Reading and Processing The Contents of a Directory (Excel Spreadsheets)

By

Ben Cochran

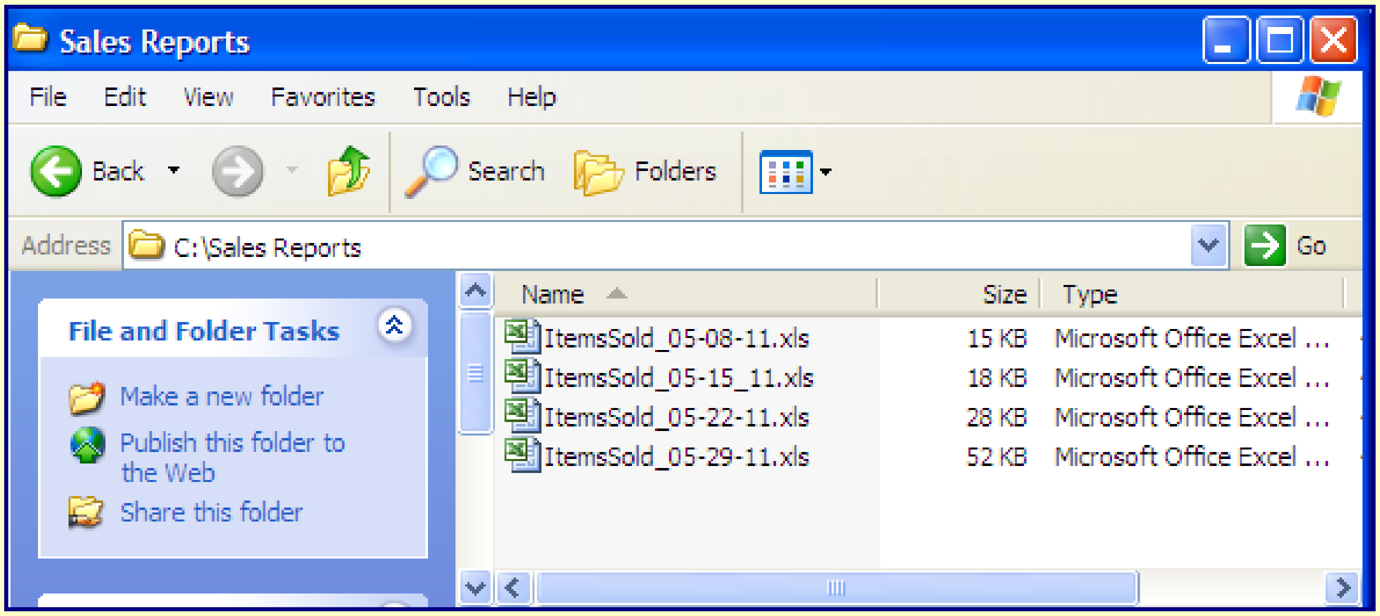
The Bedford Group, Inc.

An Affiliate Member of the SAS Alliance

1

Introduction

A company keeps its sale's reports in the "C:\Sales Report" directory.



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 ...

The data in these spreadsheets need to be converted into SAS datasets.
Notice the names of the spreadsheets.

Retrieving Directory Information

Use the **DOPEN, DNUM, DREAD** and **DCLOSE** functions to process a directory.

- 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) **;**

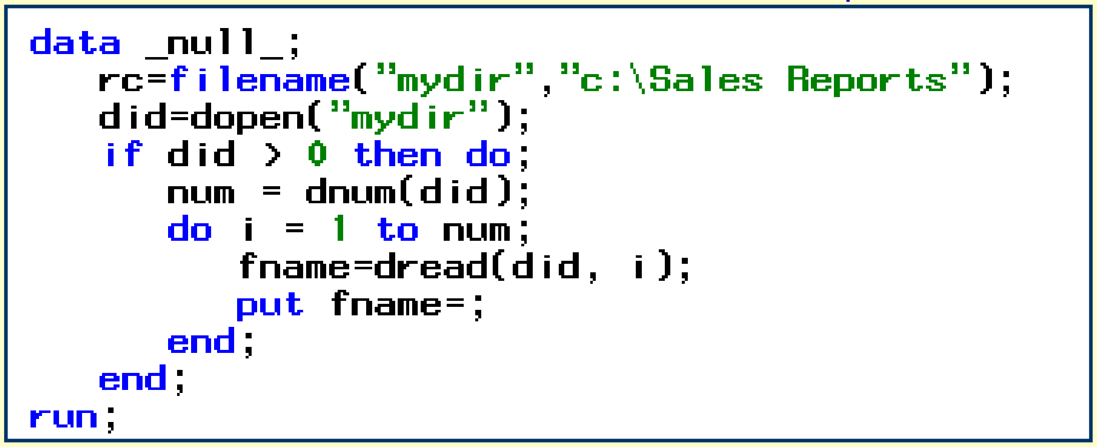
- The **DREAD** function - returns the name of a directory member
  syntax: name = **DREAD**(directory_id, member-number)
  example: filename=DREAD(directory_id, i) **;**

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

Retrieving Directory Information

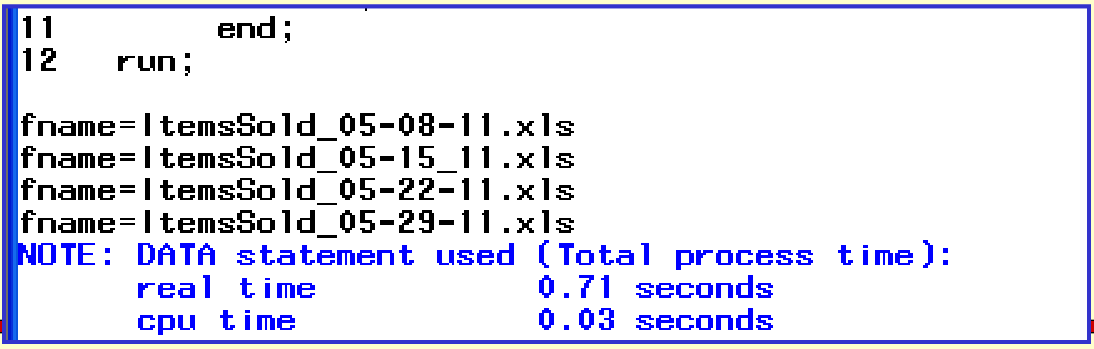
Step 1: Write a DATA Step that will read this directory and each file within it.

