Paper 222-27

How to access VSAM FILE with SAS/AF® and SAS BASE

Claude Rhéaume, Desjardins Financial Security, Québec, Canada
Gilles Turgeon, Telus Business Solutions, Québec, Canada

ABSTRACT

The purpose of this paper is to illustrate some techniques to process VSAM files with SAS.

Topics covered are :

- Reading records with sequential and keyed direct access by exact key, approximate key or generic key.
- Effective use of INFILE statement options and related automatic variables.
- Adding, updating and erasing records.
- Using Data set views to access VSAM files in an interactive SAS/AF application.

INTRODUCTION

In this paper, we will present the different techniques that can be used to work with VSAM files. We will present an overview of the main topics covered by the SAS VSAM book

We will demonstrate, using a real life example, how to use the different options available in the INFILE and FILE statements.

A major point of interest is : a method of working with these kinds of files with SAS/AF, since SCL doesn't have specific functions to access a VSAM file with a key. You can use a SUBMIT CONTINUE to access this file but it's also possible to use SAS views with a FETCH statement in SAS/AF to work with these files as a SAS dataset, meaning that you can update, erase and add record through a SAS VIEW.

SETTING UP THE VSAM MASTER FILE

You can find all necessary information including the VSAM cluster definition, the sample data and the SAS program to load the file used in this paper in your SAS help screen version 8.2 under MVS.

SAS/BASE BATCH PROCESSING

The objective of this section are :

- To present the different techniques used to maintain VSAM files with SAS/BASE
- To understand the use of return code of each type of I/O operation as read, update, erase and add records in VSAM files.
- To illustrate some typical situations, a coding example is provided.

SETTING UP THE TRANSACTION FILE

We will begin with the creation of a transaction data set : this file contains an ID which is the primary key for the VSAM master file and a transaction type.

```
data keys (keep=id trtype) ;

  array ids {3} $9 ('194304428', '378462917',
                    '888888888') ;
  array trs {3} $3 ('UPD','DEL','ADD') ;
  do i = 1 to 3 ;
    id = ids{i} ;
    trtype= trs{i} ;
    output ;
  end ;

run ;
```

READING WITH SEQUENTIAL ACCESS

Objective : To demonstrate that it is no different to read a VSAM file or other flat file.

Comment : You can use any INFILE statement OPTION as usual like END= option.

Result: We retrieve all records from the VSAM file.

```
Data SEQKSDS ;

  Infile KSDSFILE End=Endfile ;
  Input @1  id       $9.
           @10 lname    $10.
           @20 fname    $10
           @30 address  $25.
           @55 city     $15.
           @70 state    $2.
           @72 zip      $5.
           @77 balc     $5.
           @82 gpa      $4.
           @86 class    $2.
           @88 hrs      $2
           @90 finaid   $1. ;
Run ;
```

DIRECT ACCESS WITH EXACT KEY

Objective : To demonstrate how to access a VSAM file with direct access.

Comment : Note the importance of verifying the return code which the feedback is assigned to. Also you must initialize the variable for the return code AND the automatic variable _ERROR_ to zero. Don't forget that the return code is assigned after the INPUT statement is executed.

Result : In our example we have one record that is not found. (Noticed by a return code 16) Two other records were found.

```
Data DIRECT ;

  Set KEYS ;

  Infile KSDSFILE Key=id Fdbk=retcode ;
  Input @ ;

  If retcode Ne 0 Then
  Do ;
    Put 'Record not found ' id= retcode= ;
    retcode = 0 ;
    _Error_ = 0 ;
  End ;
  Else Output ;

Run ;
```

SKIP SEQUENTIAL ACCESS WIITH GENERIC KEY

Objective : To demonstrate how to retrieve records with direct access given a partial key. We will use the first digit of the variable ID to retrieve all records whose ID begin with that digit.

Comment : Note again the importance of verifying the return code which the feedback option is assigned to.

You must initialize the variable for the return code AND the automatic variable _ERROR_ to zero.

Don't forget that the return code is assigned after the INPUT statement is executed.

You need a loop statement to obtain all records for a given generic key

We use a Do Until (0); meaning an infinite loop. When the key change or the return code is different of zero, we LEAVE the loop.

