

EFFICIENT USE OF DATA STORAGE WITH SAS®

ERIC WEBSTER, BRADFORD EXCHANGE

INTRODUCTION

The SAS System is an exceptionally flexible system for data access and analysis, with the capability of working with information in a wide variety of formats. It is very easy to write programs in SAS that read data and conduct analyses; unfortunately, it is almost as easy to write code that takes up more computer resources than necessary, particularly in the area of data storage. A few simple concepts can be all you need to make your programs fast, thrifty, and efficient!

EXTERNAL FILES

What Is An External File?

Data is generally stored in the SAS System in one of two ways: SAS datasets, or external files. A SAS dataset is a kind of special SAS-only file system that is very efficient for SAS to process, but unfortunately is not well suited for any other languages. If you are working in an environment where you are accessing data created by other systems (such as corporate databases, output of real-time data gathering systems, etc.) you are working with external files. There are many kinds of external files that you can use and/or create, but one of the most common is what is known as a "flat file". External files other than flat files include indexed/VSAM files and databases such as DB2 or Paradox.

How Can I Tell If My Program Is Using External Files Or SAS Datasets?

It is usually easy to tell if a program you are working with is using external files or SAS datasets by looking at the DATA step that reads in the data:

External File: DATA EXTERN;
 INFILE MYFILE;
 INPUT <variables>;

SAS Dataset: DATA SASSET;
 SET MYLIB.MYSET;

An external file is easy to spot in a DATA step by its tell-tale INFILE statement. If you have no DATA step, and are instead working with a program that uses a PROC immediately, you are definitely working with a SAS dataset.

How Do I Read Data From An External File?

The first thing to do with an external file is to tell SAS how to find the file you want to use, with an INFILE statement such as those below:

```
PC:  FILENAME MYFILE 'C:\PCFILE.DAT' ;
MVS  FILENAME MYFILE 'FRED.FLAT.FILE' ;
```

Both of the above tell SAS to associate a file with the name MYFILE. You can then use that name in an INFILE statement:

```
DATA MYDATA;
  INFILE MYFILE;
```

Tip: If you are reading a data file that has commas between each field (such as a .CSV file on the PC), use the DLM= option: INFILE MYFILE DLM=', '!

The next thing you need to do is to provide an INPUT statement that tells SAS where each field is:

```
DATA MYDATA;
  INFILE MYFILE;
  INPUT FNAME $10.0 LNAME $30.0 ADDRESS $30.0 CITY $20.0;
```

The INPUT statement specifies where the data is in the input file and what variables you want to hold each field. In this case, it also specifies the length and type of each data element.

Another type of INPUT statement that may be more convenient to use is called column-list format, where instead of the lengths of each variable, you specify the actual locations in the input file:

```
DATA MYDATA;
  INFILE MYFILE;
  INPUT FNAME $ 1-10 LNAME $ 11-40 ADDRESS $ 41-70 CITY $ 71-90;
```

You may encounter data that is formatted in a more complicated fashion than above. For example, say you had a file of rental data from a video store, and you had one record for every customer, with a field for every rental that customer has made (which could be a few or a lot). Your job is to create a dataset of customers who have rented "Snow White":

```
DATA MYDATA;
  INFILE MYFILE;
  INPUT CUSTNAME $30.0 RENTALS 4.0 @;
  DO I = 1 TO RENTALS;
    INPUT TITLE $20.0 @;
    IF TITLE = 'SNOW WHITE' THEN OUTPUT;
  END;
  INPUT;
  RUN;
```

The step reads in the customer's name and a numeric variable that specifies how many videos this customer has rented; because there is more data to read (the list of rentals), the "@" prevents SAS from going on to the next record. The DO statement then proceeds through the list of rentals, generating an output record for each customer who rented Snow White (the OUTPUT statement). The final INPUT "releases" the record so that the next customer can be processed.

All of these techniques have been concerned with reading external files, but all apply equally to writing them, using FILE instead of INFILE and PUT instead of INPUT.

Tip: If you have a variable number of elements to write out, use trailing @ to only write as many as you need, and save space!