Use the **DOPEN, DNUM,** and **DREAD** functions to process a directory.



```
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;
```

The partial log displays the results of the PUT statements.

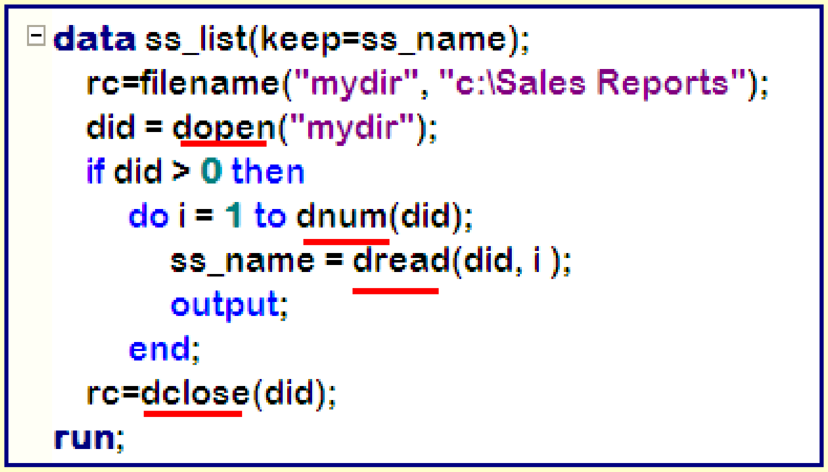


```
11         end;
12    run;

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
```

4

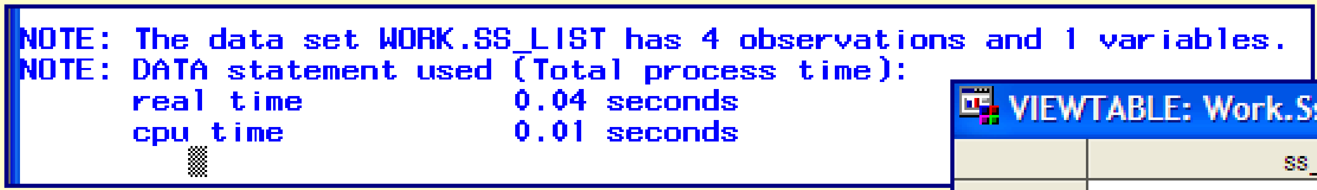
Retrieving Directory Information

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



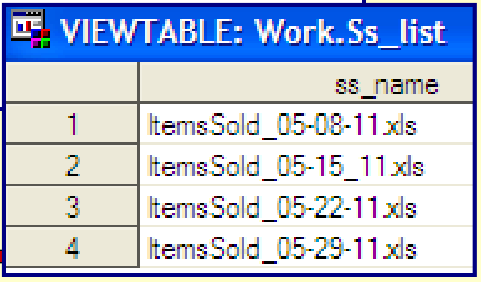
```
data ss_list(keep=ss_name);
   rc=filename("mydir", "c:\Sales Reports");
   did = dopen("mydir");
   if did > 0 then
      do i = 1 to dnum(did);
         ss_name = dread(did, i );
         output;
      end;
   rc=dclose(did);
run;
```

The partial log displays the results of the DATA step..



```
NOTE: The data set WORK.SS_LIST has 4 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.01 seconds
```

The **WORK.SS_LIST** dataset looks like this….

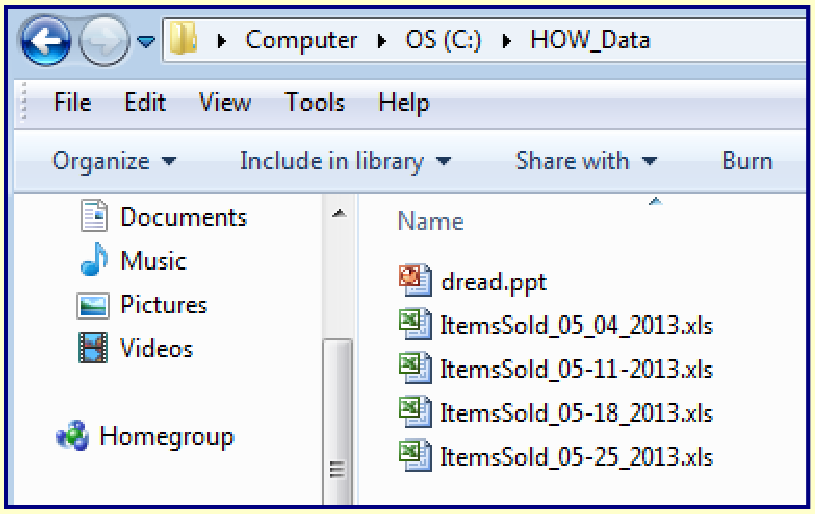


VIEWTABLE: Work.Ss_list

	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

Workshop 1

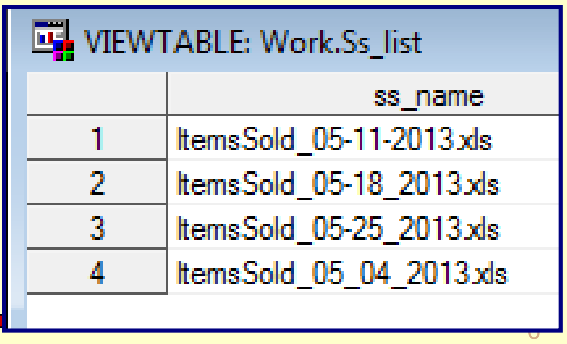
Exercise 1: Write a DATA step to read the contents of the **'c:\HOW_Data'** directory and create a SAS dataset containing only the **Excel spreadsheets.**


Computer ▸ OS (C:) ▸ HOW_Data
File Edit View Tools Help
Organize ▾ Include in library ▾ Share with ▾ Burn
Documents
Music
Pictures
Videos
Homegroup
Name

dread.ppt
ItemsSold_05_04_2013.xls
ItemsSold_05-11-2013.xls
ItemsSold_05-18_2013.xls
ItemsSold_05-25_2013.xls

What are the file 'extensions' of these files?

