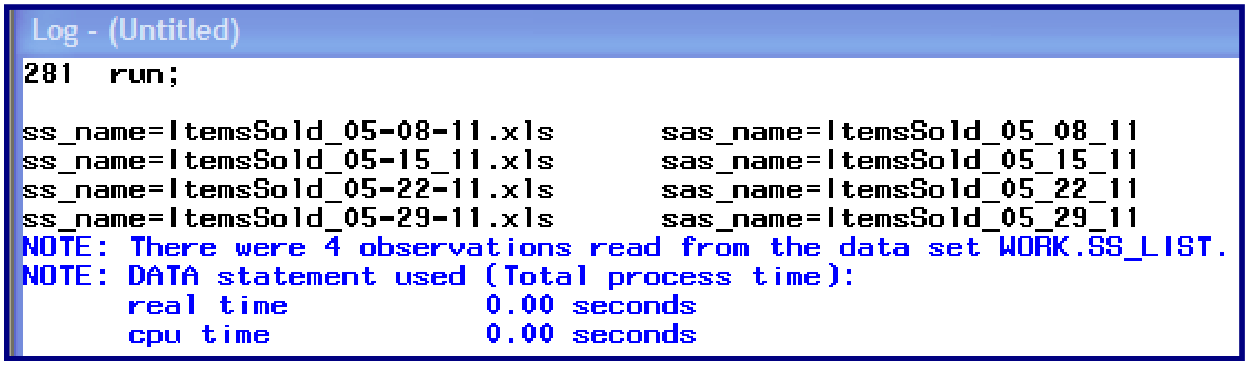
STEP 6: Write a DATA Step to create valid SAS names from the spreadsheet names.



```
data _null_;
   set work.ss_list;
   sas_name = scan(translate( ss_name,  '_',  '-' ), 1, '.' ) ;
   put ss_name=   + 5 sas_name=;
run;
```

Figure 9. The DATA step.

The **TRANSLATE** function has 3 arguments. The first is the character value to process. The second is the value to create, the third is the 'from' value. This code creates a '_' from a '-'**.** The **SCAN** function returns the all the characters up to the first **'.' .**



```
Log - (Untitled)
281  run;

ss_name=ItemsSold_05-08-11.xls      sas_name=ItemsSold_05_08_11
ss_name=ItemsSold_05-15-11.xls      sas_name=ItemsSold_05_15_11
ss_name=ItemsSold_05-22-11.xls      sas_name=ItemsSold_05_22_11
ss_name=ItemsSold_05-29-11.xls      sas_name=ItemsSold_05_29_11
NOTE: There were 4 observations read from the data set WORK.SS_LIST.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Figure 10. The SAS Log.

The SAS Log verifies that the TRANSLATE function worked the way we wanted it to work. The spreadsheet names were converted into valid SAS names.

There is one more piece of information we need to complete this process. We need to know about the **CALL EXECUTE** routine.

The **CALL EXECUTE** routine resolves the argument and issues the resolved value for the next step boundary. The syntax is:

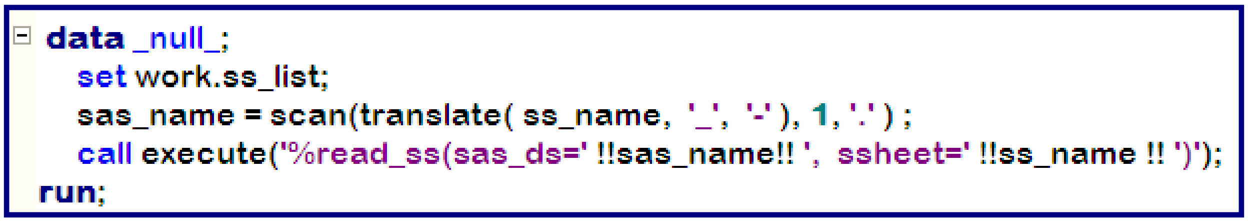
CALL EXECUTE(*argument*);

argument: specifies a character expression or a constant that yields a **macro invocation** or a SAS statement. ***Argument*** can be:

1. a character string, enclosed in quotation marks.
2. the name of a DATA step character variable. Do not enclose the name of the DATA step variable in quotation marks.
3. a character expression that the DATA step resolves to a macro text expression or a SAS statement.

If the argument resolves to a macro invocation, the macro executes immediately and DATA step execution pauses while the macro executes.

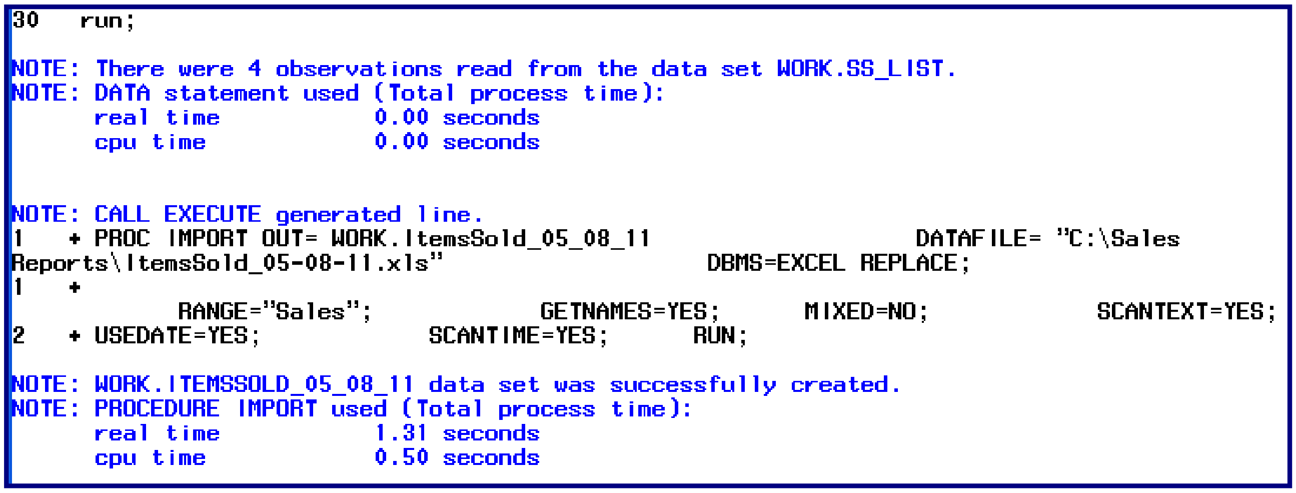
STEP7: Expand the DATA Step to execute the macro and pass to it values (parameters) that are read from the SS_LIST dataset.