For example, assume you have an external file of video rentals, each record of which contains the customer's name and the video rented, sorted by customer. Because the name is duplicated, space is wasted - instead, write out the name once, and have a variable number of rental records following. The more times the same customer has rented a video, the more space will be saved!

```
DATA RENTALS;
  INFILE RENTFILE;
  INPUT CUSTNAME $30.0 TITLE $20.0;
  RUN;

DATA _NULL_;
  ARRAY TITLES(500) $ TITLE1-TITLE500;
  RETAIN TITLE1-TITLE500 RENTNUM;
  SET RENTALS;
  BY CUSTNAME;
  IF FIRST.CUSTNAME THEN DO;
    RENTNUM = 0;
  END;
  RENTNUM + 1;
  TITLES(RENTNUM) = TITLE;
  IF LAST.CUSTNAME THEN DO;
    FILE NEWFILE;
    PUT CUSTNAME $30.0 RENTNUM 3.0 @;
    DO I = 1 TO RENTNUM; PUT TITLES(I) @; END;
    PUT;
  END;
  RUN;
```

SAS DATASETS

What Is A SAS Dataset?

The other method of storing data is in SAS datasets, which is the SAS "native" format. SAS datasets are stored in a "two-level" name, the first name showing the SAS library and the second the member in the library. The library is commonly specified with the LIBNAME statement; in PC SAS this corresponds to a DOS directory, and in MVS to a file with special dataset attributes (for those interested, DCB=(LRECL=6144,BLKSIZE=6144,RECFM=FS,DSORG=PS) on my system).

If you do not give a LIBNAME to a DATA step or PROC, a dataset will default to WORK, which SAS sets up for you when you start SAS (the SASWORK library). This means that everyone has created SAS datasets, though perhaps unknowingly!

```
DATA MYSET;
```

Is exactly the same as

```
DATA WORK.MYSET;
```

The difference is that the WORK library is deleted at the end of your session by default (this can be overridden by a combination of the WORKINIT and WORKTERM options). If you specify a LIBNAME and use that in your DATA step name, it creates a permanent SAS data set that can be used later:

```
LIBNAME MYLIB '<your library>';  
DATA MYLIB.MYSET;
```

To use the dataset, tell SAS about the library (with the LIBNAME statement) and reference the dataset in either a SET statement or a DATA= in a PROC:

```
LIBNAME MYLIB '<your library>';  
DATA MOREDATA;  
    SET MYLIB.MYSET;  
  
PROC MEANS DATA=MYLIB.MYSET;
```

You can have the output of a PROC directed to a permanent library if you wish:

```
LIBNAME MYLIB '<your library>';  
PROC FREQ DATA=MYLIB.INDATA;  
    TABLES MYVAR / OUT=MYLIB.FREQSET;
```

Why Should I Use Permanent Datasets?

SAS datasets can be very convenient for storing the output of very long jobs, so that you can have the summary results available later. They are also very nice for storing data so that PROC's can execute without an intervening DATA step each time - for example, say you had a set of data that represented a set of rentals from a video store. If you store this in a permanent SAS dataset, you can run PROC's against it at your leisure without re-running a DATA step:

```
LIBNAME MYLIB '<your library>';  
DATA MYLIB.RENTALS;  
    INFILE RENTDATA;  
    INPUT TITLE $20.0 RENTDATE MMDDYY8.0 RENTCOST 7.2 LATEFEE 7.2;  
    RUN;
```

You may want to explore this data over several days, each time without running the same DATA step again:

```
LIBNAME MYLIB '<your library>';
PROC MEANS DATA=MYLIB.RENTALS N MAX MIN SUM MEAN;
  VAR RENTCOST;
RUN;

PROC FREQ ORDER=FREQ DATA=MYLIB.RENTALS;
  TABLES TITLE;
RUN;

PROC MEANS DATA=MYLIB.RENTALS N MAX MIN MEAN SUM;
  CLASS TITLE;
  VAR RENTCOST LATEFEE;
RUN;
```

Each of these PROC's gives meaningful data back, and without the penalty of having to rerun your DATA step each time.

How Can I Keep Track Of My Permanent Datasets?