The **WORK.SS_LIST** dataset looks like this: ➪



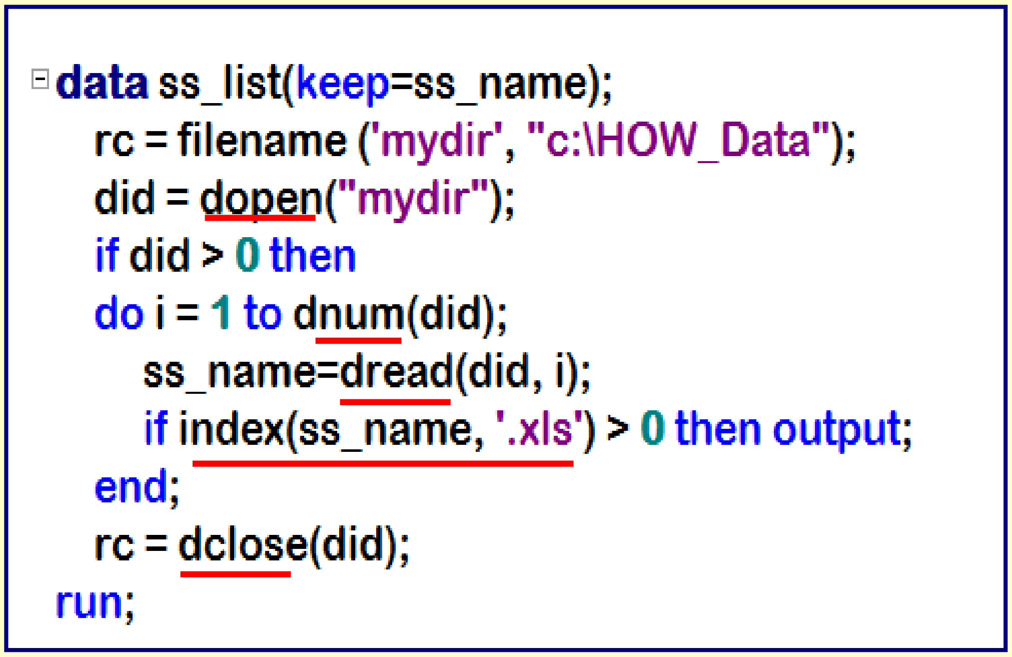
VIEWTABLE: Work.Ss_list

	ss_name
1	ItemsSold_05-11-2013.xls
2	ItemsSold_05-18_2013.xls
3	ItemsSold_05-25_2013.xls
4	ItemsSold_05_04_2013.xls

Notice the **Names** of the spreadsheets!

Workshop 1 - Solution

Exercise 1 Solution:



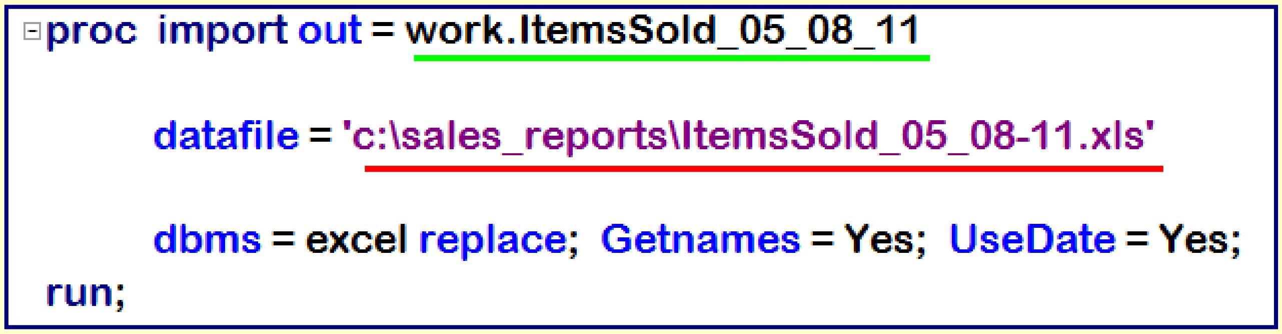
```
data ss_list(keep=ss_name);
   rc = filename ('mydir', "c:\HOW_Data");
   did = dopen("mydir");
   if did > 0 then
   do i = 1 to dnum(did);
      ss_name=dread(did, i);
      if index(ss_name, '.xls') > 0 then output;
   end;
   rc = dclose(did);
run;
```

Notice the INDEX function in the IF statement.

7

Importing Spreadsheet Data to SAS

Step 3: Next, write a program to import the first spreadsheet and create a SAS dataset.

