Paper BB-19-2014

Get Control of Your Input: Refer to Multiple Data Files Efficiently

Zhongping Zhai, Bloomington, IL

ABSTRACT

In SAS® programming, complex business rules may be imposed on data sourcing, such as locating multiple input data files and accessing file information (e.g., file size, modified date, number of rows). This paper compares several methods on referencing multiple external files and SAS® data sets so as to provide recommendations for developers at all levels to efficiently handle multiple data file reference requirements.

INTRODUCTION

Referencing multiple data files (external text files or SAS data sets) is a common task in SAS programming, especially for an ETL (Extract-Transform-Load) process. When integrating multiple data sources, complex business requirements are sometimes applied, such as storage location specifications (e.g., file name follows a certain pattern) and data selection criteria based on metadata of files (e.g., create/modification time, file size, owner, group). For example, there are business needs to combine all SAS data sets which were created over a certain time period by a specific user. Another example is to recursively read in all CSV files, which are stored in any subdirectory under a directory. In both of the examples, efficiency and resource usage should be taken into account when source data files are in large size.

SAS provides several procedures and functions that enable accessing multiple data files and file metadata. Each has its advantages and limitations. This paper illustrates applications of those methods so as to guide SAS developers to efficiently control data inputs. In the following sections, external file reference is first discussed, followed by referring to multiple SAS data sets. Unless otherwise stated, examples in this paper are based on Linux and SAS 9.3® Base.

REFER TO MULTIPLE EXTERNAL FILES

When reading multiple external files into a SAS data set, basically, there are two programming methods: (1) concatenate all external files first and then process the merged file; (2) get a list of files, process each file separately into SAS, and then merge all SAS data sets into one. Usually the second method is less efficient and it is only preferred when file information are required.

X COMMAND

This belongs to the first method. The following code illustrates how to read in both “marketing” and “sales” external files.

```
X "cat /data/marketing_*.dat /data/sales*.dat > /data/merged.dat";
data all;
    infile "/data/merged.dat";
    [--sas code to read in--];
run;
X "rm /data/merged.dat";
```

This method is a quick-and-dirty approach. It works if there is no space issue concerning the storage of the merged file, but no file information can be utilized in the DATA step.

FILENAME STATEMENT

(1) Referring to Multiple Files with FILENAME

With FILENAME statement, multiple external files can be referenced by a concatenated list of files or directories. Wildcards (“*”, “?” and “[]”) can be used for pattern matching. The code above can be rewritten with the FILENAME statement in the following way:

```
filename f "/data/marketing_*.dat, /history/sales*.dat";
data all;
    infile f <input options>;
    [--sas code to read in--];
run;
```

The codes above are more compact and more efficient, since there is no need to create a temporary data file. In addition, the “FILENAME=” INFILE option can be used to identify source name, which is very useful for data validation.

Note that the INFILE option “FIRSTOBS=” will not be applied to each file. It only works for the first file in reference. Therefore, additional coding effort is needed to eliminate “header” lines in each file. If the pattern defined in FILENAME statement matches no file, SAS will issue an error “Physical file does not exist”. To avoid this exception, the FEXIST function can be used to verify file existence.

(2) Read Multiple Compressed Files

FILENAME statement can also handle multiple compressed files by using PIPE option, which invokes a program outside SAS and redirects the program's output to SAS. This capability enables capturing multiple external data without creating an uncompressed file, so that a great deal of disk space and run time are saved. This is especially useful when reading in only first several lines (e.g. for data profiling). Below is an example:

```
filename f pipe "gzip –dc /data/web_log_*.gz /history/visit*.gz";
```

Herein, gzip option “-d” means “decompress”; “-c” is to keep original compressed files. In this example, using “FILENAME=” INFILE option cannot identify source compressed file name, since decompressed data are output to standard output.