```
data _null_;
   set work.ss_list;
   sas_name = scan(translate( ss_name, '_', '-' ), 1, '.' ) ;
   call execute('%read_ss(sas_ds=' !!sas_name!! ', ssheet=' !!ss_name !! ')');
  run;
```

Figure 11. The modified DATA step.

This DATA step executes once for every observation in the SS_List dataset (4 times). For every execution of this DATA step, an observation is read from the SS_List dataset, a valid SAS name is created from the spreadsheet name, then those values are passed into the macro program **%read_ss.** View the SAS Log…



```
30   run;

NOTE: There were 4 observations read from the data set WORK.SS_LIST.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


NOTE: CALL EXECUTE generated line.
1   + PROC IMPORT OUT= WORK.ItemsSold_05_08_11                   DATAFILE= "C:\Sales
Reports\ItemsSold_05-08-11.xls"                     DBMS=EXCEL REPLACE;
1   +
            RANGE="Sales";            GETNAMES=YES;       MIXED=NO;             SCANTEXT=YES;
2   + USEDATE=YES;             SCANTIME=YES;       RUN;

NOTE: WORK.ITEMSSOLD_05_08_11 data set was successfully created.
NOTE: PROCEDURE IMPORT used (Total process time):
      real time           1.31 seconds
      cpu time            0.50 seconds
```

Figure 12. The SAS Log.

This log shows the results of the first execution of the DATA step.

CONCLUSION

This seven step process outlined here shows how to write a DATA step to read a directory's contents, and create a SAS dataset that contains the names of all the spreadsheets in the directory. Next a macro program is created to read the spreadsheets and create a SAS dataset. And finally, a DATA step is used to execute the macro program and pass values to it that were read from the SAS dataset. Some of the power and flexibility of the DATA step is shown in this paper/presentation.

The author can be reached at:
Ben Cochran
The Bedford Group
3224 Bedford Avenue
Raleigh, NC 27607
(919) 741-0370
bencochran@nc.rr.com