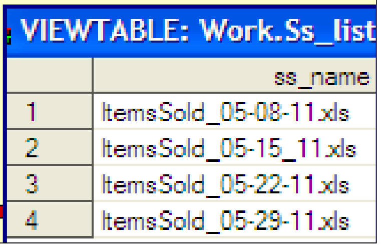


```
proc  import out = work.ItemsSold_05_08_11

        datafile = 'c:\sales_reports\ItemsSold_05_08-11.xls'

        dbms = excel replace;  Getnames = Yes;  UseDate = Yes;
run;
```

Selected **PROC IMPORT Options and Statements:**

- **DBMS =** specifies the type of data to import (csv, mdb, txt, dml, or xls) - use **XLS** if you are running **64 bit SAS** with **32 bit Excel.**
- **GETNAMES =** determines whether to generate SAS column names from the column names in the **first row** of data
- **REPLACE** replaces an existing SAS data set with the same name.
- **USEDATE =** specifies whether to assign a date format **while importing a column of data.**

The spreadsheet names are NOT valid SAS dataset names. →

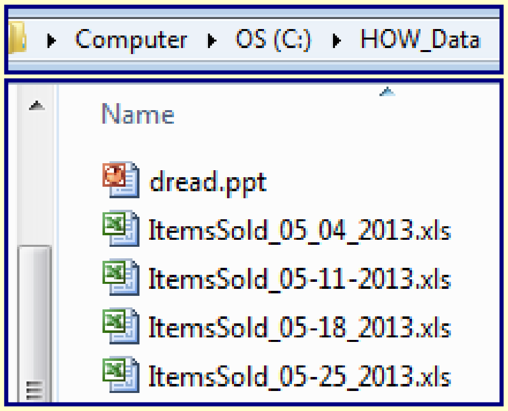


VIEWTABLE: Work.Ss_list

	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

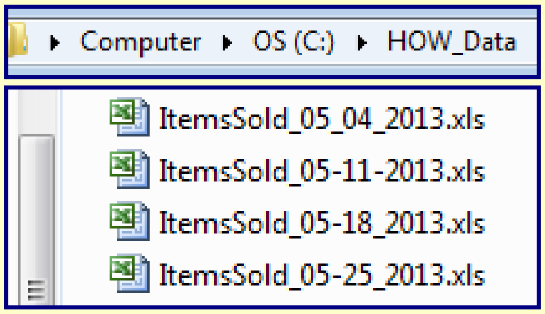
Workshop 2

Exercise 2: Write a program to import the **first** spreadsheet and create a SAS dataset named: work.ItemsSold_05_04_2013. Use PROC IMPORT to accomplish this. Pay close attention to the names of the spreadsheets.



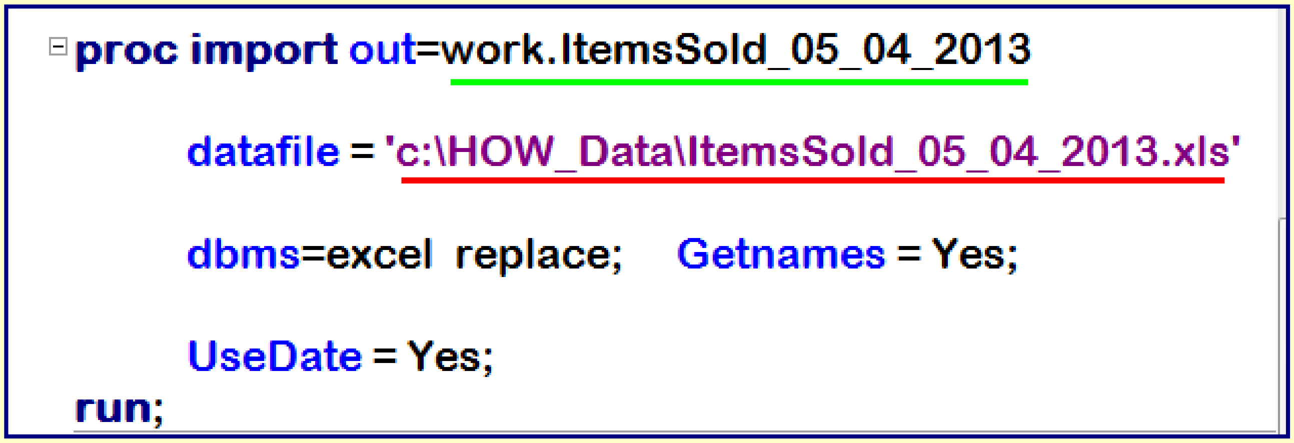
```
proc import out = work.__________________
        datafile = '__________________.xls'
        dbms =  ________  replace;
        getnames = ____;  UseDate = ____;
run;
```

Workshop 2 - Solution


Computer ▸ OS (C:) ▸ HOW_Data

ItemsSold_05_04_2013.xls
ItemsSold_05-11-2013.xls
ItemsSold_05-18_2013.xls
ItemsSold_05-25_2013.xls

