Breaking Up Isn't Hard To Do After All

Toby Dunn, AMEDDC&S, Fort Sam Houston

Abstract

In the ever varying world of the SAS® programmer, there comes a time when either the specs or a DBGB (Dilbert Pointy Haired Boss) demands either a data set be broken into N number of data sets or individual reports. While there are many ways to achieve this either through hard coding or using the Macro language, the question remains "are these solutions the most efficient"? I say Nay, what follows in this paper is an easy one step solution for each of these problems. The solutions are designed to be both easy to read, write, and maintain, thus effectively minimizing the programmers job to merely writing a general solution for either problem.

Key Words: Hash, ODS, NewFile, FileVar, Call System

Introduction

Computer science theory teaches us proper data structure, more importantly it teaches that there is an optimal data structure given any language and associated problem to be solved. SAS is an I/O bound language first normal form with redundancies. This means that the data is held in one data set and processed as one data set. While this form is great for reducing processing resources as well as the complexity of the code, it fails to acknowledge that many times the end result must be multiple data sets or files.

The focus of this paper is to illustrate how a programmer can achieve the efficiency of preserving the data in one data set (while being processed), but effectively break the data up into N number data sets or files. Specifically, through the use of FileVar, NewFile and Hash objects, the user will be shown how one can create N number of files with one pass of the data.

The following is the code to generate the data set used in all examples:

```
Proc Summary
 Data = SASHELP.prdsal3 NWay ;
 Class Country State ;
 Output Out = Sums ( Drop = _Freq_ _Type_ )
              Sum  ( Actual ) = Sales ;
Run ;
```

FileVar

One of the simplest requests a programmer can be asked is to create N number of text files from a data set. Many programmers use macros that generate multiple data steps with basically the same code to accomplish this task. Not only is this approach a waste of processing time, but it adds extra complexity to the overall program. A simpler solution is to simply use the FileVar option. This is the same option that one uses when reading in files, except in reverse.

Example:

```
Data _Null_ ;
Set Sums ;
By Country ;

FileName = CatS( 'C:\Documents and Settings\Toby Dunn\Desktop\Test\'
                 , Country
                 , '.txt' ) ;

File Dummy FileVar = FileName ;

Put @1  Country
    @10 State
    @30 Sales Dollar20.2 ;

Run ;
```

In the above example the data set SASHELP.prdsal3 is summed using the Summary procedure. Utilizing Proc Summary with the Class statement provides the advantage of negating the necessity to first sort the data set. The resulting data set will be written out in sort order.

In the next segment of code, the entire path (including the output file name), is created and stored in the variable called FileName. The CatS function is used to do the concatenations as well as ensuring that unwanted spaces are trimmed. The filename is created and passed to the File statement. The File statement is given the name of Dummy, basically instructing SAS that the specific file to write out is dynamic and will be specified in the value given to the FileVar value.

As each observation is read into the data step the path is also created. Note that the name of the country from the data is also the destination file name. This process simplifies locating the final files to read in (actuality, any valid name for your operating system could have been used). Next, the country value is passed to the File statement and the file is generated, using the Put statement to write the data out to the final file. When the last record is reached the file is closed.

NewFile

ODS can also be used to create reports. The ODS printer destinations have an option that will generate a new report based on some preset starting point. When a starting point is reached a new file is created. The name of the new file will have a number as a suffix and the number will be incremented by 1 for each new file. For example, if the first file is called Myfile.RTF, subsequent filenames will be MyFile1.RTF, MyFile2.RTF, etc. The syntax is NewFile = < Starting Point>, where valid values of starting point are:

BYGROUP	starts a new file for the results of each BY group.
NONE	writes all output to the body file currently open.
OUTPUT/TABLE	starts a new body file for each output object.
PAGE	starts a new body file for each page of output. A page break occurs when a procedure explicitly starts a new page or when you start a new procedure (not because the page size was exceeded).
PROC	starts a new body file each time that you start a new procedure.

