

Daunting Tasks Made Simple with Input Techniques and Macro Processing

Denise A. Kruse, Alpharetta, Georgia

ABSTRACT

Transferring SAS® data sets from a PC to a UNIX server should not be a difficult task. However, preserving the PC directory path to create an identical path on the UNIX server adds an element of difficulty to the task. When faced with unknown quantities of directories and data sets, it is best to leave the work to SAS.

The SAS tools which save the day in this scenario are the character pointer technique for parsing a file, in combination with, relative column pointer control and several SAS functions. Macro processing and do loops allow automation to create new UNIX directories.

INTRODUCTION

My task was to load SAS data sets from digital linear tapes (DLT) and place them on the hard drive of a remote PC. I needed to migrate version 6 (32 bit) data sets to 64 bit data sets. It seemed easy enough to FTP the data sets to a UNIX environment until I realized that the directory structure needed to be preserved.

Regardless of the reason for transferring the data, the code can be reused in the future for other projects.

IDENTIFYING THE DATA SETS

In order to find an automated way to identify the SAS data sets that I wanted to capture, I needed to create an inventory of the data on the hard drive. This example listing of data was created from the DOS prompt by using DOS commands:

```
dir *.ssd01* /S > dos_listing.txt
```

This listing is a partial example of the data to be parsed:

```
Volume in drive D is SAS Backup
Volume Serial Number is ABC-3128

Directory of D:\mysas\development\vendor1

11/18/1999 12:00a      730,641 myvendor.ssd01.Z
08/09/2000 12:00a      987,645 abcvendor.sas7bdat.Z
12/01/1999 12:00a      731,349 first.ssd01.Z
11/09/1999 12:00a      731,117 oldvendor.ssd01.Z
4 File(s)    3,180,752 bytes

Directory of D:\mysas3\development\alpharetta\georgia

08/09/2000 12:00a      9,645 eateries.sas7bdat.Z
07/14/1999 12:00a      16,500 playgrounds.ssd01.Z
07/07/1999 12:00a      7,243 parks.ssd01.Z
07/07/1999 12:00a      12,468 recreation.ssd01.Z
07/14/1999 12:00a      16,542 recreation2.ssd01.Z
5 File(s)    62,398bytes
```

This text file provided a list of all files that were residing on the DLT tape and had been transferred to the hard drive of the remote computer. Once I analyzed the text file, I was able to devise a method to read the directory paths and then capture the data sets belonging to that path.

BRINGING THE DIRECTORY PATHS AND DATA SET NAMES INTO SAS

A common INFILE statement uses absolute pointer control (@ symbol) to indicate where the desired data should begin. The @nth value will always be the starting position for the variable name that follows, whether it is the first, middle or last record of the file.

```
INFILE '/mysas/myfile.txt' TRUNCOVER ;
INPUT  @1  var1
        @11 var2 ;
```

The character pointer technique uses hard coded values to indicate where to start reading in a value for the variable.

```
INFILE '/mysas/myfile.txt' TRUNCOVER ;
INPUT  @ "Name"  var1
        @ "School" var2 ;
```

In analyzing the dos_listing.txt, I noticed that I could use the "D:" to indicate where each directory path started. Using the character pointer technique, SAS will read starting in the position directly following the hardcoded value in quotation marks and read the entire line until a blank space is encountered. PC paths normally have spaces and if this is the case for your situation, add an ampersand as noted in the next example.

```
INFILE '/mysas/dos_listing.txt' TRUNCOVER;
INPUT  @ "D:" dir_path ;
```

The next line was a little more difficult to capture. For this I would need to manipulate the relative column pointer. The data set names were variable in length and I did not want to run the risk of truncation to the data set name. Moving the pointer to the end of the record (using + sign for relative input) and moving from right to left by 20 bytes enabled me to capture the data set name and the zipped file extension. The ampersand allows for spaces in the 20 bytes retrieved.

```
INFILE '/mysas/dos_listing.txt' TRUNCOVER ;
INPUT  @ "D:" dir_path
        +(-20) dsn & ;
```

RESULTS

There are two variables in the newly created SAS data set (Figure 1). The input method used in the previous step is not perfect as it captures some additional text "noise" that needs to be handled. However, it does capture the necessary information. To solve this issue with an additional data step, we only want to choose values for the dir_path variable if it is populated. The following conditional statement will be used:

```
if dir_path ne '' then new_path=dir_path;
```