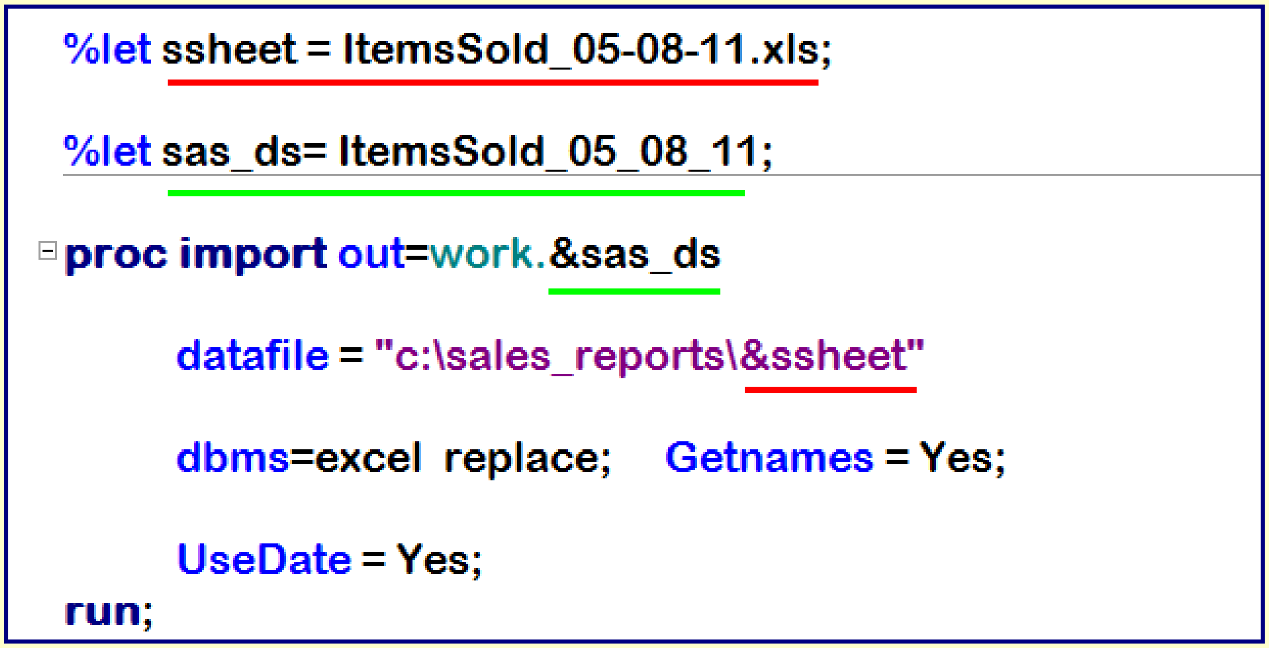
Exercise 2 Solution:


```
proc import out=work.ItemsSold_05_04_2013
        datafile = 'c:\HOW_Data\ItemsSold_05_04_2013.xls'
        dbms=excel  replace;    Getnames = Yes;
        UseDate = Yes;
run;
```

Note: If you are running a 64 bit version of SAS and have a 32 bit version of Microsoft Office Excel, you will have to use **xls** as the value for the **DBMS** option.

Importing Spreadsheet Data to SAS

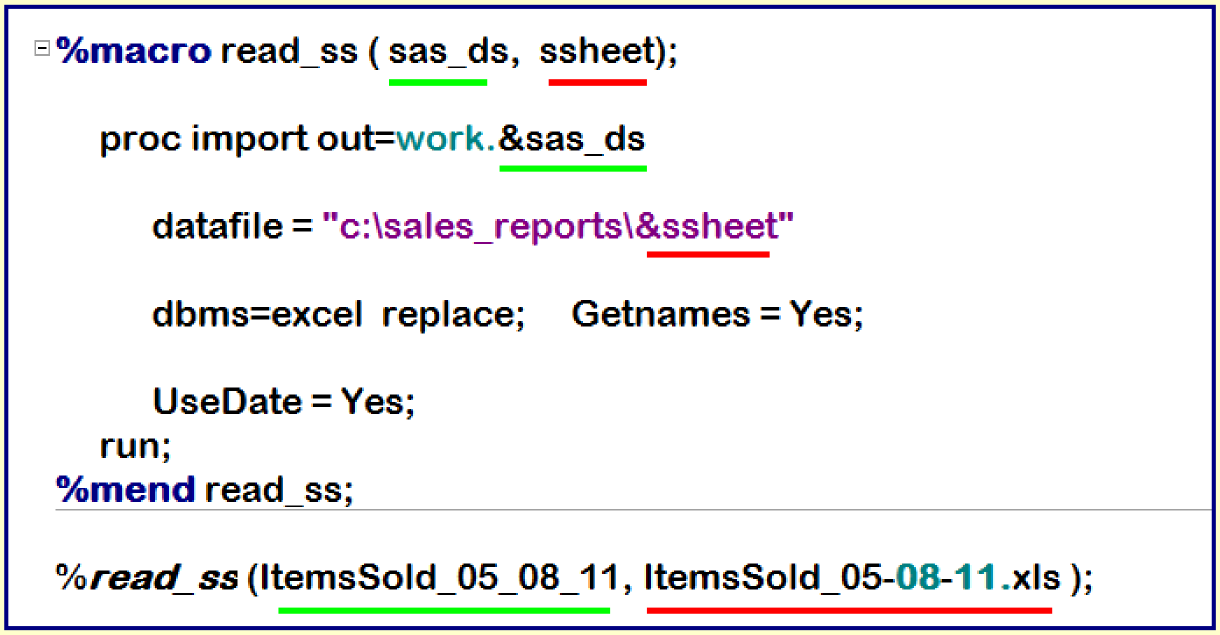
Step 4: Modify the PROC IMPORT step so that parameters can be passed to it.



```
%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;    Getnames = Yes;

        UseDate = Yes;
run;
```

Importing Spreadsheet Data to SAS

Step 5: Convert the IMPORT step into a **macro program** that accepts parameters.