We also have the INPUT statement's SKIP option which is required in the loop's logic in order to move the file pointer to the next record.

Result : In our example we retrieve eleven records.

```
Data genkey ;

  Set KEYS ;
  Length genid $1 ;
  genid= id ;

  Do Until (0) ;

    Infile KSDSFILE Key=genid Genkey Skip
           Fdbk=retcode ;
    Input @01 idfile $1. @ ;

    If retcode Ne 0 Or idfile Ne genid Then
      Leave ;

    Input @1 id $9.
             name $20.
          ... ;
    Output ;

  End ;
  retcode = 0 ;
  _Error_ = 0 ;

Run ;
```

SIMPLE MASSIVE UPDATE EXAMPLE

Objective : To demonstrate how to modify a entire VSAM file.

Comment : The file is updated in place (we use the same FileRef for Infile and File statements)

Result : This data step program will initialize at zero the variable hours (columns 88-89)

```
Data _null_ ;

  Infile KSDSFILE ;
  Input ;

  File KSDSFILE ;
  Put @1  _infile_
      @88 '00' ;

Run ;
```

KEYED UPDATE EXAMPLE

Objective : We will apply many operations type in same step like updating, erasing and appending records.

Comment : The transaction file contains an ID and a transaction code. Depending on the transaction type we will use a new INFILE statement OPTION, ERASE=. This option, if set to 1, indicates that the record retrieved is deleted. At the next PUT statement

For the append operation, you must specify each field to initialize the record in the PUT statement.

For the UPDATE operation, you specify the position and the value of each field you want to replace in the next PUT statement.

You must then initialize the variable for the return code AND the automatic variable _ERROR_ to zero.

Result : In our example we modifed one record, we deleted another one and finally we appended a new record.

```
Data DIRECUPD ;

  Set KEYS ;

  genid= id ;
  varerase = 0 ;
  Infile KSDSFILE Key=genid Erase=varerase
                  Feedback=retcode ;
  Input ;

  If trtype In('DEL','UPD') And retcode Ne 0
Then
  Do ;
    Put 'You cannot update this record ' id '
record not found' ;
    retcode= 0 ;
    _Error_ = 0 ;
    Return ;
  End ;
  Else
    If trtype = 'ADD' Then
    Do ;
      _Error_ = 0 ;
      retcode = 0 ;
    End ;

  File KSDSFILE ;

  Select (trtype) ;
     When ('UPD') Put @01 _Infile_
                      @10 'Sugi27 Update
Vsam' ;
     When ('DEL') Do ;
                      varerase = 1 ;
                      Put _Infile_ ;
                  End ;
     When ('ADD') Put @01 Id 9.
                      @10 'Sugi'
                      @20 '27'
                      @30 'South'
                      @55 'Orlando'
                      @70 'Fl'
                      @72 '12345'
                      @77 '00050'
                      @82 '02,99'
                      @86 'WW'
                      @88 '24'
                      @90 'X' ;
     Otherwise ;
  End ;

Run ;
```

GENERIC KEY UPDATE EXAMPLE

Objective : To demonstrate how to update records with direct access, given a partial key. We will use the first digit of the Data Set variable ID to update all records whose ID begins with that digit.

Comment : Note again the importance of verifying the return code which the feedback option is assigned to.

You must initialize the variable for the return code AND the automatic variable _ERROR_ to zero.

Don't forget that the return code is assigned after the INPUT statement is executed.

You need a loop statement to obtain all records for given generic key

We will use a Do Until (0); meaning an infinite loop. When the key change or the return code is different of zero, we LEAVE the loop

We also use the INPUT statement's SKIP option which is required in the loop's logic to move the file pointer to the next record.

When we are in the same record key and the status is zero, you can modify the record retrieved with a combination of FILE and PUT statements.

Result : In our example we update eleven records.

```
Data UPDGEN ;

  Set KEYS ;
  Length genid $1 ;
  genid= id ;


  Do Until (0) ;

    Infile KSDSFILE Key=genid Genkey Skip
           Fdbk=retcode ;
    Input @01 idfile $1. @ ;

    File KSDSFILE ;
    If retcode Ne 0 Or idfile Ne genid Then
        Leave ;
    Else
        If trtype = 'UPD' Then
            Put @1  _infile_
                @88 '00' ;

    Input ;

  End ;
  retcode = 0 ;
  _Error_ = 0 ;

Run ;
```

SAS/AF ONLINE PROCESSING

The objective of this section are :

- First of all, we have to built a data step view to access a VSAM file. That view id designed to perform four actions: read, update, delete or add a record.
- Then, we present the screen design
- Finally we complete the application with the .PROGRAM SCL code.