(3) Get a List of Files

With a list of files, business rules related to file information can be implemented. Using FILENAME together with PIPE option, information of files can be obtained as much as an operating system can provide. In the following code, “ls” is used to get all “.dat” files in “/data”, and to save file attributes (file name, file size, modified time, time zone, user, group and permission) into a SAS data set.

```
filename dirList pipe 'ls --full-time /data/*.dat';
data dirList;
    infile dirList length=reclen;
    input file $varying200. reclen;
    permission=scan(file,1,"");
    if scan(file,2,"")=1;
    user=scan(file,3,"");
    group=scan(file,4,"");
    file_size_MB=round(scan(file,5,"")/1024/1024,1);
    modified_time=input(catx(" ",scan(file,6," "),scan(file,7,"")),anydtdtm.);
    time_zone=scan(file,8,"");
    file_name=scan(file,-1,"");
    format modified_time datetime19.;
    format file_size_MB comma9.;
run;
```

In addition, some other options and commands can be used to get more specific information. For example, “ls -t” can be used to sort files by time and “| tail -1” can be used to retrieve the most recent file. To list files recursively, “find” command is preferred over “ls -R”, since the output of “find” is much easier to manipulate. Suppose the scenario is to refer to all CSV file under directory “/data” recursively. The FILENAME statement can be written as:

```
filename dirList pipe "find /data -name \*.csv";
```

In fact, more options of “find” command can be used to search specific files, such as “-type” (file or directory), “-mtime” (modified time range) and “-size” (file size). With a list of files as a SAS data set, “FILEVAR=” INFILE can be used to refer to file name variable in the data set, so that multiple files can be processed in a single DATA step.

Note that PIPE option is system dependent. Below are examples for getting files in Windows (in folder “c:\test”). For more examples, please refer to Brian Varney (2012).

```
filename dirList pipe "dir /a-d c:\test";/*file attributes only, no sub-folder*/
filename dirList pipe "dir c:\test\*.sas /b /s";/*file name, recursively*/
```

USE DIRECTORY/FILE FUNCTIONS

SAS has a set of functions which are designed to work with directory and file, including locating files, iteration, and retrieving file information. All functions can be used in either DATA step or macro (using %sysfunc).

(1) Directory Functions

SAS directory functions are used to iterate members in a directory. They are:

- DOPEN(fileRef): opens a directory (defined by file reference statement/function) and returns a directory identifier value, which is used by other functions. If the directory could not be opened, DOPEN returns 0.
- DNUM(dirID): returns the number of members in a directory (dirID is the directory identifier).
- DREAD(dirID, memberID): returns the file name of a given directory identifier and memberID (sequence number of file/folder within the directory).
- DCLOSE(dirID): closes the directory.

(2) File Functions

File functions are used to display file information of a single file, as shown below.

- FOPEN(fileref): opens an external file (defined by file reference) and returns a file identifier value. Note that fid=0 means an error occurred when reading in a file. In programming, this can be used to distinguish file and directory.
- FINFO(fid,<options>): returns file information of file identifier fid. Some common options are:
  - Create Time: created time in text format of "ddmmmyyyy:hh:mm:ss" (for Windows)
  - Last Modified: last modified time as text. In Windows, shown as "ddmmmyyyy:hh:mm:ss"; in Unix, shown as "Www Mon dd hh:mm:ss yyyy".
  - FileName: file name in full path
  - File size (bytes): file size in bytes
  - Owner Name(for Unix)
  - Group Name(for Unix)
  - Access Permission (for UNIX only)
- FCLOSE(fid): closes an external file, directory, or directory member.

Below is an example of using directory and file functions to list all files that are larger than 1 GB under "/data/weblog" directory.

```
%let mydir=/data/weblog;
filename dirRef "&mydir";
data file_meta;
    dirID=dopen("dirRef");
    num_of_files=dnum(dirID);
    do i=1 to num_of_files;
        file_name=dread(dirID,i);
        rc1=filename("fRef",catx("/","&mydir",file_name));
        fid=fopen("fRef");
        if fid>0 then do;
            full_name=finfo(fid,"FileName");
            bytes=finfo(fid,"File Size (bytes)");
            moddate=finfo(fid,"Last Modified");
            owner=finfo(fid,"Owner Name");
            rc2=fclose(fid);
            if bytes/1024/1024/1024>1 then output;
        end;
    end;
run;
```

SUMMARY OF REFERENCING MULTIPLE FILES

The following table summarizes the methods discussed above. When no file information is needed, FILENAME with referencing multiple files (sometimes using wildcards) is preferred. If only file name is desired and there are filtering rules based on file attributes (such as file size, modified time and owner), FILENAME with "find" should be appropriate to get list of files. Both FILENAME statement and directory/file functions can retrieve file information, but dictionary/file function is easier to program.