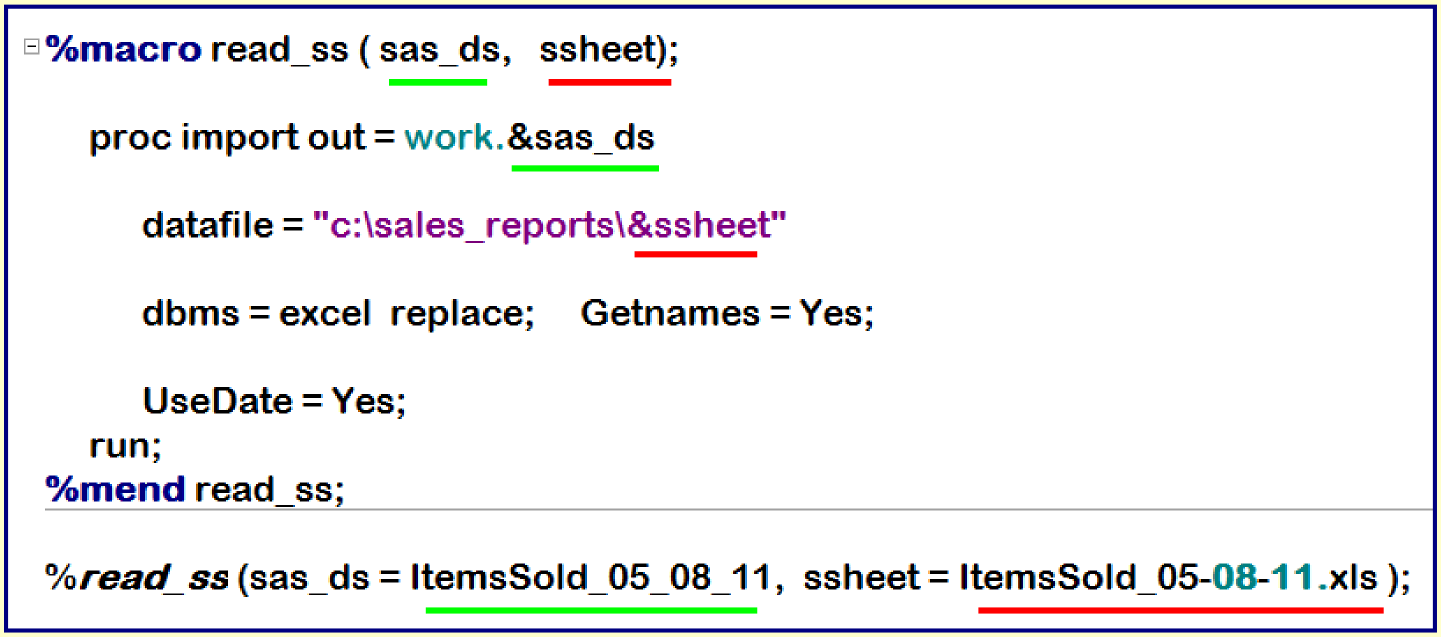


```
%macro read_ss ( sas_ds,  ssheet);

   proc import out=work.&sas_ds

      datafile = "c:\sales_reports\&ssheet"

      dbms=excel  replace;    Getnames = Yes;

      UseDate = Yes;
   run;
%mend read_ss;

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

The name of the macro program is **READ_SS.** It is called one time with the name of the first spreadsheet and first SAS dataset passed to it.

Importing Spreadsheet Data to SAS

Step 5b: Modify the READ_SS macro so that it will accept **KEYWORD** parameters.



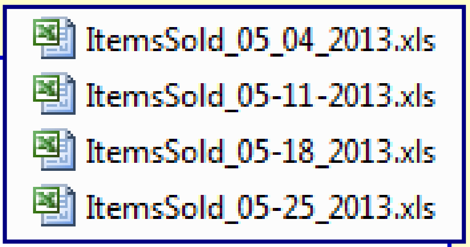
```
%macro read_ss ( sas_ds,   ssheet);

    proc import out = work.&sas_ds

        datafile = "c:\sales_reports\&ssheet"

        dbms = excel  replace;    Getnames = Yes;

        UseDate = Yes;
    run;
%mend read_ss;

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

The name of the macro program is **READ_SS.** It is called one time with the name of the first spreadsheet and first SAS dataset passed to it.

13

Workshop 3

Exercise 3: Write a SAS Macro program that contains a PROC IMPORT to read the **first** spreadsheet (to be passed as a keyword parameter) and create a SAS dataset (to be passed as a keywork parameter).



ItemsSold_05_04_2013.xls
ItemsSold_05-11-2013.xls
ItemsSold_05-18_2013.xls
ItemsSold_05-25_2013.xls

```
%macro  read_ss (sas_ds,   ssheet );
      proc import out = work.&sas_ds
         datafile = "c:\HOW_data\&ssheet"
         dbms = excel  replace;
         getnames = Yes;   UseDate = Yes;
      run;
%mend read_ss;

%read_ss ( sas_ds = ____________  ,  ssheet = ___________.xls ) ;
```

14

Workshop 3 - Solution

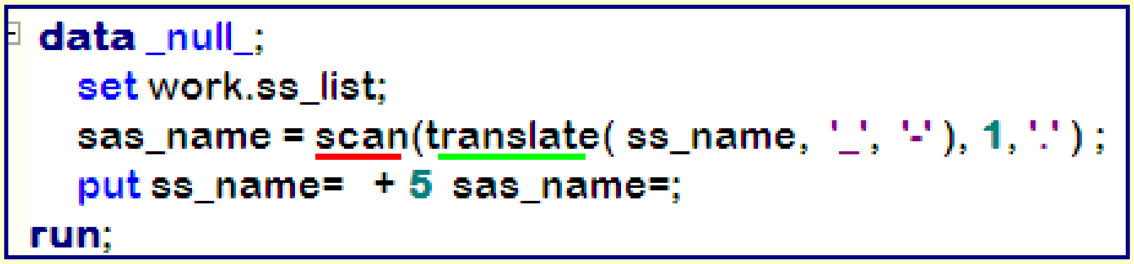
Exercise 3 Solution:

```
%macro  read_ss (sas_ds,   ssheet );
      proc import out = work.&sas_ds
         datafile = "c:\HOW_data\&ssheet"
         dbms = excell replace;
         getnames = Yes;   UseDate = Yes;
      run;
%mend read_ss;

%read_ss ( sas_ds = ItemsSold_05_04_2013 ,
                ssheet   = ItemsSold_05_04_2013.xls ) ;
```