THE KERNEL OF OUR SCL VSAM MANAGEMENT PROGRAM

- This view is use to communicate between the VSAM file and the screen variable through macro variables:
  - *Action* to perform
  - *Key* to retrieve
  - *Inf*ormation to update
  - *Rc* (return code)
- We use the INFILE statement's Options Key= to get the record with that exact key, Feedback= for the return code and finally Erase= to perform a delete action on the record
- After the INPUT, we analyze the return code versus the required action .
- Except for a simple read, the FILE and PUT statements are executed : the record is added, updated or deleted depending on the required action
- Finally the OUTPUT statement completes the view

```
Data KSDSVIEW /View=KSDSVIEW ;

  Length key $9 inf_new $90 ;

  key = Symget('key') ;
  action = Input(Symget('action'),1.) ;

  Infile KSDSFILE Vsam Key=key Feedback=rc
                Erase=vardel ;
  Input  @1  id       $9.
         @10 lname    $10.
         @20 fname    $10.
         @30 address  $25.
         @55 city     $15.
         @70 state    $2.
         @72 zip      $5.
         @77 balc     $5.
         @82 gpa      $4.
         @86 class    $2.
         @88 hrs      $2.
         @90 finaid   $1. ;
/* action : 0 ===> Read Record
         1 ===> Add Record
         2 ===> Delete
         3 ===> Update
*/
         rej = 0 ;
         Select (action) ;
         When (1) If rc = 0 Then rej = 1 ;
         Otherwise If rc Ne 0 Then rej = 1 ;
         End ;

         Call Symput('rc', Put(rej,4.)) ;
         rc= 0 ; _Error_ = 0 ;

         If Not rej And action ne 0 Then
         Do ;
         inf_new = Symget('info') ;
         File KSDSFILE ;

         If action = 2 Then vardel = 1 ;

         Put @01 inf_new ;
         End ;

         Output ;
         Stop ;
Run ;
```

VSAM FILE USER INTERFACE

- This is the screen design for the SCL program
  - Key section
  - Data Information section
  - Push button section
- First, the user fill the key field (the only unprotected field) and press Enter
- The program returns the data information or a message ; the key field is protected and data fields are unprotected.
- Then, the user can modify any data field and perform the required action through a push button.

```
        Sugi 27 KSDSFILE Interface


Key
  ID : __________

Data Information
  Last name     : ___________
  First name    : ___________
  Address       : ____________________
  City          : ________________
  State         : __
  Zip           : ______
  Balance       : ______
  GPA           : _____
  Class         : __
  Hrs         : __
  Finaid        : _

    (ADD )       (DELETE )    (UPDATE )
```