Permanent SAS datasets can sometimes be tricky to keep track of, especially on MVS where a dataset is implemented as a single file. There is a handy PROC, PROC CONTENTS, that will tell you what members are in your dataset. For example, the following is a PROC CONTENTS output for a directory in my SAS/Windows SAS installation:

```
CONTENTS PROCEDURE
-----Directory-----
Libref:          USELIB
Engine:          V608
Physical Name:   O:\SAS608\USAGE\SASHELP

#  Name      Memtype  Indexes
-----
1  COPYRGHT  DATA
2  FORMATS   CATALOG
3  SCREEN    CATALOG
4  SUPUTIL   DATA
5  USAGE     DATA
```

This first page shows that there are five datasets in my library USELIB, three of which are DATA sets and two of which are of type CATALOG (CATALOGs contain such things as key definitions, SAS/GRAPH® graphs, SAS/IML® matrices, etc.). The next is the detail data from one of these data sets (the variable list for this dataset is actually much longer, but this gives the flavor):

CONTENTS PROCEDURE

Data Set Name: USEL1B.USAGE	Observations:	168
Member Type: DATA	Variables:	34
Engine: V608	Indexes:	0
Created: 16:18 Wednesday, March 10, 1993	Observation Length:	1819
Last Modified: 16:18 Wednesday, March 10, 1993	Deleted Observations:	0
Protection:	Compressed:	NO
Data Set Type:	Sorted:	NO
Label:		

-----Engine/Host Dependent Information-----

Data Set Page Size:	14848
Number of Data Set Pages:	22
File Format:	607
First Data Page:	1
Max Obs per Page:	8
Obs in First Data Page:	5

-----Alphabetic List of Variables and Attributes-----

#	Variable	Type	Len	Pos	Format	Informat	Label
1	FIXED	Char	11	0			Release & Family when Fix was Shipped
8	KEYS	Char	200	54			Keywords to Aid in Locating SAS Note
9	MODULEN	Char	24	254			
10	PROD	Char	16	278			SAS Product Name or Identifier

With the aid of PROC CONTENTS, you can keep track of what datasets you have in each library, what variables they contain, when they were updated, etc. For actual manipulation of these datasets, see PROC DATASETS.

OTHER STORAGE TECHNIQUES

Automatic Number Conversion

There are some considerations to be made when deciding to use a permanent SAS dataset. Most importantly, SAS stores all numeric data internally as 8-byte numbers by default. For example, if you had a variable relating to a person's age, this would be stored in 8 bytes, when fewer would certainly suffice (since the maximum MVS 8-byte number of 72,057,594,037,927,935 would make someone older than the universe). In this case you would be safe telling SAS to use fewer than 8 bytes: two for MVS (maximum 255) or three for PC/Windows (maximum 8,192). Using the LENGTH statement can, as the SAS documentation frequently warns, result in loss of precision when storing numbers with decimals, but if your data represents integer data with a known maximum, the LENGTH statement can do wonders for saving space.

```
DATA CUSTS;
  LENGTH AGE 2.0 BIRTHDAY 3.0 INCOME 4.0 NUMKIDS 2.0;
  INFILE MYFILE;
  INPUT AGE BIRTHDAY INCOME NUMKIDS;
  IF NUMKIDS < 4 THEN <...>
```

This dataset will store an observation in 11 bytes, about one-third of the 32 bytes that would have been required if LENGTH hadn't been used.

Tip: If you have integer data (without decimal places) and you know the maximum the number can be, use LENGTH in permanent SAS datasets to save space!

There is another way to attack this problem if you are going to write out the fields without having to use the fields' "numberness" qualities: read numbers in as character variables. This is an all-around great efficiency technique for several reasons: (1) numbers read in as character are not reformatted internally, so even if data is in a binary format (such as MVS' IB4.0) it can be read and written safely; (2) numbers stored in character variables still compare correctly: if you INPUT NUM1 \$2.0 NUM2 \$2.0 and NUM1 = 25 and NUM2 = 15, NUM2 < NUM1 will be TRUE; (3) numbers read in as character are not expanded to 8-bytes, saving space; (4) the CPU time associated with converting numbers to the SAS 8-byte format is avoided. The above example can be rewritten as :

```
DATA CUSTS;  
  LENGTH AGE $2.0 BIRTHDAY $3.0 INCOME $4.0 NUMKIDS 2.0;  
  INFILE MYFILE;  
  INPUT AGE BIRTHDAY INCOME NUMKIDS;  
  IF NUMKIDS < 4 THEN <...>
```

We still read in NUMKIDS as numeric, since we need to use NUMKIDS' "numberness" qualities, but the rest are read in as character because they are not being used as such.