15

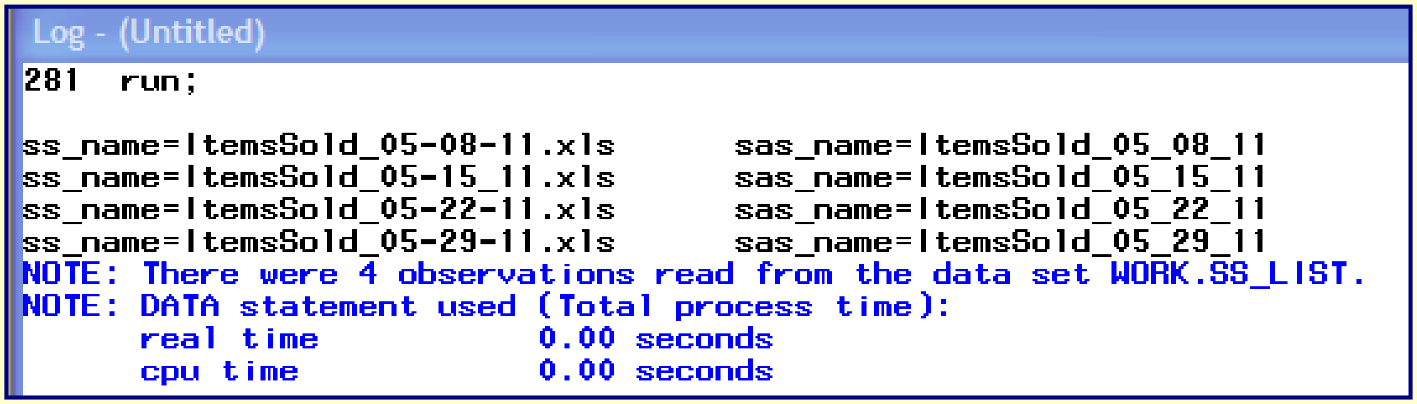
Writing the DATA Step

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;
```

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
```

CALL EXECUTE

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:

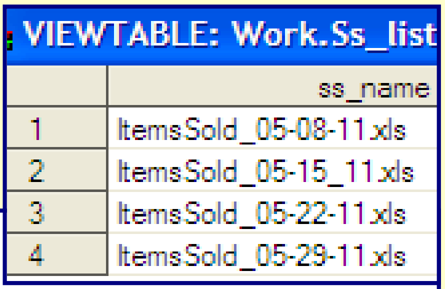
1. a character string, enclosed in quotation marks.
2. the name of a DATA step character variable. Do not enclose the name of the DATA step variable in quotation marks.
3. a character expression that the DATA step resolves to a macro text expression or a SAS statement.

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

17

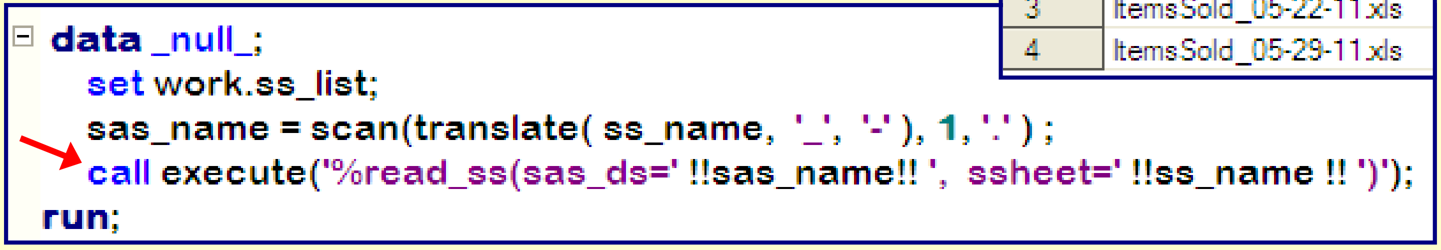
CALL EXECUTE

Step 7: Expand the DATA Step to execute the macro and pass to it values (parameters) that are read from the **SS_LIST** data set. .

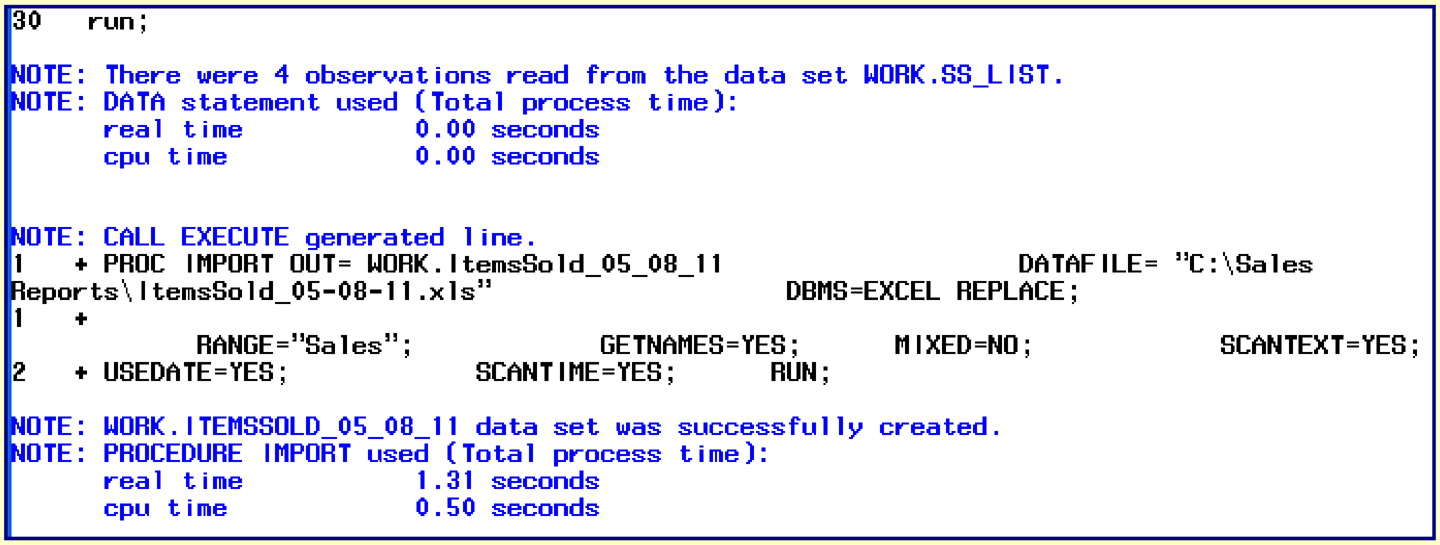


VIEWTABLE: Work.Ss_list

	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



```
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;
```



```
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
```