Additionally, you can see by looking at the values in the dsn field, that we only want the data with the version 6 extensions. The following conditional statement will be used with the INDEX function and SCANQ function:

```
if index(dsn, '.ssd01') gt 0 then new_dsn=scanq(dsn,-1);
else new_dsn="";
```

Figure 1

READMEIN.sas7bdat	
dir_path	dsn
	rive D is SAS Backup
	I Number is ABC-3128
\mysas\development\vendor1	\development\vendor1
	641 myvendor.ssd01.Z
	abcvendor.sas7bdat.Z
	31,349 first.ssd01.Z
	17 oldvendor.ssd01.Z
	3,180,752 bytes
\mysas3\development\alpharetta\georgia	t\alpharetta\georgia
	eateries.sas7bdat.Z
	playgrounds.ssd01.Z
	7,243 parks.ssd01.Z
	8 recreation.ssd01.Z
	recreation2.ssd01.Z
	62,398bytes

INDEX is used to locate the starting position of a string. The search is for the literal value '.ssd01' and if that value is not found, INDEX function returns a zero value. This statement is conditional upon finding the value in the field.

SCANQ is used to extract a specified "word" from a character expression. "Word" is defined as characters separated by a set of specified delimiters. When a negative value is used (such as in this example), the scan executes from right to left. Additionally, when a specified delimiter is omitted, the default set of delimiters are white spaces.

MODIFYING THE SAS DATA USING SAS FUNCTIONS

Now, we have to get the data in a format that UNIX recognizes. Two things that need to be modified in the variables above:

- create the path on the UNIX server where the data sets need to reside
- change back slashes to forward slashes

Our directory path on the UNIX server is:

/activities/development/denise/data

We need to join this to the value in the variable dir_path:

\mysas\development\vendor1

The UNIX path needs to be joined with the value read in the DOS listing text file. Basic concatenation would yield this result:

/activities/development/denise/data\mysas\development\vendor1

This is not a valid path for the UNIX server due to the back slashes in the latter part of the path. In modifying this path, we will use two SAS functions: CATS and TRANWRD:

mkme=TRANWRD(CATS('/activities/development/denise/data,new_path),'\'/');

- CATS: Joins two values in variables, text strings or a combination of the two while stripping leading and trailing blanks
- TRANWRD function substitutes one character string to another

mkme

```
/activities/development/denise/data/mysas/development/vendor1
/activities/development/denise/data/mysas/development/vendor1
/activities/development/denise/data/mysas/development/vendor1
/activities/development/denise/data/mysas3/development/alpharetta/georgia
/activities/development/denise/data/mysas3/development/alpharetta/georgia
/activities/development/denise/data/mysas3/development/alpharetta/georgia
/activities/development/denise/data/mysas3/development/alpharetta/georgia
```

You will notice that there are multiple duplicate lines that appear in these results. This is not an error: each of those duplicate paths map to a unique SAS data set name which is not needed to create the UNIX path. Therefore, duplicate records are created. A sort should be executed before building the macro code that follows.

Thinking ahead to creating the UNIX directories, I want to make an additional new location for each new path: **/out**. When the data sets are transferred into UNIX, they will initially reside in the <new_path> directory. After the conversion to the 64 bit data sets, the finished version will reside in the <new_path>/**out** directory.

This SAS code will run interactively on the UNIX server and use X commands to invoke the UNIX syntax such as MKDIR (command to create a new directory). The goal is to have all of these line commands contained inside the value of a SAS variable. This is what will give the code automation (see variables mkme2 and mkme3).

Reattach the existing data set name to the new path using the CATS function. Please note this variable will be used in the FTP scripting which is not covered in this paper.

```
full_path=cats(new_path,'\',new_dsn);
```

full_path

```
\mysas\development\vendor1\myvendor.ssd01.Z
\mysas\development\vendor1\first.ssd01.Z
\mysas\development\vendor1\oldvendor.ssd01.Z
\mysas3\development\alpharetta\georgia\playgrounds.ssd01.Z
\mysas3\development\alpharetta\georgia\parks.ssd01.Z
\mysas3\development\alpharetta\georgia\recreation.ssd01.Z
\mysas3\development\alpharetta\georgia\recreation2.ssd01.Z
```

