

SAS VSAM Processing for IBM Mainframe CICS Environments

David Thul

Datcom Business Solutions, Brooklyn Park, MN

Abstract

Most mainframers view Base SAS as a tool to use to write simple reports using flat files as input. It is the exception to find a shop that uses the tool as part of their production environment for not only report writing, but for updating files as well. One purpose of this paper to show easy it is to maximize SAS's power in your IBM CICS shop.

Introduction

The purpose of this paper is to show how to utilize The SAS System to read and manipulate IBM VSAM files. It will give code examples illustrating how to read IBM VSAM KSDS files (VSAM files from now on) sequentially and randomly as well as how to update, add and delete records. The paper will cite some everyday problems associated with this type of processing and how to solve or work around these problems. It is outside the scope of this paper to explain any of the technologies behind VSAM files. It is assumed that you have the necessary working knowledge of these types of files.

Sequential Read

Reading VSAM files sequentially is identical to reading flat files except you must include the VSAM parameter on the INFILE statement as follows:

```
INFILE filename VSAM;
```

Where *filename* is the DD name pointing to the VSAM file in the JCL.

Random Read (Full Key)

Of course, what good is it to use VSAM if all you are going to do is read sequentially. In order to use the technology the way it was intended, you want to read only the records you need then stop processing. This is possible by including more parameters with the INFILE statement as follows:

```
fback = 0;  
INFILE filename VSAM BUFND=buff1  
BUFNI=buff2 KEY=keyname  
FEEDBACK=fback;
```

Where *filename* is the DD name pointing to the VSAM file in the JCL, *buff1* is data buffers, *buff2* is index buffers. It is not necessary to manipulate these buffers, but doing so will add efficiency. I will not be putting it in any more examples in this paper. *Keyname* is the value of the primary key in the VSAM file that you are attempting to hit and *fback* is the name of the variable which stores the resulting feedback number. This type of read requires that you are able to build the full key before you attempt to read the VSAM file. If you were successful with your read, you will get a value of zero in the feedback variable. If the record you were attempting to read was not found, you will get a value of 16 in the feedback variable. It is a good idea to reset the feedback variable to zero before each read.

Random Read (Partial Key)

In a perfect world, you will always have the full key available to you. However, as often as not, you will have to read the VSAM file with a partial key. What SAS does in this

case is gets you to the first record that matches the partial key and you will need to read sequentially until the partial key changes. Here is the code:

```
INFILE filename VSAM KEY=keyname  
GENKEY SKIP FEEDBACK=fback;
```

The skip and feedback parameters tell SAS that the key is a partial key and to hit on the first record that matches that partial key. You will need code to read sequentially and check each record during the sequential read to see if it meets the criteria you are looking for.

For example, lets say you are given a list of customer Social Security Numbers and order dates and you need to search the VSAM order file for all orders for each customer on the list with an order date matching the date on the list. The order file is keyed by customer Social Security Number concatenated with an account trailer number so that a single person can have multiple accounts on the system. The date of the order is also on the file, but not part of the key. Here is some code you could use:

```
DATA hits;  
SET listfile(KEEP=ssn listdate);  
reckey=ssn;  
oldkey=reckey;  
fback = 0;  
INFILE filename VSAM KEY=reckey  
GENKEY SKIP FEEDBACK=fback;  
  
DO WHILE (reckey EQ oldkey AND fback  
EQ 0);  
    INPUT keyname  
           vsamdate  
           ...;  
    IF keyname EQ oldkey AND  
       vsamdate EQ listdate THEN  
        OUTPUT;  
END;
```

We are checking that *keyname* eq *oldkey* after the read to make sure we are still on the correct partial key. With the above set-up, it is possible to read one past the last record in the desired partial key group.

There is a catch here. You cannot use the partial key to travel backward through the VSAM file. Therefore, if your list contains several dates that need to be searched for the same customer and the VSAM file is not sorted in date order, you will have a problem when you try to read the same partial key for the second time. The second attempt will yield a feedback of 16 (no record found). This will force you to do one of two things. First, you could reset the key to low values before performing the second read. This will give you a hit on the second try. I have found this method to be prohibitively slow though. Alternatively you could keep all the records for a given SSN and add a step to merge this newly created file with the original driver file by SSN/date and keep all matches. This is the option I typically go with. Remember to comment the merge step because it will not be intuitive why this would be done.

Deleting a record

Because it is permanent, this is probably the scariest thing you will be asked to do. I usually reserve this for when I am using a test file to test a process which adds a record and I want to run the same key over and over again until I get it right. Having a delete job to run between tests is handy. On occasion, though, we are asked to remove production records. The code is extremely simple:

```
OPTION VSAMUPDATE;  
DATA _NULL_;  
erasevar = 1;  
INFILE filename VSAM  
KEY = erasekey  
ERASE=erasevar FEEDBACK=fback
```

Notice the similarity between this code and the code to read a record randomly. The only difference is the *erase* = *erasevar* argument. *Erasevar* is always set to 1 which

tells SAS we are going to delete the record. I tried to use a retain statement to set *erasevar* to 1 during compilation for sake of efficiency, but that wouldn't work. Remember to set the system option *vsamupdate* on whenever a record is being updated (deleting is considered updating).

Adding a record

```
OPTION VSAMUPDATE;  
DATA NULL;  
SET driver(KEEP=outrec);  
INFILE vsamfile VSAM;  
FILE vsamfile VSAM;  
PUT @001 outrec $CHAR100.;
```

This process will add a record to the *vsam* file. In this example, the data being added in the field *outrec* is being read from the SAS file called *driver*. The *outrec* data could have just as easily been read from a cards statement or a flat file, etc. While it is not intuitive why it is necessary to have the infile statement, it is needed in order for this process to work. The infile statement needs to point to the same file as the file statement. This is the file you will be adding to. Remember not to add a record that duplicates an already existing key or you will get a feedback code of 8.

Updating an existing record

```
OPTION VSAMUPDATE;  
_ERROR_=0;  
fback=0;  
INFILE vsamfile VSAM  
KEY=vsamkey FEEDBACK=fback;  
INPUT @001 all $CHAR100. ;  
FILE vsamfile VSAM;  
IF fback EQ 0 THEN DO;  
    PUT @001 all $CHAR100.  
        @024 newfield $CHAR08. ;  
END;
```

In this case, I am reading in the entire record and writing it back out then over-writing

newfield with a new value. Remember to put *newfield* after the existing data has been written. If you reverse this, you will overwrite the *newfield* with the old data.

Conclusion

I have found SAS to be an incredibly powerful tool for getting data quickly and efficiently to and from a CICS VSAM type environment. It would be overly simplistic to say that some of these techniques don't take a bit of practice. However, if you compare the amount of code required to do these steps in SAS vs COBOL, you should see the benefit.

Trademarks

SAS is a registered trademark of SAS Institute Inc.

Reference

SAS Institute, Inc.
SAS Guide to VSAM Processing, Version 6
First Edition
Cary, NC USA August, 1992

Contact Information

David Thul
Datcom Business Solutions, Inc.
9239 Telford Bay
Brooklyn Park, MN 55443
E-mail: dthul@aol.com