Table 1. Methods for Reference External Files

Method	File Type	Recursive	File Name	File Size	Created/ Modified Time	Permission	Owner	Group
X command to join files	text							
FILENAME … "multiple files "	text		X					
FILENAME … PIPE "ls --full-time ..."	any		X	X	X	X	X	X
FILENAME … PIPE " find…"	any	X	X					
FILENAME … PIPE "gunzip -dc …"	.gz files							
Directory Functions	any		X					
File Functions	any			X	X	X	X	X

ACCESSING MULTIPLE SAS DATA SETS

Some of aforementioned methods work for any file, including SAS data. But SAS provides some special procedures and system objects that can be used to retrieve more specific file information.

ONE LIBRARY FOR MULTIPLE DIRECTORIES

LIBNAME statement allows joining multiple directories (or libraries) as a single library. This provides a convenient way to access multiple SAS data sets which are stored in different directories with a single DATA step or procedure. The basic syntax is:

```
libname myLib ("directory1", "directory2", … library1, library2 …);
```

Note that the rules for data input and output are different. When reading data from “myLib”, only the first matched data set is accessed when there are multiple data sets with the same name under different reference directories (or libraries), whereas for output, data sets are always written into the first directory (or library).

DICTIONARY TABLE

At data set level, one way to get a list of SAS data sets within a library is to query against DICTIONARY.TABLES (a table view in SAS). For example, the following code is to list all SAS data sets in SAS library SASHELP.

```
proc sql;
    create table all_datasets as
        select
            *
        from
            dictionary.tables
        where
            libname="SASHELP" and memtype="DATA";
quit;
```

This query returns 41 columns (see Table 2 for table description). The table holds almost all attributes about SAS data sets. Some of the attributes are frequently used, such as data name, member type, date created, date modified, number of observations, number of variables (numeric or character), size of file and number of logical observations.

Table 2. DICTIONARY.TABLE Fields

Field Name	Description		Field Name	Description
libname	Library Name		maxvar	Longest variable name
memname	Member Name		maxlabel	Longest label
memtype	Member Type		maxgen	Maximum number of generations
dbms_memtype	DBMS Member Type		gen	Generation number
memlabel	Data Set Label		attr	Data Set Attributes
typemem	Data Set Type		indxtype	Type of Indexes
crdate	Date Modified (Numeric)		datarep	Data Representation
modate	Date Modified (Numeric)		sortname	Name of Collating Sequence
nobs	Number of Physical Observations		sorttype	Sorting Type
obslen	Observation Length		sortchar	Charset Sorted By
nvar	Number of Variables		reqvector	Requirements Vector
protect	Type of Password Protection		datarepname	Data Representation Name
compress	Compression Routine		encoding	Data Encoding
encrypt	Encryption		audit	Audit Trail Active?
npage	Number of Pages		audit_before	Audit Before Image?
filesize	Size of File		audit_admin	Audit Admin Image?
pcompress	Percent Compression		audit_error	Audit Error Image?
reuse	Reuse Space		audit_data	Audit Data Image?
bufsize	Bufsize		num_character	Number of Character Variables
delobs	Number of Deleted Observations		num_numeric	Number of Numeric Variables
nlobs	Number of Logical Observations			

At variable level, DICTIONARY.COLUMNS contains detailed variable information for all SAS data sets. As an example, the following codes display tables/columns in library SASHELP which contain word “CITY” in variable name:

```
proc sql;
    select
        memname, name
```

```
    from
        dictionary.columns
    where
        libname="SASHELP" and name like "%CITY%";
quit;
```

Since a library may contain hundreds or thousands data sets, SQL optimization should be considered. As a tip, do not use functions in the WHERE statement, since it can deteriorate query performance.

PROC DATASETS

DATASETS procedure can also list library members and variable information of data sets. The information can be obtained from ODS tables:

- "Members": Library member information, such as data set name, member type (INDEX or DATA), file size (in bytes) and modified time.
- "Variables": A detailed listing of variables with type, length and position.
- "Attributes": Data set attributes, such as number of variables, number of observations, data label, etc.
- "EngineHost": Engine and operating environment information, including data set page size, release created, host operating system, etc.