SCL CODE BEHIND THE VSAM FILE MANAGEMENT SCREEN

```
init :

   Control Label Enter ;

/*=========================================
•  Assign file and open it
•  Initialize variable
•  Protect Information data except the key
   field
========================================= */
   rc=Filename('KSDSFILE','XXX.KSDSFILE.TEST,
                  KSDS,'disp=old') ;

   If rc Then
      _Msg = "Problem to assign file " ;

   action = 0 ;
   Call symput('action',Put(action,1.)) ;

   Link Protect ;

Return ;

/*=========================================
Validation of the key field
•  Open the view and fetch a record (in fact,
   keyed INFILE and INPUT statements are
   executed on the VSAM file thought the
   view),
•  Process the return code,
•  Unprotect all data information fields
========================================= */

Valikey :

   temp_id = id ;
   call symput('key',ID) ;

   action = 0 ;
   Call symput('action',Put(action,1.)) ;

   KSDSVIEW = open('WORK.KSDSVIEW') ;
   If Not KSDSVIEW Then
      _Msg = "Problem to open file" ;
   call set(KSDSVIEW) ;
   rc = fetch(KSDSVIEW) ;

   rc = Symgetn('rc') ;
   If Not Modified(ADD) Then
      If rc Then
         _Msg = 'Record not found' ;
      Else
         If rc = 0 Then
            _Msg = 'Duplicate Key' ;

   Id = temp_id ;

   KSDSVIEW= Close(KSDSVIEW) ;

   Link Unprot ;

Return ;


/*=========================================
Main section
•  If any field is modified, fill the info
   macro variable to pass it to the view .
•  Issue a fetch on the data step view to
   perform the required action. Remember than
   the view will perform a FILE and PUT
   statement when an update type action is
   required .
========================================= */

Main :

   _Msg = _Msg_ ;

   If Modified(id) Then
      Link valikey ;

   Field_Modified = 0 ;
   If Modified(lname)   Or Modified(fname) Or
      Modified(address) Or Modified(city)  Or
      Modified(state)   Or Modified(zip)   Or
      Modified(balc)    Or Modified(gpa)   Or
      Modified(class)   Or Modified(hrs)   Or
      Modified(finaid) Then
      Field_Modified = 1 ;

   If Field_modified = 0 And
      (Modified(ADD) Or Modified(UPDATE))
   Then
      _Msg = 'No modified field' ;

   If Modified(DELETE) Or (Field_modified = 1
      And (Modified(ADD) Or
      Modified(UPDATE))) Then
   Do ;
      KSDSVIEW = open('WORK.KSDSVIEW') ;
      info = Put(id,$Char9.)        ||
             Put(lname,$Char10.)    ||
             Put(fname,$Char10.)    ||
             Put(address,$Char22.)  ||
             Put(city,$Char13.)     ||
             Put(state,$Char2.)     ||
             Put(zip,$Char5.)       ||
             Put(balc,$Char5.)      ||
             Put(gpa,$Char4.)       ||
             Put(class,$Char2.)     ||
             Put(hrs,$Char2.)       ||
             finaid ;
      Call symput('info',info) ;

      action = Modified(ADD) +
               2*Modified(DELETE) +
               3*Modified(UPDATE) ;
      Call symput('action',Put(action,1.))

      rc = Fetch(KSDSVIEW, 'Noset') ;
      rc = Symgetn('rc') ;
      If rc Then
          _Msg = 'Problem file error' ;
      Else
          _Msg = 'Update done' ;

      KSDSVIEW= Close(KSDSVIEW) ;
      Link Protect ;

   End ;
   Else
      If Not Modified(id) And
          _Msg =: 'Problem' Then
          Link Protect ;
      Else
          If Not Modified(id) And Not
            Field_modified Then
```

```
              Link Unprot ;
Return ;

Protect :

   Protect _All_ ;
   Unprotect id ;
   Cursor id ;

Return ;

Unprot :

   Unprotect _All_ ;
   Protect id _msg ;
   Cursor lname ;

Return ;

Term:

   If KSDSVIEW Then
      KSDSVIEW= Close(KSDSVIEW) ;
   rc= Filename('KSDSFILE', 'Clear');

Return ;
```

CONCLUSION

Since VSAM files are popular on MVS environment, it ‘s important to know all options and techniques available to work with those files.

In the first part, we cover some batch processing typical situations with SAS/BASE examples.

In the second part, we apply the same techniques but this time with a user interface to maintain any typical actions on the VSAM file again.

Readers interested in I/O performance should refer to the publications listed in the next section.

REFERENCES

SAS Institute, Inc.,
SAS Guide to VSAM processing, Version 8,
Cary, NC: SAS Institute Inc., 2000

Michael A. Raithel, WESTAT
Optimizing the Processing of VSAM Data Sets With The SAS System
SAS Institute, Inc.
Proceedings of the Twenty-Fourth Annual SAS Users Group International Conference, paper 302
Cary, NC: SAS Institute Inc, 1999

ACKNOWLEDGMENTS

We would like to thank David Turgeon for translation of this paper. The original text was written in French.

CONTACT INFORMATION

Please direct any questions or feedback to either of the authors at

Claude Rhéaume
Desjardins Financial Security
Québec
Canada
Email: claude.rheaume@djsfc.com

Gilles Turgeon
Telus Business Solutions
Québec
Canada
Email: gilles.turgeon@telus.com