When the data is transferred back to the hard drive of the remote computer, the path will need to include the “D:” structure again. Please note this variable will be used in the FTP scripting which is not covered in this paper.

```
dfull_path=cats('D:',new_path,'\',new_dsn);
```

dfull_path

```
D:\mysas\development\vendor1\myvendor.ssd01.Z
D:\mysas\development\vendor1\first.ssd01.Z
D:\mysas\development\vendor1\oldvendor.ssd01.Z
D:\mysas3\development\alpharetta\georgia\playgrounds.ssd01.Z
D:\mysas3\development\alpharetta\georgia\parks.ssd01.Z
D:\mysas3\development\alpharetta\georgia\recreation.ssd01.Z
D:\mysas3\development\alpharetta\georgia\recreation2.ssd01.Z
```

Constructing the X command statement to make the new UNIX directories and all the subdirectories (-p):

```
z1='x ';  
z2='mkdir -p ';  
mkme2=catx(" ",z1,"",z2,mkme,"");  
mkme3=z1||""||z2||compress(mkme||'/out')||"";  
fin =compress("""||mkme||""");  
fout=compress("""||compress(mkme||'/out')||""");
```

COMPRESS allows you to remove specific characters from a character value. In this case, it is spaces.

CATX concatenates two or more character strings while stripping both leading and trailing blanks and inserting one or more separator characters between the strings.

RESULTS: COMMAND LINE SYNTAX EMBEDDED WITHIN SAS VARIABLE VALUES

NOTE: ALL OF THE FOLLOWING VARIABLES RESIDE IN THE SAS DATA SET "MKLIST"

mkme2

```
x ' mkdir -p /activities/development/denise/data/mysas/development/vendor1 '  
x ' mkdir -p /activities/development/denise/data/mysas/development/vendor1 '  
x ' mkdir -p /activities/development/denise/data/mysas/development/vendor1 '  
x ' mkdir -p /activities/development/denise/data/mysas3/development/alpharetta/georgia '  
x ' mkdir -p /activities/development/denise/data/mysas3/development/alpharetta/georgia '  
x ' mkdir -p /activities/development/denise/data/mysas3/development/alpharetta/georgia '  
x ' mkdir -p /activities/development/denise/data/mysas3/development/alpharetta/georgia '
```

mkme3

```
x 'mkdir -p /activities/development/denise/data/mysas/development/vendor1/out'  
x 'mkdir -p /activities/development/denise/data/mysas/development/vendor1/out'  
x 'mkdir -p /activities/development/denise/data/mysas/development/vendor1/out'  
x 'mkdir -p /activities/development/denise/data/mysas3/development/alpharetta/georgia/out'  
x 'mkdir -p /activities/development/denise/data/mysas3/development/alpharetta/georgia/out'  
x 'mkdir -p /activities/development/denise/data/mysas3/development/alpharetta/georgia/out'  
x 'mkdir -p /activities/development/denise/data/mysas3/development/alpharetta/georgia/out'
```

fin (*this variable is used for FTP purposes not covered in this paper*)

```
 '/activities/development/denise/data/mysas/development/vendor1 '  
 '/activities/development/denise/data/mysas/development/vendor1 '  
 '/activities/development/denise/data/mysas/development/vendor1 '  
 '/activities/development/denise/data/mysas3/development/alpharetta/georgia '  
 '/activities/development/denise/data/mysas3/development/alpharetta/georgia '  
 '/activities/development/denise/data/mysas3/development/alpharetta/georgia '  
 '/activities/development/denise/data/mysas3/development/alpharetta/georgia '
```

fout (*this variable is used for FTP purposes not covered in this paper*)

```
 '/activities/development/denise/data/mysas/development/vendor1/out '  
 '/activities/development/denise/data/mysas/development/vendor1/out '  
 '/activities/development/denise/data/mysas/development/vendor1/out '  
 '/activities/development/denise/data/mysas3/development/alpharetta/georgia/out '  
 '/activities/development/denise/data/mysas3/development/alpharetta/georgia/out '  
 '/activities/development/denise/data/mysas3/development/alpharetta/georgia/out '  
 '/activities/development/denise/data/mysas3/development/alpharetta/georgia/out '
```

BUILDING THE AUTOMATION

CREATE "IN" DIRECTORY

Create a macro variable with the path name and record number (mk) followed by a macro variable representing number of records in the data set (cnt). The automatic SAS variable _n_ is ideal for this task. This will represent the upper limit value for number of loop executions. The macro loops through the data executing the UNIX commands.

```
data _null_;
set mklst end=no_more;
call symputx('mk'||left(_n_),mkme2);
if no_more then call symputx('cnt',_n_);
run;

%macro loop1;
%local i;
%do i=1 %to &cnt. ;
%macro makeme(dd1);
&dd1.;
%mend makeme;
%makeme(&&mk&i.);
%end;
%mend loop1;
%loop1;
```