For example, the following is to create a list of SAS data sets in library TEST:

```
ods output members=myMember;
proc Datasets lib=test memtype=data;
quit;
```

PROC CONTENTS

CONTENTS procedure functions the same as DATASETS procedure. The following code can be used to get data information, attributes and host information for all data in SASHELP library.

```
ods output members=myMember(where=(memtype="DATA"));
ods output Variables=myVar;
ods output Attributes=myAttr;
ods output Enginehost=myHost;
proc Contents data=sashelp._all_ nods;
run;
```

Herein, "nods" option is used for suppressing contents of individual files since "_all_" is specified.

ATTRIBUTE FUNCTIONS: ATTRN AND ATTRC

After obtaining a list of data sets, each of them can be processed separately. Attribute functions are handy for both DATA step and macro. ATTRN is used to retrieve numeric information about a SAS data set, while ATTNC is for character attributes. Below is a list of some commonly used attribute names:

- ATTRN(datasetID, attributeName)
  - NLOBS: Number of logical observations which are not marked for deletion
  - NDEL : Number of observations marked for deletion
  - NOBS : Number of total observations (including those marked for deletion)
  - CRDDTE: Creation time stamp of the data set
  - MODTE: Last modify date time
  - NVARS: Number of variables in the data set
  - INSINDEX: Whether or not the data set is indexed
- ATTRC (datasetID, attributeName)
  - LABEL: Data set label
  - SORTBY: Variables name by which data set is primarily sorted
  - MEM: Data set name
  - LIB: Current library for the data set

Herein, datasetID is created by OPEN function. Below is an example to define macro variables for number of logical observations and modified time with ATTRN function.

```
%let data_set=libname.data_set_name;
%let dsid=%sysfunc(open(&data_set));
%let nobs=%sysfunc(attrn(&dsid,nlobs));
%let modified_time=%sysfunc(putn(%sysfunc(attrn(&dsid, modte)),datetime18.));
%let close_flag=%sysfunc(close(&dsid));
```

OTHER METHODS TO GET NUMBER OF OBSERVATIONS

Generally, number of observations is used more often than other attributes. In the following DATA step, only descriptor portion of SASHELP.CLASS is read to get number of observations. But this may include observations that have been marked for deletion.

```
data _NULL_;
    if 0 then set sashelp.class nobs=n;
    call symputx("num_obs",n);
    stop;
run;
%put Number of Observations=&num_obs;
```

Another method is to use PROC SQL. Since it has to count all rows, it is not efficient for a large data set. But it does not include those marked for deletion.

```
proc sql;
    select count(*) into :num_obs from sashelp.class;
quit;
%put Number of Observations =&num_obs;
```

As a new feature in SAS® 9.3, automatic macro variable SYSNOBS can be used to get number of observations of the most recent data, which is closed by the previous procedure or DATA step. If the number of observations for the data set is not calculated by the previous procedure or DATA step, the value of SYSNOBS is set to -1. This is the simplest method to get number of observations of a SAS data set, given the data is used most recently.

SUMMARY OF SAS DATA SETS REFERENCE

The capability of each method discussed in the second section is summarized in Table 3. For DICTIONARY tables, PROC SQL optimizes the query and has better performance than Datasets and Contents. Also the resulted table is easier to manipulate than ODS tables from Datasets and Contents procedures. Therefore DICTIONARY is always preferred for multiple files referencing. But if file name is already obtained, attribute functions are handy in both DATA step and macro.

Table 3. Referring to SAS Data

Method	List data name	File size	# of rows	# of columns	Creation time	Modify time	Variables
Dictionary.Tables	X	X	X	X	X	X	
Dictionary.Columns	X						X
PROC CONTENTS	X	X	X	X	X	X	X
PROC DATASETS	X	X	X	X	X	X	X
ATTRN/ATTRC		X	X	X	X	X	
Data _NULL_			X				
SQL count			X				
&sysnobs			X				

CONCLUSION

This paper summarizes the methods of referring to multiple files and obtaining detailed file information. The two summary tables presented in this paper can server as a guideline for program development.

REFERENCE:

- Brian Varney, 2012, Inventory Your Files Using SAS®, SAS Global Forum 2012
- Megha Agarwal, 2012, The Power of “The FILENAME” Statement, WUSS 2012
- SAS Institute Inc. 2011. SAS® 9.3 Functions and CALL Routines: Reference. Cary, NC: SAS Institute Inc.
- SAS Institute Inc. 2012. Base SAS® 9.3 Procedures Guide, Second Edition. Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at zhaizhongping@hotmail.com.