Example:

```
ODS Listing Close ;
ODS NoResults ;
ODS RTF File = "C:\Documents and Settings\Toby
Dunn\Desktop\Test\Country.RTF"   NewFile = ByGroup ;

Title1 "Sales Report" ;
Title2 "For" ;
Title3 "#ByVal1" ;

Proc Print
 Data = Sums NoObs ;
 By Country ;
 Var Country State Sales ;
Run ;

ODS RTF Close ;
ODS Listing ;
```

The above code uses the NewFile option with a value of ByGroup. This means that for each value of a ByGroup a new report will be produced. Since there are only three values for Country in the data set, the three files that will be produced are Country.RTF, Country1.RTf, and Country2.RTF. At this point we have essentially achieved our goal of producing multiple reports with only one pass of

the data. However, the file names are not very descriptive as to the contents of each. Thus, the code below solves this problem by renaming the files so that they will reflect different values in the ByGroup variable.

Example:

```
Data Country ;
Set Sums ( Keep = Country ) ;
By Country ;
If First.Country ;
Run ;


FileName ABC Pipe 'dir /B "C:\Documents and Settings\Toby
Dunn\Desktop\Test"' ;

Data Files ;
Infile ABC TruncOver ;
Input From $100. ;
Run ;


Data _Null_ ;
Set Files   ;
Set Country ;
Country = CatS( Country , ".RTF" ) ;

Call System( "CD C:\Documents and Settings\Toby Dunn\Desktop\Test" ) ;
Call System( CatX( " " , "Rename" , From , Country ) ) ;

Run ;
```

Hash

Sometimes a programmer has a legitimate need to produce multiple data sets. One way this can be automated is by the use of Hash Objects (introduced in version 9.1 of SAS). Hash objects read the base data into multiple data sets with only one pass of the data.

Hash Objects are run time executable code meaning their parameters can be values coming from variables in a data set. This process, coupled with the .Output method which writes the entire hash table to a data set, allows us to dynamically create data sets without having to specify data set names in the Data step code or write lengthy wall paper output code.

Since a full explanation of Hash Objects is well outside the scope of this paper, I will only explain what the example code does, however, I encourage you to explore hashing on your own.

Example:

```
Data _Null_ ;
Length VarName $ 32 ;

If _N_ = 1 Then Do ;
  DCL Hash Countrys ( Ordered: 'A' ) ;
  Countrys.DefineKey ( 'Country' , '_N_'  ) ;

  Open = Open( 'Sums' , 'I' ) ;

  Do I = 1 To AttrN( Open , 'NVars' ) ;
     VarName = VarName( Open , I ) ;
     Countrys.DefineData ( VarName ) ;
  End ;

  Close = Close( Open ) ;

  Countrys.DefineDone () ;
End ;


Do _N_ = 1 By 1 Until ( Last.Country ) ;
 Set Sums ;
 By Country ;
 Countrys.Add() ;
End ;

Countrys.Output ( DataSet: Compress( Country , '.' ) ) ;

Do _N_ = 1 To _N_ ;
  Countrys.Remove();
End ;

Run ;
```

The first step in this method of breaking a data set apart is to declare the hash table and instruct SAS what variable(s) to use as lookup keys _N_ is used to ensure uniqueness of values so that all values will be written out even if there are duplicate values of Country), this is done above by the following code:

```
DCL Hash Countrys ( Ordered: 'A' ) ;
Countrys.DefineKey ( 'Country' , '_N_' ) ;
```

Next, it is necessary to define what variables need to be written out to the data sets. To avoid writing out a lengthy list of variable names, a simple solution is implemented. The meta data is read in and looped through, adding each variable name to the *.DefineData* method, thus alleviating the programmer from having to know and type the variable names.

The next Do-Loop navigates through the data, searching for a ByGroup, and the *.Add* method adds the values for the variables defined in the *.DefineData*

method to the hash table. Once all the data for a particular ByGroup has been processed, the *.Output* method will build the new data set name from values of Country and write the data out to the data set. Finally, the *.Remove* method is used to delete the hash from memory thus freeing up memory.

Conclusion

The difference between theory and practice is that in theory everything works perfectly. So although in theory you should always strive to keep your data in one data set, sometimes you, the programmer, have no choice. This paper provided three different methods to efficiently create N number of files with only one pass through the data. It is my sincere hope that the next time you have to do this in practice your job will be a little bit easier.

Your comments and questions are valued and encouraged. Contact the author at:

Toby Dunn
AMEDDC&S
Fort Sam Houston. TX
E-mail: Toby.Dunn@amedd.army.mil

Join the SAS-L! @ listserv.uga.edu or Google Groups