CREATE "OUT" DIRECTORY

Following the same steps as for the "IN" directory, create a macro variable with the path name and record number (mk2) followed by a macro variable with number of records in the data _set (cnt). This will represent the upper limit value for number of loop executions.

```
data _null_;
set mklst end=no_more;
call symputx('mk2'||left(_n_),mkme2);
if no_more then call symputx('cnt',_n_);
run;

%macro loop2;
%local i;
%do i=1 %to &cnt. ;
%macro makeme2(dd1);
&dd1.;
%mend makeme2;
%end;
%mend loop2;
%loop2;
```

After sorting the data to remove duplicate directories, the code executes in perfect fashion with a SAS data set as the driving force behind the automation. For each record in the data set, a UNIX directory is created.

CONCLUSION

This task of work was very daunting and overwhelming because of the unknown number of data sets, directory locations and tapes. This paper shows one method of simplifying a potentially monstrous process by using different input techniques and macro processing. The time spent to think through the process and create code to automate the time consuming part was well worth the effort as it runs behind the scenes and is re-useable code. Additionally, creation of the code to automate the process made the actual task interesting because it was challenging at times to create the DOS commands, SAS commands and scripts.

When faced with a daunting task, take a step back and think about how you can make it your own.

EXAMPLE OF FULL PROGRAM

```
options symbolgen mprint mlogic source;
libname ttl '/activities/development/denise/data';

data readmein ;
format dir_path $200. dsn $100. ;
infile '/activities/development/denise/data/dos_listing.txt' trunccover ;
input      @ "D:" dir_path
           +(-20) dsn & ;
run;

data ttl.fixme(keep=full_path dfull_path mkme2 mkme3 mkme fdir fin fout);
length mkme2 $500.;
set readmein;
retain new_path;

if dir_path ne '' then new_path=dir_path;
if index(dsn, '.ssd01') gt 0 then new_dsn=scanq(dsn, -1);
                           else new_dsn='';
if new_path ne '' and new_dsn ne '' then do;
    mkme=tranwrd(cats('/activities/development/denise/data', new_path), '\', '/');
    full_path=cats(new_path, '\', new_dsn);
    dfull_path=cats('D:', new_path, '\', new_dsn);
    fdir=cats('D:', new_path) ;
    z1='x ' ;
    z2='mkdir -p ' ;
    mkme2=catx(" ", z1, "", z2, mkme, "");
    mkme3=z1||""||z2||compress(mkme||'/out')||"";
    fin =compress(" "||mkme||"");
    fout=compress(" "||compress(mkme||'/out')||"");           end;

if full_path ne '' then output; run;

proc sort nodupkey data=ttl.fixme out=ttl.mklst ; by mkme2; run ;
proc sort nodupkey data=ttl.fixme out=mklst2 ; by mkme3; run ;

/* CREATE "IN" DIRECTORY: BEGIN */
data _null_ ;
set ttl.mklst end=no more;
call symputx('mk'||left(_n_), mkme2);
if no_more then call symputx('cnt', _n_);
run;

%macro loop1;
%local i;
%do i=1 %to &cnt. ;
%macro makeme(dd1);
&dd1.;
%mend makeme;
%makeme(&&mk&i.);
%end;
%mend loop1;
%loop1;
/* CREATE "OUT" DIRECTORY: BEGIN */
data _null_ ;
set mklst2 end=no more;
call symputx('mk2'||left(_n_), mkme3);
if no_more then call symputx('cnt', _n_);
run;

%macro loop2;
%local i;
%do i=1 %to &cnt. ;
%macro makeme2(dd1);
&dd1.;
%mend makeme2;
%end;
%mend loop2;
%loop2;
```

REFERENCES

Kuligowski, Andrew T., “*Easy Come, Easy Go — Interactions Between The Data Step And External Files*” found at (<http://www.gasug.org/papers/index.htm>)

Cody, Ron, “*SAS Functions by Example*” Copyright 2004, SAS Institute Inc., Cary, NC

CONTACT INFORMATION

Denise A. Kruse

SAS Certified Advanced Programmer for SAS9

Alpharetta, GA 30022

E-mail: deniseakruse@gmail.com

Web: www.gasug.org

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration.

Other brand and product names are trademarks of their respective companies.