Tip: Whenever possible, read numeric fields as character to save space and CPU time (often a lot!)

The WORK Library

The WORK library, the one you use by default when you execute most DATA steps and PROC's that create datasets, is sort of a catch-all dataset that SAS manages for you during your session. Unless you specify otherwise, this library holds all DATA sets you create, FORMATS you create, etc. Sometimes you create a WORK dataset that you do not need to - when the dataset is only being created just to feed a single PROC, and when you need a DATA step only to write an external file or run CALL SYMPUT. You can avoid the first by using VIEW's, and the second with DATA _NULL_.

VIEW's

When you run a DATA step in SAS just to feed into a single PROC, you can save the temporary storage by using a VIEW:

```
DATA CUSTS / VIEW=CUSTS;  
  INFILE BIGFILE;  
  INPUT NAME $30.0 CITY $30.0 <etc.>;  
  RUN;  
  
PROC FREQ DATA=CUSTS;  
  TABLES CITY;  
  RUN;
```

If all you need to do is count customers by city, a non-VIEW data step will read the entire customer file, write out the name, city, etc. to WORK.CUSTS, and then the PROC will run. Even if your WORK library has enough space to essentially make a copy of the input file (an even bigger one if you let numeric variables expand to 8 bytes), the total job will have to pass the data twice, once to run the DATA step and once to run the PROC FREQ. If you use a VIEW as above, however, each time the DATA step generates an observation it is passed immediately to the PROC FREQ. This can save a tremendous amount of storage! VIEW's can be used anywhere a dataset can - even being stored permanently in a permanent dataset, or as a target in a later SET statement. VIEW's can be nested in VIEW's quite deeply - for example, two VIEW's being MERGE'd in a third view, passed to a PROC!

There is still no such thing as a free lunch, though - if you need the DATA step to feed into several PROC's, such as one PROC MEANS, one PROC FREQ, and one PROC CORR, the data will be passed three times, once for each PROC. If you are careful, though, DATA VIEWS can be extremely useful.

Tip: If your dataset is only passed once, use a DATA VIEW to save intermediate storage space during your job!

DATA _NULL_

If your DATA step is only running to write out an external file (or just to execute CALL SYMPUT), use DATA _NULL_ to run the DATA step without creating an intermediate dataset:

```
DATA _NULL_;  
  INFILE RENTALS;  
  INPUT TITLE $20.0 PRICE 7.2;  
  IF TITLE = 'SNOW WHITE' THEN DO;  
    FILE NEWFILE;  
    PUT TITLE $20.0 PRICE 7.2;  
  END;  
RUN;
```

Tip: If your DATA step is only being run to create an external file, or to run CALL SYMPUT, use DATA _NULL_ to save intermediate space!

Data Compression

With the release of SAS 6.06 and beyond, datasets have a powerful new option: compression. Permanent SAS datasets (including WORK) store observation as fixed-length records - every one is the same length. However, if you have a lot of data that repeats, such as character variables with a lot of spaces, SAS can store these fields as the base string plus a compression factor. For example, if you had a 30-byte field representing city name, the two representations would be

Uncompressed:	'CHICAGO'
Compressed:	'CHICAGO' (23)'

So that SAS would know that the field really contains a small string plus 23 spaces. This technique, if your data is a good candidate for compression, does exact a performance penalty: SAS expands each compressed field into its uncompressed version on each SET statement or each time a PROC reads the observation. Another drawback is that you cannot use direct observation number access, since it is no longer easy for SAS to tell where the records start and stop without actually going through all the data. You may find, however, that the savings in storage space outweigh the performance drawbacks, especially if your dataset is an archive and rarely accessed but of substantial size.

Eric Webster
Bradford Exchange
9333 Milwaukee Ave
Niles IL 60714
Compuserve: 72240,2510
Internet: 72240.2510@compuserve.com

® Indicates USA registration.