Workshop 4 and Solution

Exercise 4: Write a DATA step to call the macro program created in Workshop 3.

```
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;
```

Creating Directories (Optional)

You can use the **DCREATE** function to create new directories.   The typical syntax of the DCREATE function is:

```
new-directory = DCREATE (directory-name<,parent-directory>) ;
```

Where: **new-directory** contains the complete pathname of the new directory, or it will be blank if the new directory cannot be created.

directory-name specifies the name of the directory to create. This value cannot include a pathname.

parent-directory contains the complete pathname of the directory in which

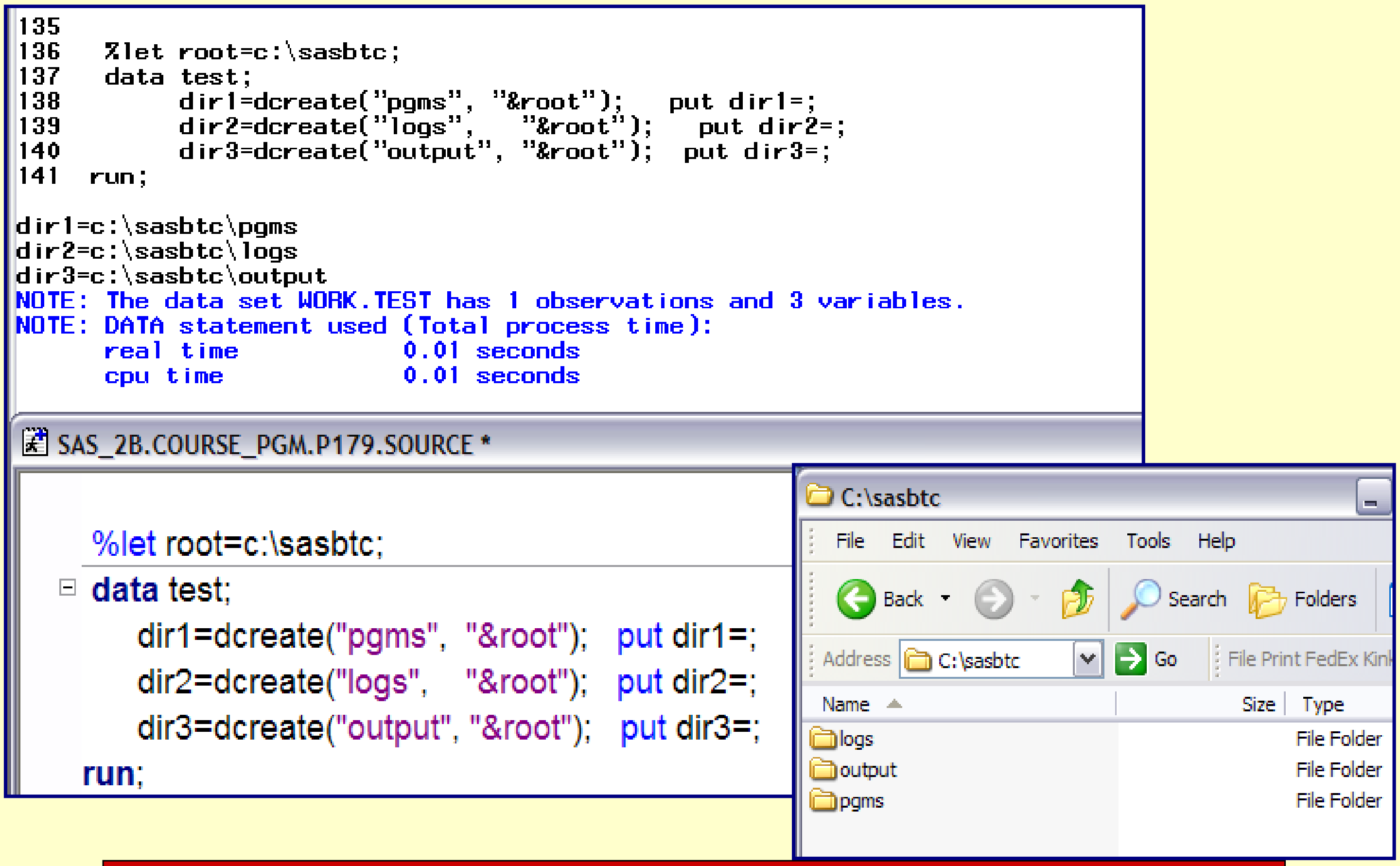
to create the new directory.   If you do not supply a value for the parent-directory, then the current directory is the parent directory.

Task: Write a DATA Step that will create a new 'main' directory and some 'sub' directories.

20

Creating Directories

Use the log to verify the creation of the directories.



```
135
136   %let root=c:\sasbtc;
137   data test;
138        dir1=dcreate("pgms", "&root");   put dir1=;
139        dir2=dcreate("logs",   "&root");   put dir2=;
140        dir3=dcreate("output", "&root");  put dir3=;
141   run;

dir1=c:\sasbtc\pgms
dir2=c:\sasbtc\logs
dir3=c:\sasbtc\output
NOTE: The data set WORK.TEST has 1 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

SAS_2B.COURSE_PGM.P179.SOURCE *

```
%let root=c:\sasbtc;
data test;
   dir1=dcreate("pgms",   "&root");   put dir1=;
   dir2=dcreate("logs",    "&root");   put dir2=;
   dir3=dcreate("output", "&root");   put dir3=;
run;
```

Name	Size	Type
logs		File Folder
output		File Folder
pgms		File Folder